Bayesian Models for Node-Based Inference Techniques

Nazia Sharmin*, Shanto Roy†, Aron Laszka†, Jaime Acosta*, and Christopher Kiekintveld*
*Department of Computer Science
University of Texas at El Paso, El Paso, Texas, USA
†Department of Computer Science
University of Houston, Houston, Texas, USA

***Abstract*—Cyber attackers often use passive reconnaissance to collect information about target networks. This technique can be used to identify systems and plan attacks, making it an increasingly challenging task for security analysts to detect. Adversaries can recover statistical information from the information collected from compromised nodes, revealing target identities such as operating systems, software and servers. A comprehensive analysis of the collected data can aid in understanding what information an adversary can deduce from this technique. With this analysis, the defender can examine the methods of inferring a target used by adversaries and model adversaries' inference techniques and belief formation. For this purpose, we propose a model-driven decision support system (DSS) based on a Bayesian belief network (BBN) to depict adversary node-based inference techniques from passively collected data and belief formation. BBN provides a compact representation of probabilistic data and allows the formalization of adversary beliefs. We demonstrate this approach with a case study based on the passively observed operating system (OS) fingerprinting data, which is evaluated utilizing p-value significance level and compared against the model generated from local networks and predictive accuracy. We also show that our methods produce models with high predictive accuracy surpassing a sequential artificial neural network (ANN).**

***Index Terms*—Bayesian Belief Network, Adversarial Belief formation, Reconnaissance**

I. Introduction

In recent years, attackers have prioritized stealth to remain undetected by defenders and intrusion detection systems (IDS) in their target networks [1]. They employ a variety of methods to collect information passively, which is then leveraged to exploit various vulnerabilities. Existing methods, such as firewalls and intrusion detection systems, cannot secure a network completely against such attacks [2]. Data breaches by stealthy attackers can cause severe damage to organizations, and they take longer to identify and might not be detected at all in some cases. According to Ponemon's 2017 study, organizations that required over 30 days to identify a breach incurred 51% higher costs compared to organizations that identified breaches in under 30 days [3]. A delay in detecting adversarial reconnaissance can enable an adversary to gather sensitive information from a network. Therefore, the ability to mitigate the threats posed by these stealthy attackers has significant relevance for any organization.

Although much research has been conducted on attack graphs and the paths leading to target nodes, where sensitive information or potential vulnerabilities exist [4]–[6]; limited works are focusing on the inference technique from passively collected data from this node.

For example, Pham et al. [1] developed a reconnaissance model for stealthy adversaries, demonstrating that such attackers can conduct effective network reconnoitering by eavesdropping and capturing packets to infer a network topology graph based on their view of the network. Nonetheless, this study does not illustrate specific techniques used by attackers for deriving conclusions from collected information. As a network contains several nodes, protecting all of them is nearly impossible. The defender can select which nodes have vulnerable information and can build a model to determine features associated with the node that leads to identifying the target information for better defensive or deceptive strategies.

a) Contributions: In order to understand the adversaries' perspective, we examine how attackers use inference techniques to gain information from passively monitored traffic. In this paper, we explore the features associated with a node and build a model based on the data that adversaries can potentially collect in order to understand their perspective. We propose a decision support system (DSS) leveraging a Bayesian belief network (BBN). Our proposed DSS utilizes passively collected data associated with operating system features to construct and represent critical knowledge. We use the BBN to learn the data, analyze features and formalize the attacker's beliefs that can aid in OS identification. Various techniques for identifying operating systems have been studied in recent years. These include, leveraging characteristics of ICMP packets [7], using naïve Bayesian classifier to determine a host's OS passively and using various machine learning algorithms [8]–[10]. However, analysis and data interpretation are crucial for algorithmic decision-making and modeling attacker belief formation. We analyze the generated traffic based on OSes using the proposed DSS and perform inference based on the BBN to find the features that make it easy to distinguish one OS from another. For example, from our proposed DSS we observe TCP features of Mac and iOS are almost identical—thus, the use of TCP features significantly increases the probability of misclassifying iOS as Mac and vice versa [11]. A counterexample would be Android. Due to distinctive TCP features, it is easily distinguishable from other operating systems, including iOS, Mac, and Windows.

Thus, creating a detailed data-driven model can be useful for extracting sensitive information. To protect the network, we also utilize the attacker's belief in the model for better security countermeasures.

Our graphical models can be learned using a variety of passively collected network and traffic data. We use BBN to model the belief structure. Due to their scalability and noise resilience, BBNs are widely applied in data analysis. They have been commonly used in the literature to represent joint distributions compactly and to perform scalable methods for general inference [12], [13].

The paper makes the following contributions to the literature: it provides a methodology for building a BBN from real network data that can support a model-driven decision support system (DSS) capable of (i) analyzing variables/features and (ii) depicting the adversaries' belief formation and target inference processes. We present a case study in OS fingerprinting, but the methodology is intended for a wide variety of cyber reconnaissance such as network (or computing) infrastructure, software running, scanning tools. Section II provides a brief overview of the Decision Support System (DSS) framework and Bayesian Belief Networks (BBNs). Sections III and IV then demonstrate how decision-makers can build a BBN-driven DSS for adversaries' belief formation and target inference processes. This model is evaluated in Section V, followed by related work discussed in Section VI. Finally, the paper concludes with a summary and an outlook for future research opportunities in Section VII.

II. Decision Support System Framework

As the framework for DSS we develop a model-driven decision support system that allows users to build models from data. It has three components: a database module, model management module, and a dialog module.

1) **Database module:** Where data collection is performed by using tools and storing it in a database.
2) **Model management module**: We divide the model management module into two parts: (i) model parameter selection and (ii) model selection.
   (i) **Model parameter selection:** Decision-makers must first select a finite set of possible candidates, $P = \{p_1, p_2, \ldots, p_N\}$, for the decision-making problem. For instance, they need to decide which elements should be protected, such as an operating system, software, network, or combination thereof. Additionally, select criteria $C = \{c_1, c_2, \ldots, c_n\}$ in order to reflect the quality of each alternative. Herein, c_i denotes the i^{th} criterion with $i = \{1, 2, \ldots, L\}$. For instance, when it comes to OS information, criteria for feature selection may include TCP/IP network stack features, user agents HTTP, the combination of TCP/IP and HTTP or an information gain algorithm to select a subset of features. In this study, we have selected TCP/IP network stack feature selection as our criterion since it is easily collected passively. [2]. Models can then be built by selecting various criteria.
   (ii) **Model selection:** Once a modeler selects candidate and relevant information, they can construct models using the data in accordance with the problem and the practicality of implementation. Here, we employ a graphical model (i.e., a Bayesian belief network) that allows users to enter and modify inputs and access the data.
3) **Dialog module:** Once the model is generated, users can interact with the model and perform sensitivity or what-if analysis about the variables in the model. These queries can involve diagnostic reasoning (inference from effects to cause), causal reasoning (top-down inference), or inter-causal reasoning (given two mutually exclusive causes, evidence on one of them "explains away" the other one) as a means of gaining further understanding into the problem and prospective solutions.

A. Bayesian Belief Networks

A Bayesian belief network (BBN) is a pair (G, O) that encodes a joint probability distribution over a finite set $X = \{X_1, ..., X_n\}$ of categorical variables. It consists of a directed acyclic graph (DAG) $G = \{V, E, \Theta\}$, in which nodes V represent the variables in X, and edges E represent direct dependencies between variables, as well as a collection of conditional probability mass $\theta \in \Theta$ for each node. These conditional probabilities define the behavior of each variable, X_i, given its parents, $\prod_i$, in the graph [14]. The Bayes network follows the equation:

$$P(X_1 = x_1, X_2 = x_2, \ldots X_n = x_n) = \prod_i^n P(X_i = x_i \,|\, \textit{Parents}(X_i))$$

where $\textit{Parents}(X_i)$ denotes the parents of the node X_i, in the graph. The representation of the full joint table, $P(X)$, requires an exponential number of variables. This complexity can be avoided by using the Markov condition, which states that in a Bayesian network, every variable is conditionally independent of its non-descendant non-parents, given its parents. Using this condition, the joint table can be expressed in a compact form as the product of local mass functions.

B. Optimal Structure and Parameter Learning

We can learn a BBN's graphical structure and parameters from data. We aim to construct a methodology to represent cyber reconnaissance information using a knowledge graph to develop a DSS based on the data. Consider a data set D of d data points $D = \{x_1, x_2, x_3, x_4, x_5, \ldots, x_d\}$, we first determine variables to include in the graph. Then we determine the structure that best describes the joint probability distribution over the data set. Usually, without expert knowledge, we can use search-and-score methods to construct BBN's [15]. We learn the structure of graph G based on the data D, $\Pr(\mathcal{G} \mid D)$. Finally, we extract a specific number of

observations to examine the parameters, given the DAG and the data $\Pr(\Theta \mid \mathcal{G}, D)$. This is given by:

$$Pr(\Theta \mid \mathcal{G}, D) = \prod_{i=1}^{N} Pr(\Theta_{Xi} \mid \Pi_{pa(Xi)})) \tag{1}$$

where $\Pi_{pa(Xi)}$ represents all possible combinations of parents of X_i Combining the previous structure learning, we obtain the following:

$$\underbrace{\Pr(\mathcal{G}, \Theta \mid \ D)}_{\text{learning}} = \underbrace{\Pr(\mathcal{G} \mid \ D)}_{\text{structure learning}} . \underbrace{\Pr(\Theta \mid \mathcal{G}, D)}_{\text{parameter learning}} \tag{2}$$

III. Methodology: Bayesian Belief Network

We intend to build a Bayesian belief network (BBN) model that will capture (1) the causal relationships between random variables that take values from the collected data and (2) a formalism for reasoning about beliefs under conditions of uncertainty. We begin with a dataset description followed by describing the methods for constructing a BBN from network traffic data that support the above from the collected passive information. We describe general techniques for building a BBN from network traffic data in multiple steps: (1) dimension reduction, (2) discretization of variables, (3) model selection and (4) inference.

A. Dataset Description

We use a publicly accessible PCAP file with detailed network traffic data CIDSS [16], including data useful for passive OS fingerprinting. We evaluate our model on a subset of the dataset, for which we convert the CIDSS PCAP file to CSV using TShark, a network protocol analyzer. It can read packets from a previously saved capture file and convert the decoded form of those packets to the standard output or write the packets to a file. We selected 10.7 million TCP packets from the 11.7 million records in the CSV file.

B. Dimensional Reduction and Feature Selection

Dimension reduction means reducing the number of factors or parameters requiring estimation; dimension reduction can eliminate some irrelevant and/or redundant dimensions. Some reduction techniques are feature analysis and cluster analysis. For dimension reduction, we consider feature analysis specific to operating system types (e.g., Windows, Mac, and Linux) and not operating system versions. For this case, the TCP/IP features are most suitable. We select features based on the literature and the features utilized by operating system identification tools [9], [17]. The selected features from the pcap file are time to live (ttl), TCP window scaling option (op_wscale), window size value (win_size), header length (hdr_len), window scaling factor (wscf), maximum segment size (mss), data fragment bit (flag.df), SACK permitted (SACK), No Option (nop). Further details of the features can be found in [18].

Feature Extension and Wireshark Analysis: The binary values of certain features may not be suitable for use as node variables. An example is the analysis of TCP packets with Wireshark, in which a value of one for nop can indicate that a nop exists in the packet. Depending on other TCP options present, such as SACK permitted (SACK), window scaling factor (wscf), maximum segment size (mss), and timestamp, more than one nop may be observed in a single packet. To account for this complexity, we have created a new feature called tcp_comb to represent combinations of TCP options observed. This feature assigns one value from array A containing sixteen possible combinations based on whether or not SACK, wscf, mss and timestamp are present in a packet. We use s, m, w, t to denote these four features: SACK permitted (SACK), maximum segment size (mss), window scaling factor (wscf), and timestamp. Sign + and - indicate value observed or not observed in the packet, respectively. Array A includes [+s+m+w+t, +s-m+w+t, +s+m-w+t, +s-m-w+t,+s+m+w-t,+s-m+w-t, +s+m-w-t,+s- m-w-t,-s+m+w+t, -s-m+w+t, -s +m-w+t , -s-m-w+t, -s+m+w-t, -s-m+w-t, s+m-w-t, -s-m-w-t], sign +,- indicate s, m, w, and t value observed or not observed.

Wireshark Analysis: We observed the feature TCP window scaling option from Wireshark, which identifies the scaling factor to be used when using window sizes larger than 64k. We found identifier contains unique values; we observe the Wireshark packet and the value usually the following by analyzing TCP packets from various operating systems from different networks op_wscale = [30301, 30302, 30303, 30304, 30305, 30306, 30307, 30308, 30309, 3030a, 3030b, 3030c, 3030d, 3030e]. As a results we consider TCP window scaling option as a feature (op_wscale).

Missing States: If collected information contains missing values and requires discretization, modelers need to handle it prior to building the model. Our dataset contains a low percentage of missing values. We use standard imputations such as median and constant imputation corresponding to OS to handle missing values for window size and ttl. For other features, if data contains missing values we impute with zero.

Discretization: We use equal bins of the following ranges for window size: (-66.0, 13107.0], (13107.0, 26214.0], (26214.0, 39321.0], (39321.0, 52428.0] and (52428.0, 65535.0]. We replace -66 with -1. Note that the range starts from -1 but is excluded as window sizes cannot be negative

C. Model Selection

Once preprocessing is complete, modelers need to select an appropriate model. Heuristic search algorithms such as simulated annealing, greedy search algorithms and genetic algorithms are often used for uncovering the structure and parameter estimates of the Bayesian belief network (BBN). However, it has been observed that these search algorithms may not work well when there is insufficient or noisy data [19].

We create the Bayesian network using the following steps.

1) **Manual Construction and Expert Knowledge:** We manually construct a dependency graph using literature, wireshark analysis, chi-square independent test and significance level to connect nodes with edges according to direct dependencies. For example, the Time-To-Live

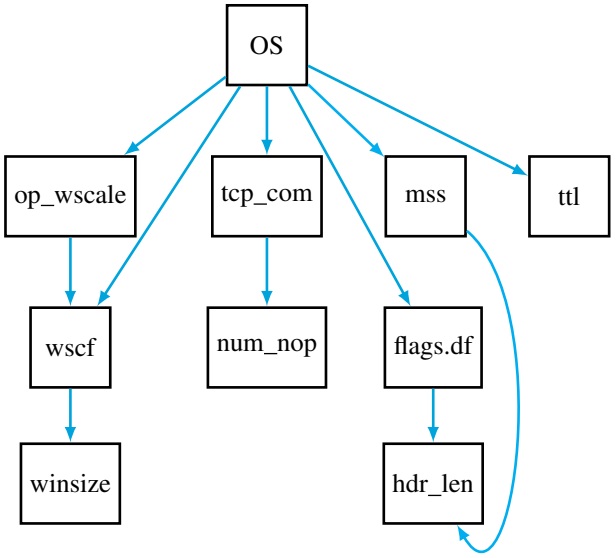


Fig. 1. Bayesian network

(TTL) is specific to an Operating System (OS), and thus directly dependent on it [20]. Additionally, window scaling factor corresponds to the size of the window as it resolves scalability issues with receiver window, advertised window or user buffer without increasing its size [20]. Furthermore, we use header length as a child node of maximum segment size (MSS). When the transport layer receives payloads, MSS can be observed and different header lengths may result in this observation [21].

2) **Wireshark Analysis:** We analyzed the number of NOPs observed with Wireshark and found that it varied depending on the number of TCP options present. As a consequence of this finding, we have included the number of NOPs as a child of the TCP combination (tcp_comb).
3) **Correlation test and structure validation:** We performed a correlation test and chi-square independent test and compare the structure with PC algorithm-generated structure to validate the BBN structure; details are provided in the evaluation section V. Fig. 1 illustrates the Bayesian network generated with these criteria, and (Fig. 2 shows its corresponding probability distribution. We use pgmpy library [22] to generate the CPT of the constructed Bayesian network using Bayesian Estimator. The Bayesian estimator does not solely depend on input data to learn the network parameters; it also utilizes prior knowledge, which is expressed through a prior distribution. This gives the estimator a reasonable starting point rather than an absolute guide in order to make up for any lack of data [23]. There are several prior distribuitions available in pgmpy library, we select the Bayesian Dirichlet equivalent uniform prior (BDeu) due to its key properties of having a uniform prior over the parameters of each local distribution in the network. This makes structure learning computationally efficient and prior knowledge from experts.

D. Inference: VE algorithm

We use a variable elimination algorithm inference using the proposed BBN. The variable elimination algorithm acts on a set of factors. The conditional probability distributions of the network comprise the first set of variables (usually tables). Elimination is triggered by a variable-by-variable ordering called the elimination order. Two factor operations are repeated several times in the algorithm—variables are multiplied and a variable is summed out of a factor. The detailed algorithm for variable elimination is given in [24].

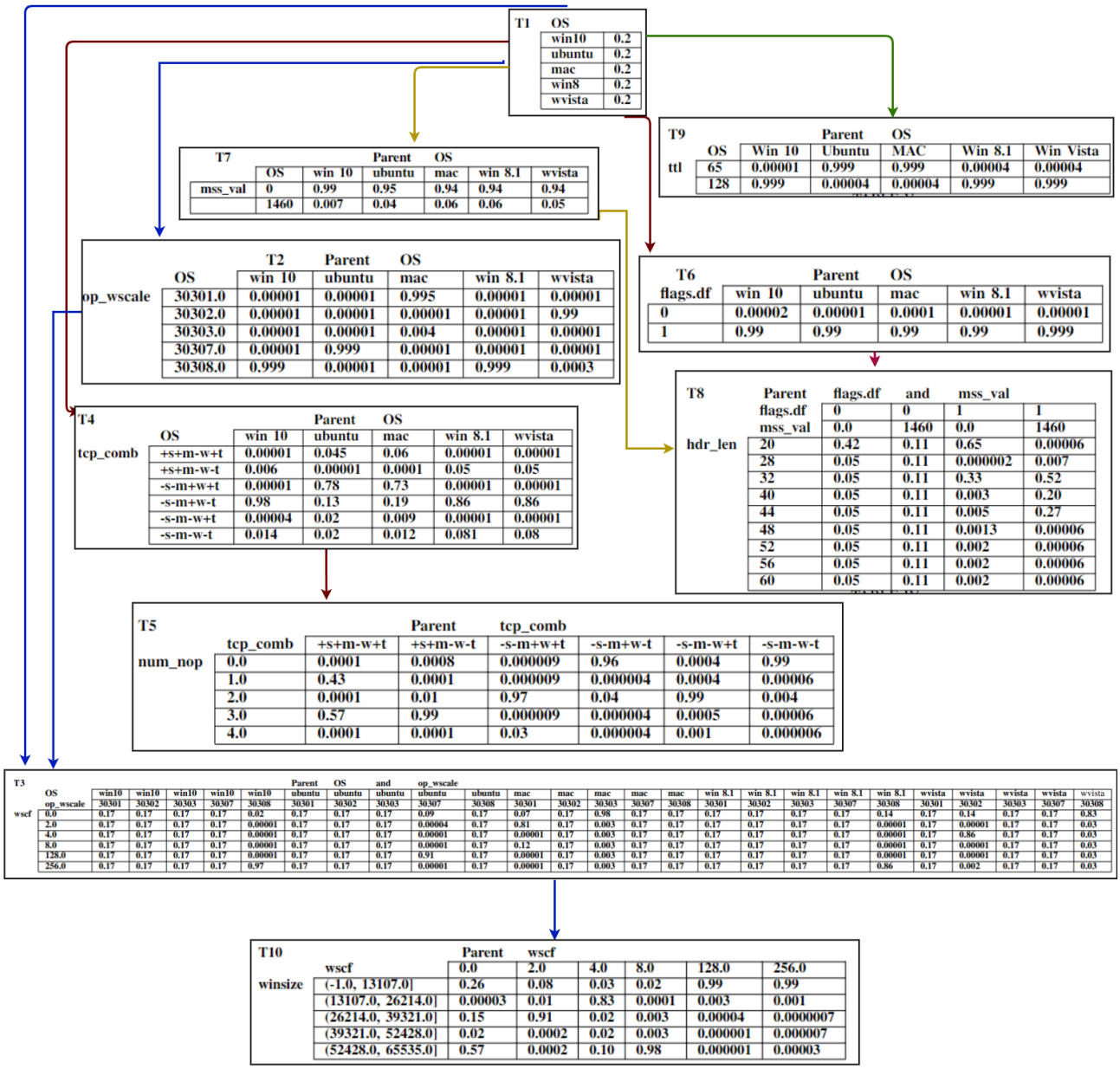
IV. Model Interpretation

We begin by defining some notation and discussing the following for DSS i) BBN model interpretation, ii) formalizing adversaries belief in the following sections. A Bayesian network (Fig. 1) consists of nodes, edges, and CPT table (containing corresponding node states and conditional probabilities represented in Fig. 2, each of which is discussed in brief below.

- **Nodes:** The nodes are classified into two categories:
  - **Root Node**: The root node is the OS, as depicted in (figure 1). The set of OSs which are represented by $\{OS_\phi\}_{\phi=a}^{z}$, where ϕ denotes the type or state of OS such as windows 10, linux etc.
  - **Feature Nodes**: All nodes except the root node in figure 1 are feature nodes, denoted by F, where $F = \{f_1, f_2, \ldots, f_n\}$. Each of these features consists of a number of attributes; $f_i = \{v_j^{p_i}\}_i$ where i represents ith feature; j and p represent the state and parent of the feature, f_i, respectively. As an example, let us assume that f_1 is ttl; the parent, p, of f_1 is the OS; and j has two states in our dataset, 65, 128 Then, $ttl = \{65, 128\}$. We can represent the feature states of f_1 as $f_1 = \{v_1^{os}, v_2^{os}\}$. Usually ttl has three default values 65, 128 and 255. However, the OSes we use here does not contain ttl value 255.
- **CPT table:** The CPT table contains the probability distributions of various OSs and the conditional probabilities corresponding to each feature state of the given parent node. Fig 2 depicts the probability distribution tables. Each table is indexed by an index T_i.

A. Bayesian Network Interpretation

We start by analyzing the Conditional Probability Distribution (CPD) tables Figure 2, which reveal several interesting findings about feature states corresponding to each Operating System (OS). We observe that for Windows 10, Windows 8.1, and Windows Vista, the probability of observing -s-m+w-t (in the feature tcp_comb) is high above 0.85, indicating that for these Windows versions, the chance is significant to observe only window scaling factor and low for mss, wscf, and timestamp. However, the probability is high for observing



T1

OS	
win10	0.2
ubuntu	0.2
mac	0.2
win8	0.2
wvista	0.2

T7

	Parent OS				
	OS	win 10	ubuntu	mac	win 8.1	wvista
mss_val	0	0.99	0.95	0.94	0.94	0.94
	1460	0.007	0.04	0.06	0.06	0.05

T9

	Parent OS					
	OS	Win 10	Ubuntu	MAC	Win 8.1	Win Vista
ttl	65	0.00001	0.999	0.999	0.00004	0.00004
	128	0.999	0.00004	0.00004	0.999	0.999

T2

	Parent OS				
OS	win 10	ubuntu	mac	win 8.1	wvista
op_wscale 30301.0	0.00001	0.00001	0.995	0.00001	0.00001
30302.0	0.00001	0.00001	0.00001	0.00001	0.99
30303.0	0.00001	0.00001	0.004	0.00001	0.00001
30307.0	0.00001	0.999	0.00001	0.00001	0.00001
30308.0	0.999	0.00001	0.00001	0.999	0.0003

T6

	Parent OS				
flags.df	win 10	ubuntu	mac	win 8.1	wvista
0	0.00002	0.00001	0.0001	0.00001	0.00001
1	0.99	0.99	0.99	0.99	0.999

T4

	Parent OS				
OS	win 10	ubuntu	mac	win 8.1	wvista
tcp_comb +s+m-w+t	0.00001	0.045	0.06	0.00001	0.00001
+s+m-w-t	0.006	0.00001	0.0001	0.05	0.05
-s-m+w+t	0.00001	0.78	0.73	0.00001	0.00001
-s-m+w-t	0.98	0.13	0.19	0.86	0.86
-s-m-w+t	0.00004	0.02	0.009	0.00001	0.00001
-s-m-w-t	0.014	0.02	0.012	0.081	0.08

T8

	Parent	flags.df	and	mss_val	
	flags.df	0	0	1	1
	mss_val	0.0	1460	0.0	1460
hdr_len	20	0.42	0.11	0.65	0.00006
	28	0.05	0.11	0.000002	0.007
	32	0.05	0.11	0.33	0.52
	40	0.05	0.11	0.003	0.20
	44	0.05	0.11	0.005	0.27
	48	0.05	0.11	0.0013	0.00006
	52	0.05	0.11	0.002	0.00006
	56	0.05	0.11	0.002	0.00006
	60	0.05	0.11	0.002	0.00006

T5

	Parent tcp_comb					
	tcp_comb	+s+m-w+t	+s+m-w-t	-s-m+w+t	-s-m+w-t	-s-m-w+t	-s-m-w-t
num_nop	0.0	0.0001	0.0008	0.000009	0.96	0.0004	0.99
	1.0	0.43	0.0001	0.000009	0.000004	0.0004	0.00006
	2.0	0.0001	0.01	0.97	0.04	0.99	0.004
	3.0	0.57	0.99	0.000009	0.000004	0.0005	0.00006
	4.0	0.0001	0.0001	0.03	0.000004	0.001	0.000006

T3

	Parent OS and op_wscale																											
	OS	win10	win10	win10	win10	win10	ubuntu	ubuntu	ubuntu	ubuntu	ubuntu	ubuntu	mac	mac	mac	mac	mac	win 8.1	win 8.1	win 8.1	win 8.1	win 8.1	wvista	wvista	wvista	wvista	wvista
	op_wscale	30301	30302	30303	30307	30308	30301	30302	30303	30307	30308	30301	30302	30303	30307	30308	30301	30302	30303	30307	30308	30301	30302	30303	30307	30308
wscf	0.0	0.17	0.17	0.17	0.17	0.02	0.17	0.17	0.17	0.09	0.17	0.07	0.17	0.98	0.17	0.17	0.17	0.17	0.17	0.17	0.14	0.17	0.14	0.17	0.17	0.83
	2.0	0.17	0.17	0.17	0.17	0.00001	0.17	0.17	0.17	0.00004	0.17	0.81	0.17	0.003	0.17	0.17	0.17	0.17	0.17	0.17	0.00001	0.17	0.00001	0.17	0.17	0.03
	4.0	0.17	0.17	0.17	0.17	0.00001	0.17	0.17	0.17	0.00001	0.17	0.00001	0.17	0.003	0.17	0.17	0.17	0.17	0.17	0.17	0.00001	0.17	0.86	0.17	0.17	0.03
	8.0	0.17	0.17	0.17	0.17	0.00001	0.17	0.17	0.17	0.00001	0.17	0.12	0.17	0.003	0.17	0.17	0.17	0.17	0.17	0.17	0.00001	0.17	0.00001	0.17	0.17	0.03
	128.0	0.17	0.17	0.17	0.17	0.00001	0.17	0.17	0.17	0.91	0.17	0.00001	0.17	0.003	0.17	0.17	0.17	0.17	0.17	0.17	0.00001	0.17	0.00001	0.17	0.17	0.03
	256.0	0.17	0.17	0.17	0.17	0.97	0.17	0.17	0.17	0.00001	0.17	0.00001	0.17	0.003	0.17	0.17	0.17	0.17	0.17	0.17	0.86	0.17	0.002	0.17	0.17	0.03

T10

	Parent wscf					
	wscf	0.0	2.0	4.0	8.0	128.0	256.0
winsize	(-1.0, 13107.0]	0.26	0.08	0.03	0.02	0.99	0.99
	(13107.0, 26214.0]	0.00003	0.01	0.83	0.0001	0.003	0.001
	(26214.0, 39321.0]	0.15	0.91	0.02	0.003	0.00004	0.0000007
	(39321.0, 52428.0]	0.02	0.0002	0.02	0.003	0.000001	0.000007
	(52428.0, 65535.0]	0.57	0.0002	0.10	0.98	0.000001	0.00003

Fig. 2. CPT table of Bayesian network.

-s-m+w+t when OS is Ubuntu or Mac with a probability of 0.78 and 0.73, respectively (Fig. 2, table T4).

We examine the relationship between the number of nops and tcp options observed (tcp_comb) and found a proportional relationship. When the minimum number of TCP options such as sack, mss, and timestamp are not observed, the number of NOPs = 0 occurrence is high (figure 2, Table T5). Furthermore, when the OS is Windows, the occurrence of -s-m+w-t is high, we observe NOP = 0 with a probability of 0.96 given -s-m+w-t. In contrast, the likelihood of -s-m+w+t is significant when Ubuntu or Mac is the operating system, and when -s-m+w+t is observed, the likelihood of the number of NOPs = 2 with a probability of 0.97. The active trial of OS $\longrightarrow$ tcp_comb $\longrightarrow$ num_nop in the CPT table demonstrates this distinction between Windows and other operating systems.

Additionally, we notice that the state of op_wscale 30301 is especially prominent when the OS is MAC (Table T2, Figure 2). The likelihood of wsc 30301 is 0.995 when the OS is MAC, but it is much lower when the OS is anything else. By analyzing the data in table T3, we can observe the effect of OS and window scale identifier (op_wscale) on window scaling factor (wscf). For example, when OS is Windows 10 and the op_wscale is 30308, the likelihood of wscf 256 is 0.97; for OS mac and op_wscale 30301, the maximum probability is 0.81 for wscf 2; and for OS ubuntu and op_scale 30307, the maximum probability is 0.91 for wscf = 128 (Fig 2), table T3).

Thus, we can conclude that it is possible to identify different OS using BBN.

Next, we observe the individual effect of mss_val (mss) and data fragment bit (flags.df) given OS. From the CPT table, we observe the occurrence of mss 1460, and the data fragment bit is 0 is low for all of the OS (Fig. 1, table T7 and T6). However, we can observe header length 20 is only observed when mss is 0, and the data fragment bit is either 0 or 1, (fig 1, table T8). When mss is not seen, it is considered mss uses default value 536; we can directly observe the effect of header length on mss value.

Similarly, we identify the feature states that remain identical corresponding to different OSs. New OS, IoT devices, and smart home devices can be included in the model—exploring OS feature states is an efficient method of estimating the ease of prediction of certain OSs. For example, if the attacker already knows the distinctive feature of an OS, they can easily target the specific OS. The defender can also utilize the CPT table to analyze traffic based on the knowledge base.

B. Attacker's Belief

BBN can reveal the perspectives of adversaries by estimating the posterior probability of any set of variables given some evidence. This is useful for both predictive and diagnostic reasoning in network systems that may have various operating systems.

1) Predictive reasoning or causal inference: Causal inference is used for the prediction of effects. For example, estimating the probability of each OS class corresponding to a certain set of feature observations involves a prediction of effects. For example, When an attacker observes the op_wscale state 30302 and wscf 4.0, the BBN computes $\Pr(OS.state \mid wsc.state = 30302, hdr\_len.state = 4.0)$. Propagating this evidence, the network computes the following OS probabilities—Windows 10 with a probability of 0.000001, Mac with a probability of 0.0000004, Windows 8 with a probability of 0.0000001, and Windows Vista with a probability of 0.99. The predictions indicate that the attacker is likely to form the belief that the observation corresponds to Windows Vista. Similarly, if an adversary observes a tcp combination's (tcp_comb) state is -s-m+w+t and num_nop value is 2, the network computes the following OS probabilities: Ubuntu (0.52), Mac (0.48), and any other OS (0) indicating that this observation is not for Windows.

Similarly, any causal inference can be deduced using BBN.

2) Diagnostic reasoning: We can use Bayesian Belief Networks (BBN) for diagnostic reasoning questions to determine the chains of behaviors that have the most significant impact, e.g., the estimation of the probabilities of unobserved variables when the Operating System (OS) is restricted. For example, if the defender or attacker intends to infer whether a given OS is Mac based on observations such as op_scale, window size, and wscf, by computing $\Pr(op\_scale, wscf, winsize \mid OS.state = mac)$ using a BBN inference algorithm, it is possible to identify the configurations of these three variables with the highest probabilities for observing Mac. The BBN's output for observing Mac is with a probability of 0.75 if the window scaling option (op_scale) = 30301.0, window scaling factor (wscf) = 2, and window size (winsize) in the range (26214.0, 39321.0]. Similarly, the network recommends the probability of Windows 10 occurrence is 0.977 with op_scale state = 30308, wscf = 256, and winsize in the range (-1.0, 13107.0]. This illustrates the advantage of utilizing BBN's in order to identify the most likely explanation or inference.

3) Inter-causal reasoning: Inter-causal reasoning in Bayes net is a method of analyzing the relationships between different events and factors. It is possible to establish relationships between different events and factors, allowing for more accurate predictions of future outcomes. For example, the adversary observes ttl = 65, wscf = 128, op_wscale = 30307.0. Our model infers a probability of 0.9728 for observing ubuntu with the above parameter observation.

New OSs are frequently introduced by vendors some of which are very easy to detect, For example, when a Mac exists in a network, the probability of identifying Mac with certain features can be calculated with high probability using the BBN. Similarly, attackers can determine the configurations of all observed feature states that maximize the evidence probability. For example, $\mathrm{argmax}_{f_1,f_2..f_n} \Pr(f_1, f_2, ...f_n \mid OS = mac)$ the most likely configuration in the observed feature set corresponding to mac. Moreover, if a defender intends to implement deception in the system, they can use a belief formation model to implement feature deception.

V. Evaluation

To validate the model, we take the following steps:

1) **f1 score, correlation test, chi-square test:** To assess our BBN structure first we compare our BBN structure against the PC algorithm for structure learning and obtain an F1 score of 0.78, additionally, we perform a correlation test for all pairs of variables in the dataset. If the p-value of the test is less than the significance level (0.05), we assume that there is a correlation between the two variables and examine whether they are d-connected in the network structure. Furthermore, we conduct chi-square conditional independence tests to check if variable X is independent from another variable Y given a set of variables Zs from the data. Where X, Y and Z represent random variable of the data.
2) **Probability matching with local network:** We set up our local network with operating systems Windows 10, Windows 8.1, Mac, Ubuntu and Windows Vista and construct Bayesnet and CPT tables. Our analysis reveals that all probability values from our model match closely with those in the CPT table of the local network's OS.
3) **Predictive accuracy:** Predictive accuracy is a common metric used to evaluate Bayesian networks [25]. We evaluate the classification accuracy of the Bayesnet and compare it with artificial neural network (ann) and random forest.

A. Comparison with Random Forest

We compare the performance of the Bayesian Belief Network (BBN) model with the Random Forest algorithm. We create five disjoint subsamples of the raw data and perform five separate evaluations, each involving one subsample as the testing data and the remaining four samples as the training data. We then average the five results. We modify the value of winsize in the test set and convert it to one of the ranges in the train set according to where it fell. Results from the BBN model show that F1 scores, precision, and recall were 1 for Ubuntu, Mac, and Windows Vista; however, F1 scores were 0.69 and 0.24, respectively, for Windows 10 and 8.1 due to insufficient features related to the Operating System (OS) version rather than OS type. However, adding more features associated with HTTP User-Agent could help resolve this problem. [11]. The prediction results with Random Forest did not improve upon those of the proposed model and generated the same classification report as BBN. The main goal of this decision support system (DSS) is not a prediction but rather to analyze feature states and demonstrate belief updates utilizing the BBN model to identify a target. Figure 3 demonstrates the classification performance of Bayes net and Random forest.

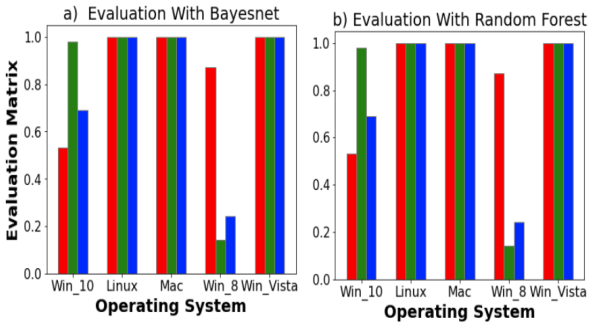


Fig. 3. Evaluation of Bayesian network and random forest classification

B. Comparison with ANN

Artificial neural networks (ANNs) have been widely applied in OS fingerprinting [9] exhibiting good performance in terms of OS prediction. However, using ANNs to interpret or analyze data is difficult.

The lack of symbolic reasoning and semantic representation in ANNs is a drawback. ANNs are generally black box models in the sense that non-linear relationships of cause and effect are not easily interpretable, making it difficult to explain the results [26]. In contrast, we can utilize Bayesnet for both prediction and interpretation. An artificial neural network (ANN) with a 9–12–5 artificial neural network (ANN) was constructed, consisting of nine input nodes, a hidden layer of twelve neurons, and a final output layer with five neurons providing the predicted OS class. Standard Scaler was used for preprocessing, and a sparse cross-entropy loss function and stochastic gradient descent (SGD) optimizer were employed to update the parameters for each training example x_i and label y_i. The model was trained in seven minutes, and the argmax of the predicted class was then taken. The same input variables used in the Bayesian network (BN) were used in the first layer of the ANN.

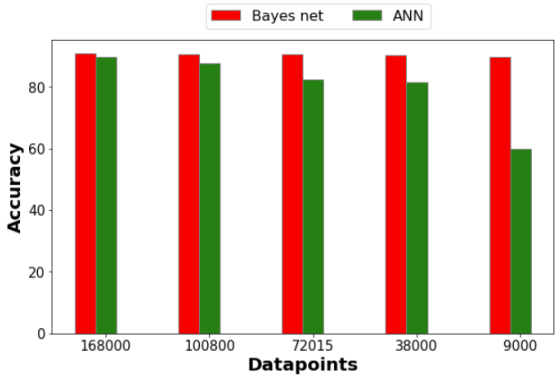


Fig. 4. Accuracy of BBN vs ANN.

We use a different number of data records to compare Bayesnet and the ANN. For the test set, we modify the value of winsize in the test set and converted it to one of the ranges in the train set according to where it fell.The results with different data records for different data points, 168000, 100800, 72015, 38000, and 9000 are presented and we compare the accuracy (Fig. 4). Comparison of the proposed BayesNet to ANN reveals that the former exhibits higher accuracy based on lesser data and is also computationally less expensive (Fig. 4).

VI. Related Work

Yu Liu and Hong Man proposed a Bayesian network model for generating attack graphs to describe exploitation of vulnerabilities; they utilized C++, which automatically generates an attack graph with edges and weights [27]. Bayes net is used for probability calculation from attack paths to the vulnerable node. However, reaching the vulnerable node can not guarantee to find the vulnerability. Our model intends to show how Bayes net can be used for inference once a node is compromised. Our model aims to demonstrate using Bayesian network for inference after a node is compromised. The Bayesian network is generated from real-world data and contains several variables, and is used for parameter learning to depict adversaries' belief formation. The studies [4]–[6]; focused on the attack graph and the path to reach the vulnerable node. In [28], the authors developed a model on how advanced attackers establish and maintain their footprint within a target system with the assumption that the attackers have prior knowledge of the target network before compromising nodes within that network. The study assumes that the adversary has complete knowledge of the network. All the studies above did not focus on the inference technique from the collected data from network nodes. In contrast, the approach proposed in this study first learns from the data and determines the characteristics of the features corresponding to each OS. This node-based enables the defender to adopt different security policies for different nodes based on feature analysis.

VII. Conclusion

We develop a BBN model to infer from the passively collected data and reasoning attackers' belief formation. We evaluate our BBN model with a p-value significance level, matching probability with a network setup, and predictive accuracy. Building BBN models capture what targets are easily identifiable by adversaries. It depicts the influential factor that identifies the target. Additionally, the paper describes how the BBN can be incorporated into a DSS to support causal and diagnostic reasoning analyses of target information. The defender can utilize the BBN for various nodes in the network, where workstations, databases, and servers exist, and determine the relationship of the variables that lead to target identification. In future research, we will add IoT devices to the model and analyze their behavior. In addition, we will investigate the use of Bayesian belief networks (BBN) to identify optimal variables for feature deception and assess the efficiency of such deception by masking those variables in order to identify a target. Furthermore, we plan to utilize BBN models to evaluate the feasibility and technical complexity associated with modifying feature states as part of implementing deceptive measures.

Acknowledgment

This work was supported by the Army Research Office under award W911NF-17-1-0370. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Army Research Office.

References

[1] L. H. Pham, M. Albanese, R. Chadha, C.-Y. J. Chiang, S. Venkatesan, C. Kamhoua, and N. Leslie, "A quantitative framework to model reconnaissance by stealthy attackers and support deception-based defenses," in *2020 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2020, pp. 1–9.

[2] D. H. Hagos, A. Yazidi, Ø. Kure, and P. E. Engelstad, "A machine-learning-based tool for passive OS fingerprinting with TCP variant as a novel feature," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3534–3553, 2020.

[3] L. Ponemon, "Cost of data breach study," *Ponemon Institute*, 2017.

[4] X. Ou, W. F. Boyer, and M. A. McQueen, "A scalable approach to attack graph generation," in *Proceedings of the 13th ACM conference on Computer and communications security*, 2006, pp. 336–345.

[5] M. S. Barik, A. Sengupta, and C. Mazumdar, "Attack graph generation and analysis techniques," *Defence science journal*, vol. 66, no. 6, p. 559, 2016.

[6] M. Albanese, E. Battista, and S. Jajodia, "A deception based approach for defeating OS and service fingerprinting," in *2015 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2015, pp. 317–325.

[7] J. Wei-hua, L. Wei-hua, and D. Jun, "The application of ICMP protocol in network scanning," in *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies*. IEEE, 2003, pp. 904–906.

[8] R. Beverly, "A robust classifier for passive TCP/IP fingerprinting," in *International Workshop on Passive and Active Network Measurement*. Springer, 2004, pp. 158–167.

[9] J. Song, C. Cho, and Y. Won, "Analysis of operating system identification via fingerprinting and machine learning," *Computers & Electrical Engineering*, vol. 78, pp. 1–10, 2019.

[10] A. S. Khatouni, L. Zhang, K. Aziz, I. Zincir, and N. Zincir-Heywood, "Exploring NAT detection and host identification using machine learning," in *2019 15th International Conference on Network and Service Management (CNSM)*. IEEE, 2019, pp. 1–8.

[11] M. Lastovicka, T. Jirsik, P. Celeda, S. Spacek, and D. Filakovsky, "Passive OS fingerprinting methods in the jungle of wireless networks," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–9.

[12] H. J. Suermondt, "Explanation in bayesian belief networks," Ph.D. dissertation, Stanford University, 1992.

[13] X. Zhang, J. Kang, and T. Jin, "Degradation modeling and maintenance decisions based on bayesian belief networks," *IEEE Transactions on Reliability*, vol. 63, no. 2, pp. 620–633, 2014.

[14] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[15] C. Lee and P. v. Beek, "Metaheuristics for score-and-search bayesian network structure learning," in *Canadian Conference on Artificial Intelligence*. Springer, 2017, pp. 129–141.

[16] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." in *ICISSp*, 2018, pp. 108–116.

[17] B. Anderson and D. McGrew, "OS fingerprinting: New techniques and a study of information gain and obfuscation," in *2017 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2017, pp. 1–9.

[18] A. Orebaugh, G. Ramirez, and J. Beale, *Wireshark & Ethereal network protocol analyzer toolkit*. Elsevier, 2006.

[19] C. Yuan and B. Malone, "Learning optimal bayesian networks: A shortest path perspective," *Journal of Artificial Intelligence Research*, vol. 48, pp. 23–65, 2013.

[20] M. Fisk and W.-c. Feng, "Dynamic adjustment of TCP window sizes," *LANL Report: LA-UR 00-3221*, 2000.

[21] geeksforgeeks, *Overview of Maximum Segment Size*, 2022 (Online; accessed September 20, 2022), https://www.geeksforgeeks.org/overview-of-maximum-segment-size/.

[22] A. Ankan and A. Panda, "pgmpy: Probabilistic graphical models using python," in *Proceedings of the 14th python in science conference (scipy 2015)*, vol. 10. Citeseer, 2015.

[23] M. Scutari, "Dirichlet bayesian network scores and the maximum relative entropy principle," *Behaviormetrika*, vol. 45, no. 2, pp. 337–362, 2018.

[24] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Pearson, 2002.

[25] S. Schietekat, A. De Waal, and K. G. Gopaul, "Validation & verification of a bayesian network model for aircraft vulnerability," in *12th INCOSE SA Systems Engineering Conference*. INCOSE, 2016.

[26] A. Ghorbani, A. Abid, and J. Zou, "Interpretation of neural networks is fragile," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 3681–3688.

[27] Y. Liu and H. Man, "Network vulnerability assessment using bayesian networks," in *Data mining, intrusion detection, information assurance, and data networks security 2005*, vol. 5812. SPIE, 2005, pp. 61–71.

[28] L. H. Pham, M. Albanese, and B. W. Priest, "A quantitative framework to model advanced persistent threats." in *ICETE (2)*, 2018, pp. 448–459.