Cyber Security Using Bayesian Attack Path Analysis

Remish Leonard Minz, Sanjana Pai Nagarmat, Ramesh Rakesh
Research & Development
Hitachi India Ltd.
Bangalore, India
Email: {remish.minz, sanjana, ramesh.rakesh}
@hitachi.co.in

Yoshiaki Isobe
Research & Development
Hitachi Ltd.
Yokohama, Japan
Email: yoshiaki.isobe.en
@hitachi.com

***Abstract*—Network security is gaining huge attention in today's world. Attack path analysis provides a comprehensive view of the attack surface for a network infrastructure, thereby assisting decision makers to choose better network protection strategies. Other than several deterministic methods to model the attack graphs, the uncertainty of attacks on the network infrastructure encourages probabilistic modeling which makes the Bayesian network a suitable model to represent the attack graph and to analyze the attack paths. Existing research focuses on representing the network topology into a Bayesian network model and uses a state-of-the-art algorithm to calculate the attack paths. However, practical issues concerning their scalability largely remain unaddressed. In this paper, we provide an efficient modeling mechanism for analyzing the attack paths in the network infrastructure using the Bayesian network. Our approach covers vulnerability identification, collection and mapping, semi-automatic attack graph creation and attack path visualization. In addition to this, we list the bottlenecks in the existing approaches and address some limitations in the existing Bayesian libraries. The details on how we have implemented our approach and conducted the attack path analysis on an enterprise network infrastructure are covered in this paper.**

Keywords–cybersecurity; Bayesian network; attack path analysis; Weka; Py-BBN.

I. INTRODUCTION

Recent cyber-attacks have made headlines due to their enormous impact on business [1]. This has drawn considerable attention to cybersecurity research. Organizations are using considerable resources to protect their network infrastructure. One methodology to protect the network infrastructure is to analyze all the possible paths in a network that an attack can take from an Internet-facing device of the network topology to a target device in the network. This methodology is called the attack path analysis. The representation of the network topology into the graph structure on which analysis is performed is called the *attack graph* [2]. Figure 1 shows an example network topology along with its attack graph. The network topology is shown on the left of the figure and the corresponding attack graph is shown on the right. Vulnerabilities in the devices are represented as nodes *v1*, ..., *v9* in the attack graph. All the paths from the Gateway to any other device in the graph represent attack paths that an attacker can take. Attack paths can be modeled in a deterministic manner to include fine-grain details of the network components as discussed in [2][3]. However, this approach blows up the attack graph making the attack path analysis an NP-Hard problem. In addition to that, the modeling approach misses to include the information about the attackers skill, the device targeted by the attacker, the know-how of the vulnerability used by the attacker to compromise the device. Such an uncertain environment encourages attack path analysis to be modeled in a probabilistic manner rather than a deterministic way. Along with the probabilistic approach, the graph structure of the network infrastructure makes the Bayesian network a suitable tool to model the attack graph and to perform attack path analysis.

Existing research on Bayesian network-based attack path analysis [4] focuses mostly on representation techniques of the network infrastructure. The security conditions of the network and the vulnerabilities in network services are modeled as nodes in the Bayesian network. Vulnerabilities in a device are considered the parent node of the security conditions node. These vulnerabilities are identified either by deploying agents in the network devices or by scanning the open ports of devices on which applications are executed. State of the art Junction Tree algorithm is used in [5] for attack path analysis. The network infrastructure used for attack path analysis in this reference is a synthetic example.

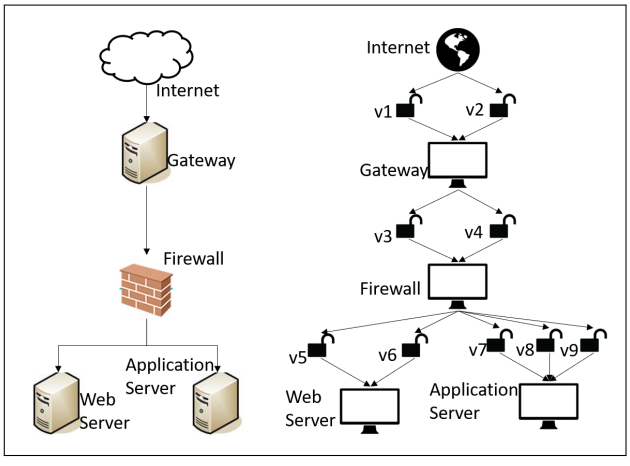


Figure 1. Network topology and attack path

Literature formalizes the modeling of the attack path using Bayesian semantics. However, some design and implementation issues concerning network scalability remain unaddressed. These include implementation concerns of modeling the attack path, assuming the existence of vulnerabilities in the attack graph, visualization as well as a need to qualitatively or quantitatively measure the usefulness of the attack path to the industry. The existing attack paths considered in the literature

 ISBN: 978-1-61208-683-5

are fine-grained causing the attack graph to eventually increase in size. Moreover, the existing attack path analysis is done on an example network topology, thus restricting the scale of the network considered.

The remaining part of this paper is organized as follows. Section II discusses the related work in the attack path analysis. Section III describes the details of the technical background which covers the attack path analysis, Bayesian modeling of the attack graph, vulnerability identification, and Bayesian attack path calculation. Section IV describes our approach of Bayesian graph modeling, vulnerability collection, Bayesian attack path calculation and visualization of the attack paths. Section V shows the evaluation of our approach. Section VI describes the conclusion of our approach and we end this paper by acknowledging our collaborating organizations.

II. Related Work

Early work on attack graph modeling can be found in [2][3]. They address the problem of early practice of manual attack graph generation by the Red Teams. Sheyner et al. [2] claim that automating generation of attack graph is exhaustive and succinct. Exhaustive means that the attack graph represents all possible attacks and succinct means that the attack path contains only those state which lies in the actual path of the attacker's goal. The network is modeled in a finite state machine and a model checker tool is used to automatically generate all the possible attack paths. The approach taken in it is symbolic model checking. A similar but scalable approach is taken by Ammann et al. [3]. In this approach again, model checking is used to automatically generate all the possible attack paths. They introduce an assumption on the attacker's behavior. The assumption states that the privilege of the attacker is monotonic in the target network as the attack progresses. This makes the modeling and analysis of attack path less complex. However, model checking approaches suffer from blow up of attack path and do not scale. Recent work in [4][5] uses a probabilistic approach. Both agree that inherent nondeterminism encourages the Bayesian network to be an ideal fit for modeling attack graphs. Frigault et al. [4] focus on modeling a static network infrastructure into a Bayesian network. They have come up with methodologies to model the vulnerabilities in the Bayesian network and ways to handle cycles in it. Further, they also discuss the scenario where the network infrastructure is dynamic. Munoz-Gonzalez et al. [5] discuss various exact inference techniques in the Bayesian network. They talk about, the Variable Elimination technique, Belief Propagation technique, and the Junction Tree technique. Here also, they discuss both static and dynamic network infrastructures. Vulnerability collection is another area of research where [6] evaluates several vulnerability scanning tools and shows that there is a lot of scope to improve the accuracy of the detection of vulnerabilities in both agent-based detection and port scan. A semi-automated way is tried by [7] to address this problem.

III. Technical Details

A. Attack path analysis

An attack path is a trail of vulnerabilities and devices formed from a sequence of attacks an attacker needs to perform on a given network topology, to explore and reach a target device. In the case of the network topology shown in Figure 1, if the attacker intends to target the application server, first, he needs to compromise the Gateway, as it is the only device in the network which can be accessed from the Internet. The attacker does this by exploring the presence of vulnerabilities *v1* and *v2* and exploiting any one of them to get into Gateway. Once the attacker has gained access to the Gateway, he needs to explore the next level of reachable devices in this network and find a way to gain access on those devices. In this example, it is the Firewall. In a similar fashion, the attacker moves step by step to reach the target device. At each step, the attacker needs to know the set of reachable devices in the next level. Once he gains access on a device, he runs a scan to list the reachable devices. After gaining the information of the reachable devices, the attacker finds the vulnerabilities in those devices. One of these vulnerabilities is exploited to gain access to one of the next reachable devices. Referring to the attack graph in Figure 1, an attacker having access to the Gateway can attack the Firewall by exploiting either of the two vulnerabilities *v3* or *v4*. Thus, an example of an attack path will be [Attacker] $\rightarrow$ *v2* $\rightarrow$ [Gateway Server] $\rightarrow$ *v3* $\rightarrow$ [Firewall] $\rightarrow$ *v8* $\rightarrow$ [Application Server].

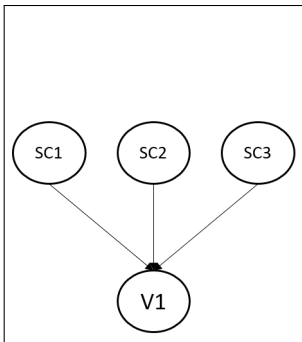
To protect the network infrastructure, a security operator needs to identify the vulnerabilities present in it. Once the vulnerabilities are identified, the list of vulnerabilities has to be mapped on the services and devices in the network. This comprehensive view also termed the *attack graph* [2][3] helps the operator understand the possible attack paths in the network. The *attack graph* is a standard way to express all the possible attacks from one external facing device in the network to the other potential target devices in the network. However, since information on aspects, such as the expertise of the attacker, vulnerability being exploited by the attacker to gain access to the next device, objective or target device of the attacker, etc. is unknown, a probabilistic approach to modeling the attack graph seems appropriate. Frigault et al. [4] and Munoz-Gonzalez et al. [5], apply the Bayesian network to model the attack graph and analyze the attack paths.

B. Modeling network topology to Bayesian graph

The Bayesian belief network is a probabilistic graphical model that represents a set of variables and their conditional dependencies via a Directed Acyclic Graph (DAG). The network components and the vulnerabilities are modeled as nodes while the edges represent how they are related to each other. Each node in the graph defines a causal relationship of itself with its parents. The step by step attack scenario shows how vulnerabilities have a causal relation among themselves. This makes the Bayesian network a suitable choice for analyzing the attack path. The causal relationship between a node and its parents is encoded in the conditional probability table. Frigault et al. [4] and Munoz-Gonzalez et al. [5], model Bayesian network by considering the attack graph like the one shown in Figure 1. They consider security conditions, where the conditional probability table is constructed using the vulnerability scores provided by the National Vulnerability Database (NVD) [8]. However, instead of devices in the attack graph, there are security conditions. The scores used in modeling the conditional probability table for the security conditions are set to 1.

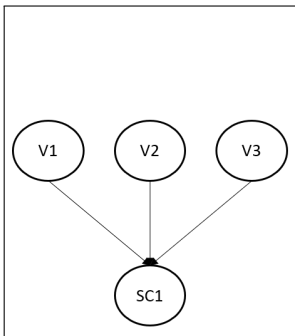
The semantics of the Bayesian model conveys that, for a vulnerability node to get exploited, all its parent's security conditions must be true. Additionally, for a security condition to be true, any one of its parent vulnerability should be true.

 ISBN: 978-1-61208-683-5



SC1	SC2	SC3	$V1^0$	$V1^1$
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	.27	.73

Figure 2. Conditional Probability Table: Conjunction



V1	V2	V3	$SC1^0$	$SC1^1$
0	0	0	1	0
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1

Figure 3. Conditional Probability Table: Disjunction

They call these two semantics as conjunction and disjunction. Figure 2 and Figure 3 show the conjunctive and the disjunctive causal relation between a node and its parents respectively. The disjunctive causal relation is shown for a security condition SC1 and the conjunctive causal relation is shown for a vulnerability V1. The conditional probability of the vulnerability V1 is 0.73. Each row in the conditional probability table shows the probability of the node being true given its parents. As an example, the second row in the table shown in Figure 2 shows variables SC1, SC2, and SC3 are assigned values 0, 0 and 1 respectively. Based on this assignment of the random variables SC1, SC2 and SC3, the probability of child variable V1=0, denoted by $V1^0$ is 1 and the probability of V1=1 denoted by $V1^1$ is 0. Being conjunction causal relation the exploitability score of 0.73 is used only in the case when all three variables SC1, SC2, and SC3 are assigned values 1, 1 and 1. In all other cases, the probability of V1 getting attacked is zero. Similarly, this probability distribution is called the prior probability which represents just the child-parent relationship. However, we need to know how an arbitrary target node is related to the root node in the attack graph. It shows how difficult it is to exploit the target node given that the root node is already compromised. Calculation of marginal probability does the same. Given an observation that a node (observed node) in the network is already compromised, we get the probability of the node (target node) getting exploited in the path from the observed node to the target node. This probability distribution of each node is called posterior probability.

C. Vulnerability Information

Assuming we have a list of vulnerabilities, we create a mapping between the vulnerabilities and services running in the network that contain these vulnerabilities. Vulnerabilities are found by matching the names of the software applications installed in the network devices against a list of vulnerable software in the NVD. This database of vulnerabilities is crowdsourced and maintained by the National Institute of Standards and Technology (NIST). Each reported vulnerability is examined by a team of experts and added to the database with relevant details. The details include an ID for the vulnerability, Description, Vulnerability Score, References, Technical details, Common Weakness Enumeration (CWE) and Common Platform Enumeration (CPE). An example of a vulnerability ID is CVE-2017-1300. Here CVE stands for Common Vulnerability Enumeration. CPE contains a machine-readable format of the reported vulnerable software application. This machine-readable format is called Well-Formed CPE Name (WFN). A CPE entry consists of colon separated values. An example CPE representing Microsoft Internet Explorer 8.0.6001 Beta is wfn[part="a", vendor="microsoft", product="internet_explorer", version=8.0.6001, update=beta]. The Unique Resource Identifier (URI) for the above WFN is cpe:/a:microsoft:internet_explorer:8.0.6001:beta. The names of the services or the software application executing in the network devices need to be matched with the product attribute of WFN.

D. Bayesian network based attack path analysis

There are several algorithms to calculate the posterior probability $p(X_i = x)$ from an attack graph having conditional probabilities for each node. Where X_i is a random variable in the Bayesian network. Belief Propagation and Variable Elimination are the algorithms used for calculating posterior probability for a target node. If X is the set of all random variables in the Bayesian network, $|X| = n$ and $Y = X - X_i$ then posterior probability $p(X_i = x)$ is given by,

$$p(X_i = x) = \sum_{Y} p(Y) = \sum_{Y} \prod_{k=1}^{n} p(X_k \mid X_{k_p})$$

where X_{k_p} is the parent nodes of node X_k. Calculation of posterior probability is an NP-Hard problem. Both Belief Propagation and Variable Elimination provide posterior probabilities for one target security condition. However, these two methods have to be executed for each target device to get all the possible attack paths. On the other hand, the Junction Tree algorithm provides a posterior probability for each target device. The Junction Tree algorithm is a method used in machine learning to extract marginalization in general graphs. It performs Belief Propagation (A message-passing algorithm for performing inference on graphical models, such as Bayesian networks. It calculates the marginal distribution for each unobserved node, conditional on any observed node) on a modified graph called the Junction Tree (In machine learning, tree decompositions are called Junction Trees, Clique Trees, or Join Trees). The steps performed by the Junction Tree algorithm on an attack graph with conditional probability tables for each node are as follows.

1) Graph Moralization
2) Introduction of Evidence
3) Graph Triangulation
4) Junction Tree Creation
5) Belief Propagation

The underlying functionality of this algorithm is summarized in the below steps:

1) Graph Moralization: If the graph is directed, then moralize it to make it undirected. A moral graph is used to find the equivalent undirected form of a DAG. An example is shown in the figure below. The moralized counterpart of a DAG is formed by adding edges between all the pairs of nodes that have a common child followed by making all edges in the graph undirected. An example of Graph Moralization is shown in Figure 4.

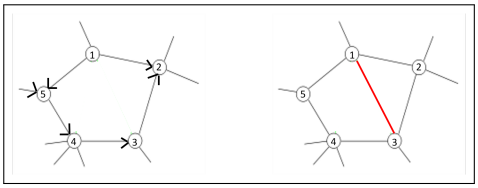


Figure 4. Graph Moralization

In the above graph, we convert the directed edges to undirected edges. Further, node 1 and 3 have a common child, node 2. Thus, in the equivalent moralized graph we introduce an undirected edge between the two parents node 1 and 3.

2) Introduction of Evidence: The second step is setting variables to their observed value. This shows the current event which has occurred. The posterior probability is conditioned on this observed random variable. These variables are also said to be clamped to their value. In case of attack path analysis, the root node of the graph is set as evidence.

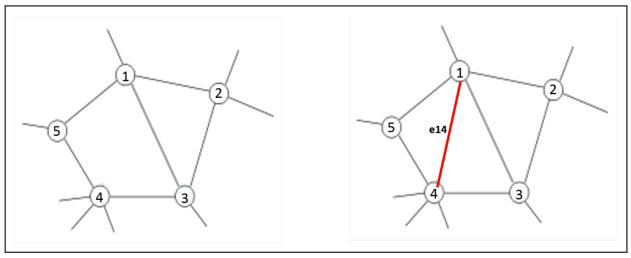


Figure 5. Graph Triangulation

3) Graph Triangulation: Triangulate the graph to make it chordal. The third step is to ensure that the graphs are made chordal if they aren't already chordal. A chordal graph is one in which all the cycles of four or more vertices have a chord, which is an edge that is not a part of the cycle instead connects two vertices of the cycle (A graph in which a cycle of length 4 and above must not exist) An example for graph triangulation is shown in Figure 5.
This graph can be triangulated in many ways. This will result in adding more edges to the initial graph, in such a way that the output will be a chordal graph. Edge (e14) is introduced such that the cycles of 4 or more vertices in the graph have a chord. Here, the cycle is formed by the node set {1, 3, 4, 5}.

4) Junction Tree Creation: Construct a Junction tree from the triangulated graph. The next step is to construct the Junction tree. To do so, we use the graph from the previous step and form its corresponding clique graph. Cliques are a subset of vertices of an undirected graph such that every two distinct vertices are adjacent and its induced subgraph is complete. An example of cliques in a triangulated graph is shown in Figure 6.

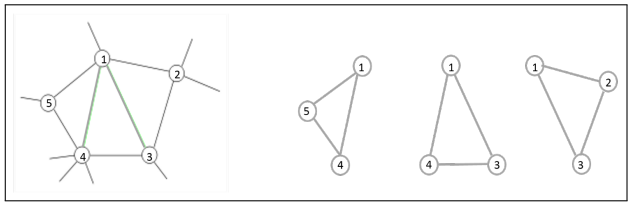


Figure 6. Cliques of Triangulated Graph

In this graph, there are three cliques {1,4,5}, {1,3,4} and {1,2,3}. Additionally, separator sets are sets of common nodes between the adjacent cliques. The total number of separator sets for a graph with n vertices is (n - 1). Therefore, in this case with 3 cliques, there are 2 separator sets {1,4} and {1,3} as shown in Figure 7.

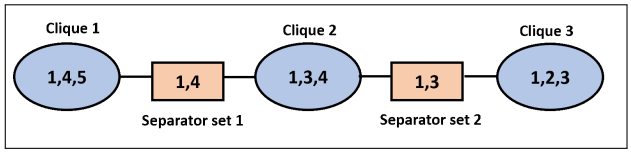


Figure 7. Junction Tree

5) Belief Propagation: Propagate the probabilities along the Junction tree using Belief Propagation algorithm, conditioned on all observed nodes. Marginal Probability for each unobserved node in the Junction Tree is calculated.

Junction Tree algorithm is used in [5] to calculate posterior probabilities which are further used for attack path analysis.

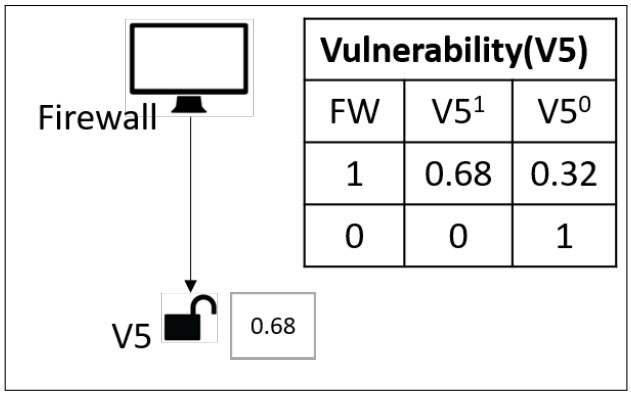
IV. Approach

A. Modeling Network topology to Bayesian graph

We performed attack path analysis and identified that the purpose of the analysis is to help an organization take an efficient approach to safeguard their network topology. This boils down to installing patches for the vulnerable software application or services in the right order. To fulfil this requirement, modeling the network devices as attack graph nodes is sufficient, compared to modeling the network services as attack graph nodes. This is because during the patching process, an entire device is patched at a time, mitigating all detected vulnerabilities in that device. This coarse-grained model provides an efficient and concise representation of the network topology. Therefore, to model the Bayesian attack graph from the given network topology, we first draw the network topology and make all the vulnerabilities in a device its parents. This is shown in the attack graph in Figure 1. The conditional probability tables for both vulnerabilities and devices are disjunctive. This is shown in Figure 8 and Figure 9. In the next subsection, we provide a semi-automatic method for the attack graph creation by manually specifying the topology and automatically collecting vulnerabilities.

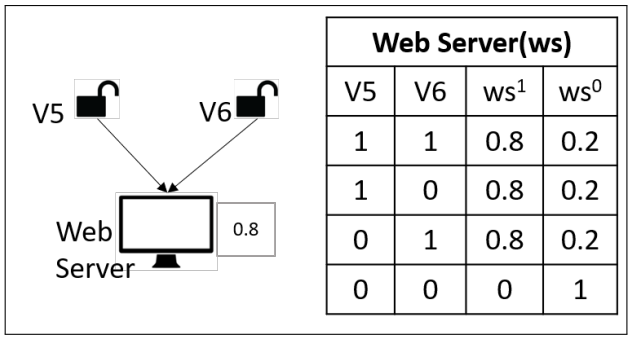
In Figure 8, as Firewall is the only parent of the vulnerability V5, the conditional probability table of V5 will have two rows. One parent can take only two possible values, 0 or 1 corresponding to each row in the table. As shown in Figure 9, conditional probability table of the Web server has four rows due to two parents V5 and V6.

 ISBN: 978-1-61208-683-5



Vulnerability(V5)		
FW	$V5^1$	$V5^0$
1	0.68	0.32
0	0	1

Figure 8. Conditional Probability Table: Vulnerability



Web Server(ws)			
V5	V6	ws^1	ws^0
1	1	0.8	0.2
1	0	0.8	0.2
0	1	0.8	0.2
0	0	0	1

Figure 9. Conditional Probability Table: Device

B. Vulnerability Collection

Attack path analysis requires topology information and vulnerability information. Figure 1 shows a simple network topology containing a Gateway server connected to the Internet and a Firewall attached to two servers below it. Once we have the topology information, we need to identify the vulnerability information for each device in the network. In this example, we collect software information for Gateway server, Firewall, Web server and Application server. This can be collected automatically, either by agent-based scripts or through port scans. In case of agent-based scripts, a script is executed on a privileged mode on each of the devices. This script collects names of all the software installed in the corresponding devices. It is an exhaustive list of installed software in the device. Once the list of the software is known, vulnerabilities associated with this software list are identified automatically. Since this approach provides an exhaustive list of software, the number of vulnerabilities identified is more.

Agent-based scripts will work only in a network where the operator has privileged access to each device. In the case of a port scan, a network scanning tool is used to scan the ports of each device. This scan provides information on all the open ports and applications running on them. This method only lists the applications currently being executing on an open port. Unlike the former method, this method does not perform an exhaustive check. However, it provides sufficient information of the applications which can be accessed externally. Compared to the agent-based approach, this approach is faster, as it only needs a network command to scan a list of IPs. The list of software names from each of the devices in the topology is automatically matched against a list of vulnerable software application reported in NVD in CPE format.

There is a standardization issue with the naming convention of the software applications. This is a challenge in automating vulnerability detection, as the product name in the CPE does not match with the names of the installed application. For example, in case of Internet Explorer, the application name in Windows registry is 'Internet Explorer'. However, there are two different entries for the same application (Internet Explorer) in the CPE dictionary. These are cpe:2.3:a:microsoft:internet_explorer:11:-:*:*:*:*:*:* and cpe:2.3:a:microsoft:ie:5.5:*:*:*:*:*:*:*. with product name 'internet_explorer' and 'ie' respectively. In this case, the application name does not match with any of the product names. A semi-automated approach is provided in [7]. On the other hand, we use vulnerability scanning software, Nessus [9] to address the challenge. Nessus reports vulnerabilities associated with the applications existing in the devices. Figure 1 shows the detected vulnerabilities in the example topology.

C. Bayesian attack path analysis

We use Weka's [10] implementation of the Junction Tree algorithm to automatically calculate the posterior probabilities of the nodes for attack path analysis. Weka is a generic suite of machine learning software based on JAVA. It provides Application Programming Interface (API) to model and query a Bayesian network. The methods addNode() and addArc() of the EditableBayesNet Class are used to add nodes and edges of the Bayesian network respectively. The prior probabilities in the conditional probability table are set using setDistribution() method of EditableBayesNet Class and the marginal probabilities are retrieved using calcMargin() method of MarginCalculator Class.

We have executed our approach on enterprise networks. A simpler scaled down network is modeled and discussed here. The topology of this simple network consists of one Firewall, four switches and four servers as shown in Figure 10.

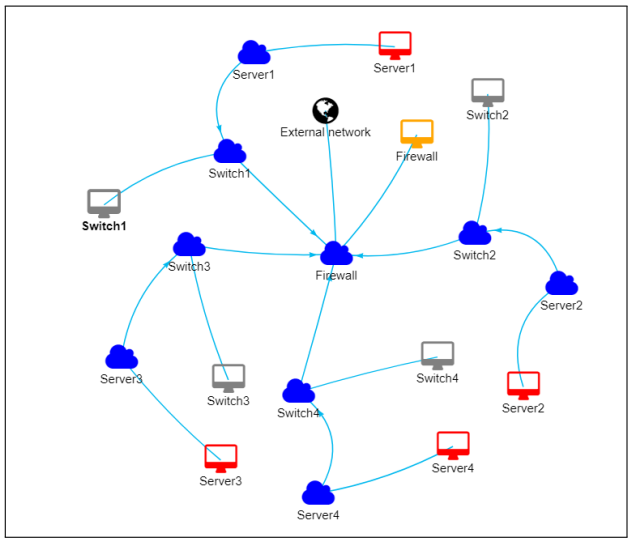


Figure 10. Topology of a sample enterprise network

The attack graph for the modeled network is shown in Figure 11. In this graph, the attack paths are created based on the existence of vulnerabilities in the devices. In the attack graph, there is no path leading to Server1 due to the absence of vulnerabilities in it. However, there are attack paths to other remaining servers.

In this example, in order to protect Server2 from external attacks, a security manager can analyze the attack path and then decide to mitigate the vulnerability in switch2. This will break the attack path from the external facing Firewall device. A security manager can further plan to patch the two vulnerabilities associated with Server2 during off-peak hours of the business. The vulnerabilities in the devices show either the Plugin ID of the vulnerabilities from the Nessus Database or the CVE ID from NVD.

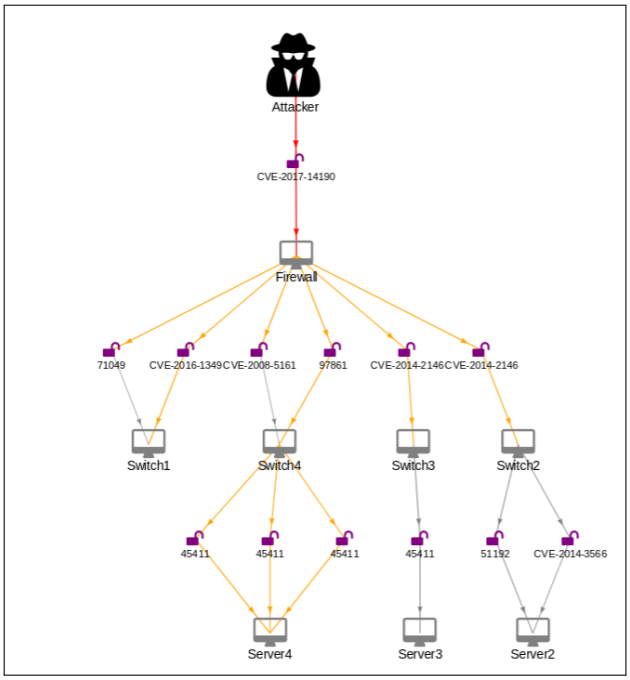


Figure 11. Extended attack graph

During the modeling of the attack path with a larger number of nodes, we identified some issues in Weka. Particularly in our case, when the number of vulnerabilities in the devices was greater than 600, Weka failed to provide appropriate output due to the underflow issue in the JAVA implementation. An underflow situation occurs due to the multiplication of a large number of probability values of the order of 10^{-4}. Particularly, this situation arises when in one level of the attack graph, any device has a large number of child nodes or vulnerabilities. This device along with its child nodes form one wide clique during marginalization. In the Junction Tree algorithm, the prior probability of each node in the clique is multiplied. This led to the generation of Not a Number (NAN) exception in JAVA. We addressed this underflow issue in Weka using a standard approach. Since the change was made in the Weka library, no larger datatype was adopted, neither the formula was converted to the logarithm, due to associated side effects. Rather, the variable collecting the product of a large number of probability values was initialized with a large value in the order of 10^{300}.

Weka holds a General Public License (GPL) license, which restricts the redistribution of the software without making it opensource. We addressed this restriction by replacing Weka with an alternate library, Py-BBN [11]. Py-BBN is an open-source Python implementation of the Junction Tree algorithm, whose implementation is similar to that of Weka. However, during the integration of our application with Py-BBN, we identified some issues in the generated attack path. Py-BBN failed to generate the output in some cases while in few others, it missed to include the leaf nodes or the terminal nodes in the generated attack graph.

On debugging the Py-BBN library, we identified the cause for the above issues and fixed the glitches in the underlying code. Reporting these identified issues along with their fixes in Weka and Py-BBN community for further improvement of the libraries are under progress.

D. Visualization

Maximum benefit from an attack path analysis can be achieved if a security manager can visualize the attack path and make appropriate decisions for securing the network topology. Thus, visualization plays an important role in the attack graph modeling and attack path analysis. We show our attack path in an Angular JS [12] based application, where the library used for rendering the attack graph is Vis.js [13]. Figure 10 and Figure 11 show the topology and the extended attack graph for the sample enterprise network respectively. However, the extended attack graph shown in Figure 11 becomes highly cluttered if the number of devices and the vulnerabilities associated with them is large. Due to the scaling issue associated with Vis.js, the browser times out on rendering attack graphs having nodes beyond 1500. To overcome the cluttered representation, we came up with a consolidated view of the attack graph as shown in Figure 12. Both the attack graphs show the same attack paths. However, all the vulnerabilities of a device are grouped into three categories in the consolidated view. These categories represented by red, yellow and grey colored vulnerability icons indicate high, medium and low severities respectively. Each vulnerability icon now shows the count of vulnerabilities along with the sum (cost) of the marginal probabilities of the vulnerabilities in that category.

V. Evaluation

Despite several benefits of modeling the network topology in a Bayesian network with either Weka or Py-BBN, there are a few challenges that remain unaddressed. Effective modeling of the network devices can only help in analyzing moderately sized networks. However, larger networks still cannot be analyzed with libraries like Weka and Py-BBN. Figure 13 shows how the average execution time changes with the number of nodes in the attack graph. The average execution time roughly grows in a linear manner with the number of nodes in the graph. We do not include the edges in the evaluation as the number of edges depends on the total number of vulnerabilities in the attack graph. Precisely, the number of edges is twice the number of vulnerabilities in the attack graph. As the number of vulnerabilities in each device is random, calculating the distribution is beyond the scope of this work. Further, we assume the graph to be sparse. Vis.js, on the other hand, has its own limitations in rendering graphs. Evaluation of its performance is shown in Figure 14. When the number of nodes reaches close to 500, the time taken to render the attack graph is approximately 50 seconds. Standard browsers like Mozilla and Chrome timeout after 28 seconds of execution.

VI. Conclusion

In this work, we modeled each device of the network infrastructure as a node in the attack graph for analysing the overall security of the network. On the other hand, existing

 ISBN: 978-1-61208-683-5

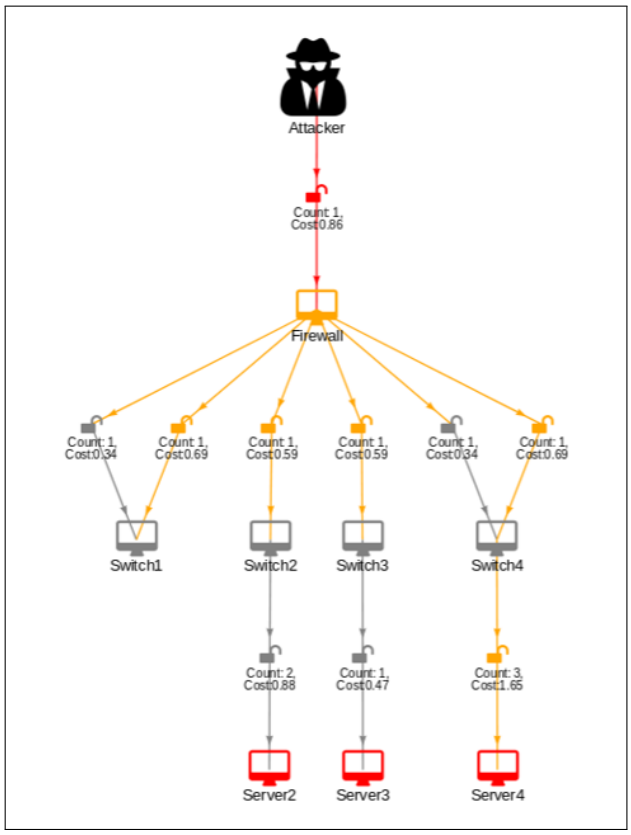


Figure 12. Consolidated view of attack graph

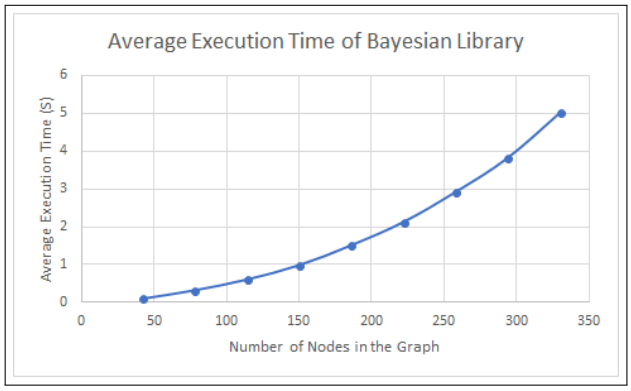


Figure 13. Performance of Py-BBN

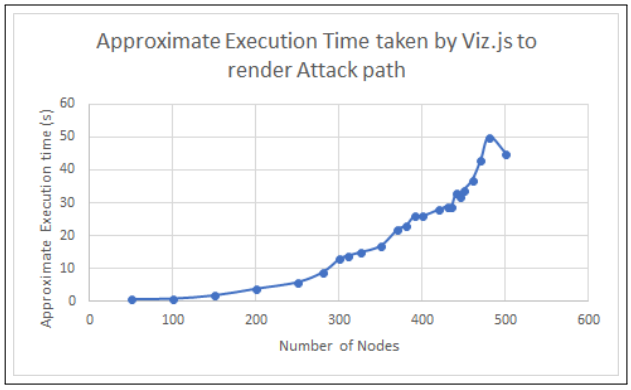


Figure 14. Performance of Vis.js

works model network services as nodes in the attack graph. Based on our evaluation we conclude that device level of modeling of the attack graph is both efficient and sufficient for attack path analysis. We say so as the number of network service is considerably greater than the number of devices in the network. Further, in this work, we identified the challenge in matching software application name with its corresponding product name in the NVD feeds. The existing work on attack path analysis does not discuss this issue. Benthin Sanguino et al. [7] provide a semi-automated approach to address this problem. Inspired by this semi-automated approach, we consider providing an automated approach to address this challenge in our future work. However, in this current work of attack path analysis, we use Nessus to address this challenge. We provide a semi-automatic approach for creating an attack graph by using device level modeling information and information on vulnerabilities collected using Nessus. We have also pointed out practical concerns affecting the scalability of the implementations of the Bayesian network. Scaling issue with Weka, which led to NAN and implementation specific bugs in Py-BBN were identified and fixed. We also plan to report the issues, along with their fixes to the open source communities for their further improvement. An attempt to evaluate the performance of Vis.js library was also made in this paper.

ACKNOWLEDGEMENT

This research is in collaboration with CTI (Center for Technology Innovation) Laboratory, Hitachi Ltd. Research and Development group, Yoshida-cho, Totsuka Yokohama, Japan.

REFERENCES

[1] "National Cyber Security Center," 2018, URL: https://www.ncsc.gov.uk/topics/cyber-attacks [retrieved: November, 2018].

[2] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in Proceedings 2002 IEEE Symposium on Security and Privacy May 12–15, 2002, Berkeley, CA, USA. IEEE, Explore Digital Library, May 2002, ISBN: 0-7695-1543-6, ISSN: 1081-6011, URL: https://ieeexplore.ieee.org/document/1004377/.

[3] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in Proceedings of the 9^{th} ACM conference on Computer and communication security(CCS'02) November 18–22, 2002, Washington, DC, USA. ACM, Nov. 2002, pp. 217–224, ISBN: 1-58113-612-9, URL: https://dl.acm.org/citation.cfm?id=586140.

[4] M. Frigault, L. Wang, S. Jajodia, and A. Singhal, "Measuring the Overall Network Security by Combining CVSS Scores Based on Attack Graphs and Bayesian Networks," in Network Security Metrics. Springer, Cham, Nov. 2017, chapter 1, pp. 1–23, in Network Security Metrics, Springer, Cham, ISBN: 978-3-319-66505-4,.

[5] L. Munoz-Gonzalez, D. Sgandurra, M. Barrere, and E. Lupu, "Exact Inference Techniques for the Analysis of Bayesian Attack Graphs," in IEEE Transactions on Dependable and Secure Computing. IEEE, 2017, pp. 1–1, in IEEE Transactions on Dependable and Secure Computing, ISSN: 1545-5971.

[6] H. Holm, T. Sommestad, and M. Persson, "A quantitative evaluation of vulnerability scanning," Information Management and Computer Security, vol. 19, 2011, pp. 231–247, ISSN: 0968-5227.

[7] L. A. B. Sanguino and R. Uetz, "Software Vulnerability Analysis Using CPE and CVE," in arXiv May 15, 2017, Cornell University Library, NY, USA. arXiv, May 2017, URL: https://arxiv.org/abs/1705.05347.

[8] "National Vulnerability Database," 2018, URL: https://www.nist.gov/programs-projects/national-vulnerability-database-nvd [retrieved: September, 2018].

[9] "tenable: Nessus, Professional," 2018, URL: https://www.tenable.com/products/nessus/nessus-professional [retrieved: September, 2018].

[10] "Weka," 2018, URL: https://www.cs.waikato.ac.nz/ml/weka/ [retrieved: November, 2018].

[11] "PyBBN," 2018, URL: https://github.com/vangj/py-bbn [retrieved: September, 2018].

[12] "AngularJS," 2018, URL: https://angularjs.org/ [retrieved: November, 2018].

[13] "AngularJS - VisJS," 2018, URL: https://github.com/visjs/angular-visjs [retrieved: September, 2018].

 ISBN: 978-1-61208-683-5