Probabilistic Modeling and Inference for Obfuscated Network Attack Sequences

Haitao Du
Probabilistic Modeling and Inference for Obfuscated Network Attack Sequences

by

Haitao Du

A Dissertation Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy in Computing and Information Sciences
in
GCCIS PhD Program in Computing and Information Sciences

Supervised by

Dr. Shanchieh Jay Yang

Department of GCCIS PhD Program in Computing and Information Sciences

B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York

Aug. 2014
The dissertation “Probabilistic Modeling and Inference for Obfuscated Network Attack Sequences” by Haitao Du has been examined and approved by the following Examination Committee:

Dr. Shanchieh Jay Yang
Associate Professor
Dissertation Advisor

Dr. Pengcheng Shi
Professor
Dissertation Committee Member

Dr. Bo Yuan
Associate Professor
Dissertation Committee Member

Dr. Justin Domke
Senior Researcher
Dissertation Committee Member

Dr. Michael Kuhl
Professor
Dissertation Defense Chair
Dedication

I dedicate my dissertation work to my family, especially my beloved wife Tingting.
Acknowledgments

I would like to thank my advisor Dr. Yang and my committee members for all the guidance, support and help through these years.
Abstract

Probabilistic Modeling and Inference for Obfuscated Network Attack Sequences

Haitao Du

Supervising Professor: Dr. Shanchieh Jay Yang

Prevalent computing devices with networking capabilities have become critical network infrastructure for government, industry, academia and every-day life. As their value rises, the motivation driving network attacks on this infrastructure has shifted from the pursuit of notoriety to the pursuit of profit or political gains, leading to network attack on various scales. Facing diverse network attack strategies and overwhelming alters, much work has been devoted to correlate observed malicious events to pre-defined scenarios, attempting to deduce the attack plans based on expert models of how network attacks may transpire.

We started the exploration of characterizing network attacks by investigating how temporal and spatial features of attack sequence can be used to describe different types of attack sources in real data set. Attack sequence models were built from real data set to describe different attack strategies. Based on the probabilistic attack sequence model, attack predictions were made to actively predict next possible actions. Experiments through attack predictions have revealed that sophisticated attackers can employ a number of obfuscation techniques to confuse the alert correlation engine or classifier. Unfortunately, most exiting work treats attack obfuscations by developing ad-hoc fixes to specific obfuscation technique. To this end, we developed an attack modeling framework that enables a
systematical analysis of obfuscations.

The proposed framework represents network attack strategies as general finite order Markov models and integrates it with different attack obfuscation models to form probabilistic graphical model models. A set of algorithms is developed to inference the network attack strategies given the models and the observed sequences, which are likely to be obfuscated. The algorithms enable an efficient analysis of the impact of different obfuscation techniques and attack strategies, by determining the expected classification accuracy of the obfuscated sequences. The algorithms are developed by integrating the recursion concept in dynamic programming and the Monte-Carlo method.

The primary contributions of this work include the development of the formal framework and the algorithms to evaluate the impact of attack obfuscations. Several knowledge-driven attack obfuscation models are developed and analyzed to demonstrate the impact of different types of commonly used obfuscation techniques. The framework and algorithms developed in this work can also be applied to other contexts beyond network security. Any behavior sequences that might suffer from noise and require matching to pre-defined models can use this work to recover the most likely original sequence or evaluate quantitatively the expected classification accuracy one can achieve to separate the sequences.
Contents

Dedication .................................................. iii

Acknowledgments ........................................ iv

Abstract ................................................... v

1 Introduction .............................................. 1
   1.1 Overwhelming, diverse and evolving network attacks .......... 1
   1.2 Intrusion detection and alert correlation .................... 2
   1.3 Host clustering and botnet detection ....................... 5
   1.4 Network attack modeling and prediction .................. 7
   1.5 Network attack data set .................................. 11
   1.6 Overview the rest of the dissertation .................... 16

2 Network attack characterization, modeling and prediction ........ 18
   2.1 Attack action sequence features .......................... 18
   2.2 Predict possible attack actions ........................... 30
   2.3 Technical gap for dealing noisy attack sequences ........ 40

3 Modeling obfuscated attack sequences ........................ 42
   3.1 Attack obfuscation and countermeasures .................. 42
   3.1.1 Review of network attack obfuscation .................. 42
   3.1.2 Current attack obfuscation countermeasures ............ 51
   3.2 Probabilistic modeling on attack obfuscations .......... 51
   3.2.1 Type-I model for action alteration .................... 53
   3.2.2 Type-II model for action insertion and action removal .. 57
   3.3 Probabilistic inference and impact assessment for obfuscated sequences .... 61

4 Inference algorithm design ................................ 66
   4.1 Probabilistic inference algorithm design .................. 67
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.1 Basic probabilistic inference problem</td>
<td>67</td>
</tr>
<tr>
<td>4.1.2 Inference for different obfuscation models</td>
<td>74</td>
</tr>
<tr>
<td>4.2 Performance evaluation for sequence classification</td>
<td>80</td>
</tr>
<tr>
<td>4.3 Algorithm efficiency verification</td>
<td>82</td>
</tr>
<tr>
<td>5 Network attack obfuscation simulation and results</td>
<td>86</td>
</tr>
<tr>
<td>5.1 Simulation framework and set up</td>
<td>86</td>
</tr>
<tr>
<td>5.1.1 Attack action space</td>
<td>86</td>
</tr>
<tr>
<td>5.1.2 Attack models</td>
<td>88</td>
</tr>
<tr>
<td>5.1.3 Obfuscation models</td>
<td>89</td>
</tr>
<tr>
<td>5.1.4 Simulation overview</td>
<td>90</td>
</tr>
<tr>
<td>5.2 Type-I model simulation and results</td>
<td>91</td>
</tr>
<tr>
<td>5.2.1 A case study of action alteration obfuscation</td>
<td>91</td>
</tr>
<tr>
<td>5.2.2 Action alteration impact evaluation</td>
<td>94</td>
</tr>
<tr>
<td>5.2.3 Evaluation the impact inaccurate obfuscation model</td>
<td>98</td>
</tr>
<tr>
<td>5.3 Type-II model simulation and results</td>
<td>102</td>
</tr>
<tr>
<td>5.3.1 Action insertion simulation</td>
<td>102</td>
</tr>
<tr>
<td>5.3.2 Action removal simulation</td>
<td>105</td>
</tr>
<tr>
<td>6 Conclusion</td>
<td>109</td>
</tr>
<tr>
<td>Bibliography</td>
<td>111</td>
</tr>
<tr>
<td>A Snort alerts explanation</td>
<td>119</td>
</tr>
<tr>
<td>B Recursion rule theorem proof</td>
<td>125</td>
</tr>
</tbody>
</table>
# List of Tables

1.1 An example of alert correlation scenario .............................................. 4
1.2 Review of alert correlation work ......................................................... 5
1.3 Review of host clustering work ............................................................ 7
1.4 Review of alert correlation and prediction work with data set .................. 15
1.5 Review of host clustering work with data set ....................................... 15

2.1 An example of attack from real-world data ......................................... 19
2.2 Examples of coordinated attacks with different strategies ....................... 21
2.3 Packet level details for attack source 0.86.249.218 ................................ 29
2.4 Per attribute prediction example by VLMM ......................................... 31
2.5 Fuzzy inference rules used to combine VLMM outputs ............................ 33
2.6 Projection performance achieved by F-VLMM for the various datasets ..... 36
2.7 A high efficiency attack with percentile rank interval achieved by F-VLMM 38
2.8 A low efficiency attack with percentile rank interval achieved by F-VLMM 39
2.9 A stealthy attack with percentile rank interval by F-VLMM .................... 39

3.1 Hex payload and explanation for RPC Sadmind Snort rule ..................... 43
3.2 An example scenario of action alteration obfuscation ............................ 45
3.3 An example scenario of noise injection obfuscation ............................... 46
3.4 An example scenario of self-throttling obfuscation ............................... 47
3.5 An example scenario of activity splitting obfuscation ............................ 48
3.6 An example scenario of action removal obfuscation ............................... 49
3.7 An example of attack sequence obfuscation ........................................ 50
3.8 An example of $P(Y_k|X_k)$ for HMM ................................................. 56
3.9 An example of $P(Y_k|X_k)$ for proposed action alteration model ........... 56
3.10 An example of parameter set up for link removal toy example ............... 61
3.11 Notations used in algorithm design .................................................... 65

5.1 Attack model 1 (partial) used in simulation ......................................... 89
5.2 Attack model 2 (partial) used in simulation ......................................... 89
# List of Figures

1.1 Attack sophistication vs. Intruder technical knowledge .......................... 2  
1.2 Examples of data points for host clustering ......................................... 6  
1.3 Example of Bayesian networks for attack prediction .............................. 8  
1.4 An example of attack prediction using recommendation system [27] .......... 9  
1.5 Examples of suffix tree for VLMM attack prediction .............................. 10  
2.1 Example of spatial feature for coordinated attacks ................................. 21  
2.2 Example of ASG analysis ................................................................. 23  
2.3 Attack source labeling ................................................................. 25  
2.4 Degree centrality based attack sources clustering ................................. 25  
2.5 Spatial pattern sets probabilities .................................................... 27  
2.6 Label sequence comparison for different attacking strategies .................... 27  
2.7 Label Sequence and ASG Subgraphs for Attack Source 0.86.249.218 ........ 30  
2.8 A scatter plot of $Proj_i$ versus $Proj_d$ of all hosts .............................. 32  
2.9 The fuzzy membership functions ................................................... 33  
2.10 The I/O surface plot of the fuzzy combination system .......................... 34  
2.11 Network A used for attack prediction simulation ................................. 35  
2.12 Network B used for attack prediction simulation ................................. 36  
2.13 The number of targets receiving different threat scores: all targets (top) vs.  
only attacked targets (bottom) .......................................................... 37  
3.1 An example of RPC Sadmind Snort rule ........................................... 43  
3.2 Network used for illustrating obfuscation techniques ............................. 44  
3.3 Graphical model notation for 1st order model .................................... 53  
3.4 Graphical model notation for HMM ................................................... 54  
3.5 An example of proposed Type-I model (second order) .......................... 55  
3.6 Graphical representation for action insertion .................................... 58  
3.7 Graphical representation for action removal .................................... 59  
3.8 Graphical representation of action insertion model 1 .......................... 59  
3.9 An example of adding dependencies on Type-I model .......................... 60
Chapter 1

Introduction

1.1 Overwhelming, diverse and evolving network attacks

Prevalent computing devices with networking capabilities have become critical network infrastructure for government, industry, academia and every-day life. As their value rises, the motivation driving network attacks on this infrastructure has shifted from the pursuit of notoriety to the pursuit of profit [1, 2] or political gains, leading to network attack on various scales.

As new software vulnerabilities are discovered by few elite attackers, they are routinely bought and sold by underground organizations. Corresponding attack tools take advantage of the novel vulnerabilities and recruit new zombie (compromised) hosts for the attacker. As shown in Fig. 1.1, Fuchsberger [3] published ten years ago, gives the trends of network attack, i.e., attack sophistication will increasing over time, at the same time only limited intruder’s knowledge are needed. All of these trends are coming from the automated attack tools and compromised zombie hosts.

According to the Symantec Internet security threat report [1] [4], the largest botnet observed in 2010 had over 1 million bots under control, and underground economy advertisements promote 10,000 bots for $15. There are 552 million identities exposed during year 2013, which is almost five times than identities exposed during 2012. Moreover, there are more targeted attacks [4] with reasonable level of sophistication for specific sensitive data, in addition of random attacks created by virus or worms for propagation. Large-scale cyber attacks can take the traditional form of a botnet, from which a large number of hosts
2

Figure 1.1: Attack sophistication vs. Intruder technical knowledge

perform similar actions, e.g., Distributed Denial-of-Service (DDoS) or distributed stealthy scans [2]; they can also consist of a set of colluding sources dividing up tasks, interleaving actions over time and dispersing over the IP and port spaces to conceal their overall strategy.

1.2 Intrusion detection and alert correlation

Network attacks can be captured and observed by Intrusion Detection System (IDS) alerts. The essential goal of a IDS is to differentiate malicious activities from the normal ones and report suspicious actions to network analyst. For more than two decades, significant effort has been put into advancing intrusion detection via anomaly-based and signature-based systems [3]. Anomaly-based detection techniques model the users or systems normal behavior, and report outliers as potential malicious activities. On the other hand, signature-based detection systems usually maintain a database of malicious behavior signatures and use pattern matching techniques to detect malicious actions. Anomaly-based detection systems can be very effective to deal with novel attacks, but suffer from overwhelming false
positives. Signatures-based system can have more accurate detection but can be ineffective on novel attacks. Details of intrusion detection techniques can be found in several survey papers, e.g., [5, 3, 6, 7].

Intrusion detection is challenging because of the variety of normal behaviors, fast changing cyber environment (network services and configuration) as well as new vulnerabilities and attacks. While intrusion detection techniques continues to evolve and improve, the overwhelming and heterogeneous IDS alerts has made the analysis difficult and unable to provide an effective situation assessment. As a result, alert correlation has become a popular topic in the past decade [8, 9]. Ideally, the goal of alert correlation is to determine collections of IDS alerts, where each collection corresponds to a high-level description of the attack.

The tasks of alert correlation can be categorized into Normalization (Norm.), Aggregation (Agg.), Correlation (Corr.) and Strategy Analysis (SA) [9]. The normalization organizes the format of alerts from heterogeneous IDS sensors. Aggregation combines alerts that share the same root causes, e.g., originated from same source IP or attack the same target. In the correlation step, aggregated alerts are mapped into an attack scenario template. Further, the causal relationship, i.e., pre-condition and post-condition, can be used in strategy analysis to infer attack intention and strategies.

The details of alert correlation and a performance evaluation can be found in [9, 10]. Table 1.1 uses an example to illustrate an ordered sequence of seven alerts for alert correlation. The example is extracted from [11] which provides a comprehensive framework for alert correlation.

The monitored network has four heterogeneous IDS sensors, i.e., network based IDS 1 and 2, (N1,N2), host based and application based IDS (H, A). The attacker (31.3.3.7) first launches a port scan against 10.0.0.1 and discovers the vulnerability of Apache server. After

---

1Performance evaluation is very challenging for alert correlation. To best of our knowledge, Haines’s work [10] is the only one systematically evaluates and compares the performance for different alert correlation system. In the end of Chapter 2, we will discuss the technical gap of performance evaluation for existing attack analysis systems.
Table 1.1: An example of alert correlation scenario

<table>
<thead>
<tr>
<th>Alert ID</th>
<th>Description</th>
<th>Sensor</th>
<th>Start/End Time</th>
<th>Source</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IIS Exploit</td>
<td>N1</td>
<td>12.0/12.0</td>
<td>80.0.0.1</td>
<td>10.0.0.1, port: 80</td>
</tr>
<tr>
<td>2</td>
<td>Scanning</td>
<td>N2</td>
<td>10.1/14.8</td>
<td>31.3.3.7</td>
<td>10.0.0.1</td>
</tr>
<tr>
<td>3</td>
<td>Port Scan</td>
<td>N1</td>
<td>10.0/15.0</td>
<td>31.3.3.7</td>
<td>10.0.0.1</td>
</tr>
<tr>
<td>4</td>
<td>Apache Exploit</td>
<td>N1</td>
<td>22.0/22.0</td>
<td>31.3.3.7</td>
<td>10.0.0.1, port: 80</td>
</tr>
<tr>
<td>5</td>
<td>Bad Request</td>
<td>A</td>
<td>22.0/22.1</td>
<td>localhost,</td>
<td>localhost, Apache</td>
</tr>
<tr>
<td>6</td>
<td>Local Exploit</td>
<td>H</td>
<td>24.6/24.6</td>
<td>linuxconf</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Local Exploit</td>
<td>H</td>
<td>24.7/24.7</td>
<td>linuxconf</td>
<td></td>
</tr>
</tbody>
</table>

scanning, the attacker performs a successful Apache buffer overflow exploit on the target and obtains user privilege on the server. Finally, the attacker launches privilege escalation by using a local exploit `linuxconf`. In addition, there is one noisy alert triggered by a worm that probes the same target while the attack is in progress. The ideal output of alert correlation would successfully group Alerts #2 and #3 as malicious scanning, Alerts #4 and #5 as vulnerability attempts, Alerts #6 and #7 as privilege escalation, and the noisy Alert #1 will be marked as irrelevant. After the aggregation of alerts, the scanning, vulnerability attempts and privilege escalation should be correlated and reported to security analysts. This example shows that once correlated, an analyst can be more effective to process the high-level attack descriptions instead of individual alerts.

The methodologies widely used in correlation engines include similarity-based clustering and causal relationship based (pre/post condition) reasoning [9]. The attack scenario templates can be pre-defined or automatically learned from data. Uncertainties usually are captured with Bayesian networks, which will be reviewed in Section 1.4. Table 1.2 is a summary of representative alert correlation approaches.

Alert correlation synthesizes the raw IDS alerts into attack scenarios and provides better situation awareness to security analysts. However, as discussed earlier, the ever-changing environment, e.g., software patches, new IDSs, customized alerts, and new exploits, make alert correlation challenging. In addition, the attack-scenario (pre/post-condition) based
Table 1.2: Review of alert correlation work

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Framework</th>
<th>Methodology</th>
</tr>
</thead>
</table>

approaches are accurate but will not scale for a large number of diverse and unknown attacks, similar to the limitation of signature-based intrusion detection.

### 1.3 Host clustering and botnet detection

Going beyond focusing on examining the detailed attack action level and alert correlation, in a large scale, host clustering work allows us to group a large number of (normal or malicious) hosts that share similar behavior into clusters. Host clustering and botnet detection are other useful tools for network analyst to understand large scale attacks.

The input for host clustering usually is passive Internet backbone traffic [22, 23, 24, 25] or malicious traffic [26, 27] and the outputs are host clusters with behavior pattern descriptions, which could be useful for anomaly detection and attack profiling. Some malicious activities, such as worms, scanning and DDoS attacks, can be detected behavior by host clustering work [24, 25]. An example of host clustering for botnet detection can be found in [25]. Scan activity, spam activity, binary downloading and exploit activity are classical activities that botnet hosts would behave. By looking at the host clusters that exhibit these different activities, one can infer the structure of botnet zombie hosts.
The key differences between various types of host clustering work can be understood as selecting different features to form a multi-dimensional data point and objective function for clustering them. The data point \( x \) represents an attacking host that contains several features on security context. Among all the basic attributes in the communications, the widely used features are statistics from flow information (source IP, destination IP) and protocol information (TCP/IP protocol, source port, destination port). Figure 1.2 is an example of data points from host clustering work [23] and [28]. For the left subfigure, every point in the figure represents one attacking host. Three features, Relative Uncertainty (RU) on source port, destination port, and destination IP are taken into account to characterize an attacker’s behavior. For the right subfigure, every data points represents a set of attacking sources sharing the same spatial attributes, e.g., number of targets, etc... The data points in Fig. 1.2 exhibit cluster patterns. After selecting the features, the next step is clustering the data points, which optimizes on specific objective function. Different objective functions group data points based on different principles. Some of them minimize the distance between data points and cluster center (e.g., K-means clustering) and some utilizing the graph cut notion (e.g., spectral clustering) to separate points are distinct from each other. Table 1.3 is a summary review of host clustering work.

Figure 1.2: Examples of data points for host clustering

Host clustering analyzes a large number of attacking sources by grouping similar attack sources and extracting patterns of clusters. It may be ineffective to discover sophisticated
attacks, which take slow actions and utilize multiple attack sources. Nevertheless, the features and the clustering analysis methodology can be very helpful for treating overwhelming numbers of alerts. Further, the overwhelming attacking data with various types of features has led to challenges to extract, comprehend and predict the diverse attack strategies and goals within the mixture of various attack strategies.

### 1.4 Network attack modeling and prediction

Network attack modeling is a widely used term, in early stage, most models are descriptive and deterministic model. In late 1990’s, Cohen et al. [29] provided one of the pioneering network attack modeling frameworks. They used cause-and-effect models to deduce 37 threat profiles (behaviors), 94 attacks (physical and cyber), and 140 defense mechanisms, and reported a set of simulation results [30]. Their work, along with several others [31, 5, 32] in late 1990’s and early 2000’s, have provided a comprehensive understanding of the different cyber attack types and their effect to networked systems.

Computational and probabilistic modeling is another important branch of attack modeling. Bayesian networks are widely used to model uncertainty in the security context because the conditional dependency fits perfectly with pre/post-condition and attack scenarios. Several works [19, 33, 34] utilize Bayesian networks for predicting high-level goal of an attack. The use of Bayesian networks to model high level attack plans requires a
mapping between specific alerts to attack categories, but it allows probability inference and helps reduce from all possible future attack actions to a differentiable list of likely future attacks. Figure 1.3 gives an example of Bayesian networks modeling applied to the security context.

Consider a simplified attack scenario with four random variables representing the stages of an attack. Let $B$ denote “install Backdoor on the system”, $C$ as “Compromise application account and password”, $M$ as “Monitor confidential transactions”, and $S$ denote “Successfully obtain the confidential data”. The conditional dependencies of the random variables can be described in Fig. 1.3. It indicates that the success of $C$ (compromise application account and password) and $M$ (monitor confidential transactions) are independent given their parent $B$ (install back door on system). Meanwhile, $S$ (Successfully obtain the confidential data) depends on $C$ and $M$.

![Figure 1.3: Example of Bayesian networks for attack prediction](image)

In addition to the structure of the model, the parameters, i.e., the conditional probabilities, of the model could potentially be derived from security knowledge. In particular, a complete model will specify $P(B)$, $P(C|B)$, $P(M|B)$ and $P(S|C,M)$. Therefore, according to the conditional dependencies, the joint distribution $P(B,C,M,S)$ can be decomposed into $P(B)P(C|B)P(M|B)P(S|C,M)$. With simplified joint probability, one can perform any inference on interested events.

There are many challenges when using Bayesian networks for cyber attack modeling and prediction. The key problem for Bayesian networks is the assumption of the model. Unlike other fields, the model structure and parameters have very high uncertainty for multistage cyber attacks. Furthermore, training using up-to-date multistage data is almost
impossible, as little ground truth exists for the stages a cyber attack goes through. Manually specifying both the structure and the parameters may be error-prone for a large network.

Because of its limitations on scalability and the requirement of domain knowledge, Bayesian networks are typically used to predict the high-level goal of an attack, for example, whether the attacker will compromise an application account and password. On the other hand, sequence modeling, e.g., [27, 35], has been utilized to predict more detailed attack actions, e.g., the next attack target or service.

Generally speaking, sequence modeling techniques learn attack patterns from observed attack sequences and predict the future actions of a given sequence based on the aggregate likelihood of similar attack patterns. Soldo et al. [27] developed a cyber attack prediction system by drawing an analogy from the context of recommendation systems [36], which has been used to recommend movies based on similar users’ preferences. Figure 1.4 shows an example of this analogy between recommendation systems and cyber attack prediction, reproduced from Soldo et al. [27]. Figure 1.4(a) is the matrix denoting a user’s preference in a recommendation system. Element $a_{i,j}$ represents whether the item $i$ is borrowed by user $j$. Figure 1.4(b) presents a similar idea in the context of cyber attacks. Similar preference on the choice of targets from similarly behaving attacks are used to ‘recommend/predict’ the targets of a given attack.

![Figure 1.4: An example of attack prediction using recommendation system [27]](image)

A more explicit approach to extract the sequential dependencies between attack action attributes within each attack sequence is to use a Variable Length Markov Model (VLMM
Consider an ongoing attack with $N$ observed actions $\{X_1, X_2, \cdots, X_N\}$. A model of order $L$ assumes the current observed event is conditional depend on previous $L$ events. The probabilities are obtained from sample counts in historical and ongoing attack sequences. A sequence of length $N$ will contribute to the building of $L^{th}$ order models for $1 \leq L \leq N$. More specifically, a sequence of length $N$ will provide one sample to the $n^{th}$ order model, two samples to the $(N - 1)^{th}$ order model, ... , and $N$ samples to the 1st order model. For implementation, a suffix tree can be used to record the samples and to store models of different orders, which allows making predictions based on observed context in $O(N)$ time given a sequence of length $N$. Figure 1.5 shows the suffix tree corresponding to a single sequence of ‘$A, B, A, B, B$’, where the edge weights indicate the number of times the corresponding transition has occurred.

In reality, the suffix tree will be built with many attack sequences and continuously updated with incoming alerts. The overall suffix tree represents the various possible Markov relationships of different orders. For a given ongoing sequence with length $N$, the for $o^{th}$ order model $P_o(X_K), \forall 1 \leq o \leq N$ can be found from the suffix tree in $O(N^2)$ time, which may be further simplified. The probability of next event in the sequence $P_o(X_{N+1})$ can then be blended to make the prediction for next event as

\[ P_o(X_{N+1}) \]

\[ \text{From the perspective of Bias-Variance trade-off [37], we may not need to learn a very higher order model, because there is no sufficient data to train such order, and will lead to a very high variance. The weight assignment for different order’s model takes this factor into account, and generally the higher order model will have a very small weights.} \]
\[
P(X_{K+1}) = \sum_{o=-1}^{K} w_o \cdot P(X_{K+1}|X_{K-o+1}, \ldots, X_K)
\]
where \( w_o \) is the weight associated with the \( o \)th order model, and \( \sum_{o=-1}^{K} w_o = 1 \). Note that finite sequences should be penalized by their rarity and rewarded by their specificity. The weights are designed to be adaptive to take into account the Bias-Variance trade-off [37]. Examples of the weight functions can be found in [38]. Fava [35] shows that there is no significant performance difference when using different weight functions, as long as the the weights satisfy the properties described above. Notice that the summation starts at \(-1\).

The minus-one order model assigns all characters a probability of \(1/|\Omega|\) to prevent the zero frequency problem [38]. The zero order model assumes all observations are independent and holds the frequency count of all \(X \in \Omega\). For the given example in Fig. 1.5, after observing ‘A, B, A, B, B’, we predict next event by considering the minus-one order model \((P_0(A) = P_0(B) = 0.5)\), zero order model \((P_1(A) = 0.4, P_1(B) = 0.6)\), first order model \((P_2(A|A) = 0, P_2(B|A) = 0.5, P_2(B|B) = 0.25, P_2(B|B) = 0.25)\), all the way to the fifth order model. The predictions in different models will be blended using different weights.

The VLMM model allows us to discover patterns within attack sequences without explicitly defining attack plans [35]. In fact, an ongoing attack sequence can match patterns from numerous different types of preceding attack sequences. A VLMM combines the probabilities associated with all matched patterns and produce a better guess.

### 1.5 Network attack data set

The lack of representative data set with comprehensive ground truth label always be the fundamental challenge for network security research, from intrusion detection, alert correlation to attack prediction. In this section, we will review widely used data set in aforementioned related work. Such review allows us to have better understanding of related work’s objective and limitations.

In general, it is very difficult to acquire network data sets with ground truth label. For
typical enterprise networks, the collection of IDS alerts and host logs would not be useful for security research, because the data contains no label to the intrusion stage or specific attacking behavior. There are also many privacy concerns to release the such data for research purposes. Therefore, some milestone data set generated by government or research institute, such as MIT Lincoln lab (DARPA data set [39] [40] [10]) and DEFCON data set [41], are widely used across most security research.

In 1998 and 1999, DARPA intrusion detection evaluation group of MIT Lincoln Laboratory collected and distributed two data set, DARPA 1998 and DARPA 1999, for IDS evaluation purpose, i.e., the goal is investigating the performance of IDS detection engine on reporting malicious activities. The data set was collected over several weeks in a controlled environment to simulate an enterprise network. Various types of attack tools, including probes, remote to local attacks, user to root attacks and DoS attacks are utilized to conduct attacks (some of the attacks would not be valid today because the vulnerability does not exist anymore, such as Ping of Death vulnerability). Finally, the data sets are divided into training data set and testing data set and labeled with ground truth intrusion activities.

In 2000, DARPA released two data sets with more sophisticated, multi-stage attack scenarios. The attack goal is to install components and conduct DDoS attack in the target network. The whole process has five attack phases, over the course of which the attacker probes the network, breaks in to a host by exploiting the vulnerabilities, installs Trojan DDoS software and eventually launches a DDoS attack from compromised hosts [42]. The data sets also contains detailed labels for intrusion activities over the phases. Unlike DARPA1998 and DARPA1999 data set, whose purpose is validating the IDS detection engine, DARPA 2000 data set contains multistage attack and is widely used to validate the alert correlation research. In 2003, DARPA Cyber Panel Program released Grand Challenge Problem (GCP) for alert correlation evaluation[10]. Dataset is generated by an attack simulator, which simulates two innovative worm attack scenarios in an enterprise network. Data set includes multiple heterogeneous IDS alerts, and firewall logs generated by attacks
as well as many background alerts that make alert correlation and attack strategy detection more challenging. GCP data set is one important benchmark data to evaluate alert correlation system performance and the results are published in [10].

In addition to the DARPA data set, hacking competitions provide another useful resources of understand network attacks. DEFCON is the world’s largest annual computer hacker conference, and it organizes Capture the Flag (CTF) hacking competition every year. During the competition, hackers are organized as teams, and vulnerable virtual machine images are provided to each team. The goals are protecting its own system and attack other teams. DEFCON CTF 8 and CTF 9 data set (collected in 2000 and 2001) are also used for intrusion detection and alert correlation research. University of California, Santa Barbara and United States Military Academy West Point also organize International catch the flag (iCTF) [43] and cyber defense exercise [44] hacking competitions and release the traffic capture, IDS alerts and host logs to public.

Comparing to the benchmark data set created by DARPA, there are limited work using the hacking competition data. Cipriano et al. [45] developed a production system called Nexat. The approach groups intrusion alerts into attack sessions based purely on the source and destination IP addresses recorded in the alerts. Then, statistics are recorded to determine which types of attack actions are more likely to be in the same session. Prediction of future attack actions are, thus, chosen based on the overall statistics given recently captured alerts. Nexat was evaluated using the iCTF competition dataset and has show reasonably good prediction performance. However, it is unclear whether the simplistic definition of the attack session and the statistics can be generalized and applicable for large-scale networks with diverse attack goals.

On the other hand, in a larger scale it is possible to capture Internet attacks by passive data collection. Passive attack data can be originated from worm propagation probing, DDos and distributed stealthy scans. Such data sets that are widely used for host clustering, includes CAIDA data set [46], Dshield data set [47], and other Honeypot data set. The Cooperative Association for Internet Data Analysis (CADIA) is an organization based at
the University of California’s San Diego supercomputer center. It collects several different types of data at geographically and topologically diverse locations, and makes the data available to the research community [46]. Similarly, DShield Internet Storm Center is an organization that collects data for different contributes for discovering trends in activity, confirming widespread attacks, or assisting in preparing better firewall rules [47].

As discussed earlier, Soldo et al. [27] adopted recommendation systems to predict victim networks based on similarly behaving malicious source IP. While theoretical sound and performing well against the DShield dataset, the recommendation system approach does not provide insights on how the attack actions happen sequentially or causally. Note that DShield data reports attack incidents on the Internet for blacklisting purposes, but does not really contain sophisticated multistage attack strategies. It serves for a different purpose of predicting victim networks from blacklisted source IP, but not predicting next attack actions among ongoing attacks in enterprise networks. Cheng et al. [48] developed a system that measures similarity between attack progressions and project into future actions based on most similar portions of the progressions seen in other attack sequences. The approach is based upon the solution for the classical Longest Common Subsequence problem, and a key novelty lies in the definition of the attack progression as a time-series of 3-digit numbers: the first digit indicates the zone distance between the source and destination IP, the second digit stands for the network protocol used, and the third digit reflects the distance between port clusters. From there, an attack sequence becomes a trajectory moving in this 3-digit space. While the idea is interesting and unique, the authors evaluated their system against the DARPA dataset, which is limited in terms of attack sophistication as discussed before.

Tables 1.4 and 1.5 summarize the aforementioned alert correlation and host clustering works with data sets.

After reviewing the data sets that widely used in intrusion detection, alert correlation, host clustering and attack prediction, we want to argue that, to the best of our knowledge,
there is no work that comprehensively evaluate the performance for different alert analysis system and take possible attack obfuscations into consideration. One of the reason is because that the widely used data sets contain no ground truth on attack obfuscations.

For the works based on DARPA 1998, and 1999 data set, they are dealing with the attacks without multiple steps and stepping stone hosts and there is no obfuscated attacks. This is because the major goal of creating these two data sets is evaluating the performance of IDS. For the works based on DARPA 2000 and GCP data set, most work are correlating
the observable into complicated attack scenarios and eliminating unrelated alerts. The variety and number of attacks in DARPA 2000 and GCP data set are also limited (one attack with five phases and two worm attacks respectively). For the works based on DEFCON data set, few attack obfuscation exists because the goal of hacking competition is capture the flag instead of to be stealthy. On the other hand, most passive and honeypot data sets, such as CAIDA data set and DShield data set are collections of malicious probings and sophisticated attack with stealthy and decoy actions may not exist.

1.6 Overview the rest of the dissertation

Combating against large-scale, sophisticated attacks requires advances on various fronts, including intrusion detection, alert correlation, host clustering, attack modeling and prediction. Advances in intrusion detection [53, 3], though not perfect, have provided significant observables or alerts that contain attributes of individual malicious actions. Alert correlation, e.g., [20, 54, 14, 13, 55, 11, 16, 56, 57], processes and groups the observed alerts based on their similarity or a pre-defined attack scenarios. Attack characterization and prediction, e.g., [19, 58, 56, 59, 35, 51, 60], aims to analyze the temporal or sequential characteristics of alert sequences of individual attack sources, so as to predict behaviors of future actions. Host clustering, e.g., [22, 23, 25, 27, 24], explores the spatial characteristics as well as packet and flow level anomalies among Internet hosts, to group them into clusters of normal, infected or botnet hosts.

While the computational techniques used to analyze network attacks are advancing, the attack tools, hacker skills and the attack strategies are also becoming more sophisticated. This calls for new techniques to characterize attacks and their strategies when the critical observables are embedded in the large volume and diverse malicious activities. In addition, dealing with the attack obfuscations, e.g., IDS evasion, stealthy and decoy attack, will be another challenge for network analysis. This chapter reviews and summarizes related work shows that there is no work to address the attack obfuscations explicitly, to the best of our knowledge.
Chapter 2 will expand the discussion on attack characterization and modeling and discuss our works that investigate how temporal and spatial analyses can be used to discover attack sources that play different roles in a network attack and to discover different attack strategies. We will show for different algorithms understand and projecting next possible actions is a daunting challenge, especially with noisy observations. Chapter 2 will end by discussing the technical gap for dealing with noisy attack sequences, which gives us the motivation of probabilistic modeling and inference for obfuscated network attack sequences.

Chapter 3 analyzes existing attack obfuscation techniques and derive attack obfuscation models from the security domain knowledge. Eventually propose a framework of modeling attack obfuscation explicitly to understand and quantitatively access the impact of obfuscation on current framework attack modeling, alert correlation and attack prediction.

To take advantage of the proposed framework, efficient algorithms are necessary to preform probabilistic inference for given attack and obfuscation model. Chapter 4 derives theorems and the algorithms to perform exact inference on proposed probabilistic model. Given the inference algorithm, the problem of impact assessment for obfuscated sequences is discussed and solved by Monte-Carlo approximation with desired accuracy.

Chapter 5 gives the simulation and results for different attack scenarios based on proposed framework. And we will demonstrate how such knowledge can be used for security analysis. Finally Chapter 6 concludes the dissertation. The detailed IDS alerts explanation, theorem proofs are given in appendix.
Chapter 2

Network attack characterization, modeling and prediction

This chapter discusses our work on attack characterization modeling and prediction and gives the motivation of the framework proposed in Chapter 3. We begin by assessing the real attack data from hacking competition [61] and network telescope data produced by UCSD [62]. Several useful features are proposed to describe different types of attacks, and used to cluster the attack hosts into different groups [26] [28]. The idea is treating hosts as instances and attacking relationship as social connections. Taking advantage of social network analysis, e.g., centrality, community structure, influence concept, possible coordinated attacks can be discovered [28]. In addition, this chapter discusses the novel attack action transition feature [63] and spatial feature [64] to discover non-trivial attack sources which are difficult for classical packet analysis to reveal. Finally, an attack prediction model is presented [51] to show the challenge of dealing with noisy attack sequences on simulated data set, which will lead to the discussion of attack obfuscation in Chapter 3.

2.1 Attack action sequence features

The mixture of organized cyber crimes, random attacks and computer virus against enterprise and government networks has led to overwhelming data to be analyzed. In this section, we start our investigation from a more controlled environment, International Capture the Flag (ICTF) hacking competition [43, 61], where participants are divided into teams
to accomplish specific attack task. This enables us to investigate how coordinated, sophisticated attacks work, which take advantage of multiple host level attack sources and are conducted by an attacker or a group of attackers.

Table 2.1 gives an example coordinated attack by listing a sequence of Snort [53] IDS alerts from real-world data. Basic alert attributes are listed, which include time, source IP, target IP and attack signature. The attack signature is the Snort description of alert which can be looked up in an IDS database to get more information about the attack action.

Table 2.1: An example of attack from real-world data

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Src. IP</th>
<th>Dest. IP</th>
<th>Snort alert description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11:02:07</td>
<td>10.13.148.213</td>
<td>10.14.0.100</td>
<td>(portscan) TCP Portscan</td>
</tr>
<tr>
<td>3</td>
<td>12:52:02</td>
<td>10.13.148.223</td>
<td>10.14.0.2</td>
<td>ICMP PING NMAP</td>
</tr>
<tr>
<td>4</td>
<td>12:52:02</td>
<td>10.13.148.223</td>
<td>10.14.0.3</td>
<td>ICMP PING NMAP</td>
</tr>
<tr>
<td>5</td>
<td>12:52:02</td>
<td>10.13.148.223</td>
<td>10.14.0.4</td>
<td>ICMP PING NMAP</td>
</tr>
<tr>
<td>6</td>
<td>12:52:02</td>
<td>10.13.148.223</td>
<td>10.14.0.5</td>
<td>ICMP PING NMAP</td>
</tr>
<tr>
<td>7</td>
<td>12:52:02</td>
<td>10.13.148.223</td>
<td>10.14.0.6</td>
<td>ICMP PING NMAP</td>
</tr>
<tr>
<td>8</td>
<td>12:52:02</td>
<td>10.13.148.223</td>
<td>10.14.0.7</td>
<td>ICMP PING NMAP</td>
</tr>
<tr>
<td>9</td>
<td>12:52:02</td>
<td>10.13.148.223</td>
<td>10.14.0.8</td>
<td>ICMP PING NMAP</td>
</tr>
<tr>
<td>10</td>
<td>12:52:02</td>
<td>10.13.148.223</td>
<td>10.14.0.9</td>
<td>ICMP PING NMAP</td>
</tr>
<tr>
<td>12</td>
<td>14:36:15</td>
<td>10.13.148.217</td>
<td>10.14.1.9</td>
<td>(portscan) TCP Portscan</td>
</tr>
<tr>
<td>17</td>
<td>17:00:22</td>
<td>10.13.148.150</td>
<td>10.14.1.17</td>
<td>WEB-MISC http directory traversal</td>
</tr>
</tbody>
</table>

There is a coordination among attack sources described in Table 2.1. According to the alert description, Alerts #1–12 are reconnaissance actions, and Alerts #13–18 are web

---

<sup>1</sup>In general, it is hard to infer about the attacker(s), because the basic unit of most observable is at host level (e.g., IP address) which can be easily spoofed.
server vulnerability attempt actions. Although these two steps are closely related, *i.e.*, vulnerability attempts depend on reconnaissance, but they originated from different sources. Attack source 10.13.148.218 is utilized for probing across the target space within the subnet 10.14.0.x. After the reconnaissance, 10.13.148.210 and 10.13.148.150 are utilized to try different vulnerabilities on web services.

In fact, this is a simple example where the source IP subnet can be indicative of coordinated behavior. This property occurs because the example is extracted from hacking competition and participating teams are assigned specific range of source IPs. In the real world where IP spoofing and zombie machines are common, inferring coordinated attack sources by source IP alone will not be practical.

The use of multiple attack sources, possibly spoofed, make detection of coordinated attack team more difficult. Table 2.2 includes a sequence of alert data from the ICTF hacking competition [43]. Alerts triggered by two teams, *ENOFLAG* (EF) and *Chocolate Makers* (CM), are listed in the table. Both teams utilize multiple attacking sources to work together for achieving a certain goal. Basic alert attributes are listed, and the last column shows the team assignment for each attack source.

The two teams shown in Table 2.2 have different strategies, *i.e.*, the attacks performed by two teams are different in both spatial and temporal domains. For the reconnaissance stage, team EF is more *centralized*. EF mainly uses three hosts for comprehensive reconnaissance. On the other hand, team CM is more *distributed*. Eleven sources are used for probing the whole target space. If we use vertices to represent hosts and edges represent attacks, the graphical representations are very different for two teams, as shown in Fig. 2.1.

In the temporal domain, the sequences of attack sources by the two teams are also different from each other. Team EF’s members perform TCP port sweep and port scan

---

2Note that, there are missing steps between Alert #12 and Alert #13. Vulnerability attempts are conducted against 10.14.1.17, but no alerts indicates the target has been probed and discovered. This is because Snort alert is one type of observed evidence, and it may not be comprehensive since current IDS does not perform perfect detection.

3Table 2.2 only lists 8 alerts for each team. Figure 2.1 represents all actions from the two teams.
Table 2.2: Examples of coordinated attacks with different strategies

<table>
<thead>
<tr>
<th>Src. IP</th>
<th>Dest. IP</th>
<th>Snort alert description</th>
<th>Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.13.1.86</td>
<td>10.100.113.48</td>
<td>(portscan) TCP Filtered Portscan</td>
<td>EF</td>
</tr>
<tr>
<td>10.13.1.86</td>
<td>10.120.113.42</td>
<td>(portscan) TCP Filtered Portscan</td>
<td>EF</td>
</tr>
<tr>
<td>10.13.1.86</td>
<td>10.199.113.36</td>
<td>(portscan) TCP Filtered Portscan</td>
<td>EF</td>
</tr>
<tr>
<td>10.13.1.32</td>
<td>10.100.113.24</td>
<td>(portscan) TCP Filtered Portscan</td>
<td>EF</td>
</tr>
<tr>
<td>10.13.1.32</td>
<td>10.100.113.9</td>
<td>(portscan) TCP Filtered Portscan</td>
<td>EF</td>
</tr>
<tr>
<td>10.13.1.32</td>
<td>10.100.113.3</td>
<td>(portscan) TCP Filtered Portscan</td>
<td>EF</td>
</tr>
<tr>
<td>10.13.1.32</td>
<td>10.100.113.8</td>
<td>(portscan) TCP Filtered Portscan</td>
<td>EF</td>
</tr>
<tr>
<td>10.13.1.32</td>
<td>10.100.113.7</td>
<td>(portscan) TCP Filtered Portscan</td>
<td>EF</td>
</tr>
<tr>
<td>10.33.1.13</td>
<td>10.100.133.1</td>
<td>ICMP PING NMAP</td>
<td>CM</td>
</tr>
<tr>
<td>10.33.1.12</td>
<td>10.100.133.2</td>
<td>ICMP PING NMAP</td>
<td>CM</td>
</tr>
<tr>
<td>10.33.1.16</td>
<td>10.100.133.3</td>
<td>ICMP PING NMAP</td>
<td>CM</td>
</tr>
<tr>
<td>10.33.1.16</td>
<td>10.100.133.4</td>
<td>ICMP PING NMAP</td>
<td>CM</td>
</tr>
<tr>
<td>10.33.1.17</td>
<td>10.100.133.5</td>
<td>ICMP PING NMAP</td>
<td>CM</td>
</tr>
<tr>
<td>10.33.1.12</td>
<td>10.100.133.6</td>
<td>ICMP PING NMAP</td>
<td>CM</td>
</tr>
<tr>
<td>10.33.1.20</td>
<td>10.100.133.7</td>
<td>ICMP PING NMAP</td>
<td>CM</td>
</tr>
<tr>
<td>10.33.1.16</td>
<td>10.100.133.8</td>
<td>ICMP PING NMAP</td>
<td>CM</td>
</tr>
</tbody>
</table>

(a) Spatial relationships of attacks by EF  
(b) Spatial relationships of attacks by CM

Figure 2.1: Example of spatial feature for coordinated attacks

against the target space. For team CM, six attack sources collaboratively use NMAP to probe the target space. Therefore, for all alerts listed in the table, building the action sequence for each attack source will lead to following results: team EF will have two
sequences with length 4 and 6 respectively. Team CM will have ten sequences, which with only one action.

Classical works may not be effective for treating coordinated attacks because the actions are originated from different attack sources. For alert correlation, with missing observables, it is very challenging to perform correlation by examining the pre-condition and post-condition. For example, consider the attack conducted by team CM given in Table 2.2, coordinated reconnaissance, the pre/post-condition and attack scenario based approach will fail to group the coordinating attackers. For attack characterization and prediction, Bayesian networks and sequence modeling may be also inefficient in the presence of missing actions, because coordinated attacks will break the dependence structure of the model (for both Bayesian networks and Markov model). Host clustering may work for same cases where attack sources behave similarly, such as botnet DDoS attack, but may not work for the general case of coordinated attack.

Classical works may not be effective on coordinated attacks because they mostly focus on examining individual attack source’s actions, which will only give local information and ignores the global information on the whole network. On the other hand, classical works suggest Markov models and graph-based analysis can greatly benefit the analysis for large-scale attacks. Using spatial features can provide additional insights about the relationship of the attack sources and the use of sequence modeling can help for characterize the attack behavior.

To analyze the spatial feature, an Attack Social Graph (ASG) can be defined as follows to represent attacks in a given network [26] as follows:

An Attack Social Graph $\text{ASG}_T(V, E)$ is a directed graph representing the malicious traffic within a time interval $[0, T]$, where a vertex $v \in V$ is a host, and an edge $e_{(u,v)} \in E$ exists if attacks are observed from $u$ to $v$. Edge direction is from the attacking host to its target.

In addition to the hacking competition data, ASGs exhibit many interesting patterns
on other real-world data, which can provide us important insights on understanding coordinated attacks. For example, consider two ASG subgraphs shown in Fig. 2.2 which are extracted from CAIDA Network Telescope data [62, 65]. In Fig. 2.2(a) and 2.2(b), the attack sources circled with the dotted line act similarly to attack one and only one target. This is unlikely to happen by chance due to the large target space. On the other hand, the two attack sources circled with the solid line in Fig. 2.2 are suspicious. They attack several heavily attacked targets and are the common denominator among the all attacking sources. One possible interpretation of such a situation is that this is a coordinated attack. The sources in dotted circles are zombie machines controlled by the two sources in the solid circles.

Figure 2.2: Example of ASG analysis

Based on the intuition given in Fig. 2.2, two approaches can be applied to analyze ASG. The first approach is to calculate certain graph properties, such as centrality distribution [66]. The second approach is to define the labels that have specific meanings, and analyze attack sources with label sequences. For the spatial labeling approach, Attack Conspirator, Heavily Attacked Target (HAT) can be defined as follows:

Let $T_u \doteq \{v \mid e_{(u,v)} \in E, v \in V_t\}$ be the targets attacked by $u \in V_s$, the attack conspirators of $u \in V_s$, denoted as $C_u$, is a set of vertices: $C_u \doteq \{v \mid T_u \cap T_v \neq \emptyset, v \in V_s\}$.

---

*In the real-world data set, there is no ground truth suggesting how attack sources are coordinated. We collect evidence to support our assumption. In this two example, inter-arrival time and geographical information are consistent with the assumption of leader and zombie hosts.*
where $V_t$ and $V_s$ represent the set of targets and attacking sources respectively.

A heavily attacked target is a vertex $v \in V_t$, s.t. $d_{in}(v) \geq H$, where $H$ is a pre-selected threshold.

Using these definitions, each source can be characterized by its spatial pattern. In particular, we can label the attack sources according to their behavior, and examine their roles in coordinated attack. There are many disjoint subgraphs in each ASG, the first level of labeling is to differentiate the sources based on the type of subgraph they reside: \textit{one-to-one}, \textit{one-to-many}, \textit{many-to-one} and \textit{many-to-many} relationships. The first three ASG subgraph types are relatively easy to analyze in the security context. \textit{One-to-one} relationships could indicate the sensor was triggered by chance\footnote{The spatial pattern example is extracted from UCSD data set, in which the alert can be triggered by chance such as mis-configuration. In hacking competition data, we do not have such a case.} or focused attack on specific target. \textit{One-to-many} relationships represent service scanning on a set of targets. \textit{Many-to-one} relationships represent DDoS or coordinated attacks. In the case of \textit{many-to-many} subgraphs, additional factors are needed to differentiate the attack sources. Specifically, we use the HAT to differentiate whether a source is part of a potentially coordinated attack, \textit{i.e.}, unlikely to happen by chance. The attack sources that do not have any HAT are further differentiated depending on whether they have any Heavily Attacking Conspirator (HAC). The idea here is to examine whether the source, which can be attacking because of mis-configuration or it has a specific target, has a conspirator that is part of a potentially coordinated attack. Note that, the sources that have at least one HAT must have HACs. We further differentiate sources with only one HAT or multiple targets. Figure 2.3 summarizes our labeling approach.

Spatial relationships among attack sources and targets are important features of coordinated attacks. With the degree centrality measure and labeling scheme, one can characterize the possible role for given attacking sources. Figure 2.4 gives the result of degree centrality based clustering results. In Du’s work [26], connectivity attributes and hierarchical clustering are used for categorize the coordinated attack. Figure 2.4(a) and 2.4(b) are the clustering results at the top level and Fig. 2.4(c) and 2.4(d) show the corresponding
attack sources in the ASG.

Figure 2.3: Attack source labeling

(a) Dendrogram of attack sources

(b) Cluster of attack sources

(c) Cluster C in ASG

(d) Cluster B in ASG

Figure 2.4: Degree centrality based attack sources clustering
By analyzing the clusters from Fig. 2.4(a), we find several sets of interesting collaborative attack patterns. First, the five feature points within cluster $C$ are actually five distinct attack sources that attack a large number of targets within the network monitored by the Network Telescope. These five sources are outliers on the 2D plane (see cluster $C$ in Fig. 2.4(b)), it is shown in Fig. 2.4(c) to identify these sources in the ASG. Second, some attack sources form cluster $B$ since these sources all have a “special” conspirator 0.211.214.160, which is an attack sources in Cluster $C$ and has out-degree 18,920. Being a conspirator of such hosts makes their features significantly different from others and thus forms a cluster. Figure 2.4(d) identifies some of the Cluster $B$ sources in a zoom-in view of the corresponding ASG.

For the spatial labeling approach, by analyzing the joint probabilistic distribution over time of the spatial labels, the patterns can be extracted from a very large training set (2,322,134 attack sources). For convenience, let $d(p)$ denote the number of unique values in a label pattern $p$. We define this value as the diversity of an attack sequence. Furthermore, let $\{a,b\}$ denote the label patterns with $d(p) = 2$ that contain $x$ and $y$ regardless of the order over which they occur. Similarly, we can define $\{a,b,c\}$ and $\{a,b,c,d\}$ for patterns with $d(p) = 3$ and $d(p) = 4$, respectively. Figure 2.5 shows the probabilities of occurrence for label patterns with $d(p)=1, 2, 3$ and 4., for which there are a total of 8, 28, 56, 78 patterns in each set, respectively. The most popular label pattern sets are highlighted in the subfigures. These patterns represent the cases where DDoS and distributed scanning occurred and sometimes switched targets.

The connectivity features are also effective for differentiating different attack behavior. Consider two different probing behaviors: web probing and share probing. The web probing attacks ports 80, 8000 and 8080, and it is widely used to identify live targets at the beginning of the reconnaissance. The destination ports of share probing include ports 139 and 445, and it is also used for host discovery and OS fingerprinting. The spatial label can

---

6For anonymity reason, in this chapter, the first byte of IP address from real-world data (UCSD CAIDA data set) is masked with 0.
effectively differentiate these two different behaviors.

Figure 2.6 is an example to compare the target IPs, target ports, and label sequences
of two attack sources from each group. The subfigures (a), (b) and (e) on the left show the behavior of *web probe* and those on the right (c), (d) and (f) show the share probe for comparison. Note that target IP and target ports are two key factors to describe an attacking behavior. The target IPs and ports are shown in a $2^{12} \times 2^{12}$ IP space and a $2^8 \times 2^8$ port space. Subfigure (e), (f) provide the corresponding label sequence.

Comparing *web probe* with *share probe*, their target port and IP selections are distinct. In addition to ports 80, 8080 and 8000, *web probe* also attacks other variations, such as ports 808, 1080 and 2080. *share probe* only attacks ports 139 and 445. In terms of target IP, *web probe* often explores randomly over the IP space. *share probe* focuses on scanning individual subnets – the strips in (c) represent the continuous target IPs.

The label sequences of different strategies are very different from one to another. For *web probe*, the majority of the labels are Label-2s and Label-5s, along with other non-zero labels spreading over 24 hours. In such case, it is more likely to be an automatic script attack, and have few HATs (Label-6 and Label-7). For *share probe*, the majority of the labels are Label-0s with occasional occurrences of Label-2s, Label-5s and Label-7s. This suggests *share probe* is somewhat sporadic with short breaks, which is consistent with it scanning on a subnet basis. Furthermore, Label-7 will also occur sometimes because the concentration of target IPs is likely to hit some HATs.

As discussed earlier, the challenge of discovering coordinated attacks is due to the stealthy actions distributed across multiple attack sources. The spatial features can effectively discover such sophisticated attacks buried in the overwhelming data. For example, consider an attack source 0.86.249.218, which sent 59 malicious packets over 24 hours in the Network Telescope data set [65] (the whole data set contains over $10^9$ packets).

Table 2.3 gives the details of the first 10 packets if only traffic volume and target range are considered. There are 21 distinct targets, seemingly randomly selected, and 4 distinct ports (ports 80, 8080, 808 and 8000). There is no evidence suggesting this attack source is important and worthy of further investigation. However, the label sequence for this attack source suggests it could be indicative of advanced attack, where the hacker is switching
between compromised hosts, which can be exhibited by the ASG subgraphs. Therefore, we further verify our discoveries by examining the corresponding ASG over time. Figure 2.7(a) to 2.7(d) give the ASG subgraphs in four consecutive time frames $T_6$ to $T_9$. The attack source 0.86.249.218 is highlighted with the solid circle. For better visualization, only the attack source, its targets and its conspirators are shown. The extracted ASGs show that the attack source is not a inconsequential attacker. Although the malicious traffic volume and the out-degree of the attack source are both small, most of its targets are heavily attacked. During the entire 20 minutes, the attack methodically narrowed down the point-of-interest and increased the volume of attacks on specific targets. At $T_6$, it attacked 6 targets, and 4 of them were also attacked by other sources. At $T_7$, it reduced the range of targets by 1. Among the remaining 5 targets, 2 were attacked by others. For $T_8$, the target declined to 4, but 2 of them were heavily attacked by 7 and 14 other sources, respectively. In addition, there were 3 suspicious sources that attacked both HATs. Note that, in a large targets space, it is unlikely to have 4 hosts simultaneously select the same two targets. One can hypothesize that these attack sources are controlled as zombie machines or collaborated together for the attack. Finally, at $T_9$ the source reduced the attack to only three targets and focused on one to perform comprehensive attacks using multiple hosts. This example illustrates that an attack source with transitions between Label-6 and Label-7 can be critical
and worth further investigation.

Figure 2.7: Label Sequence and ASG Subgraphs for Attack Source 0.86.249.218

2.2 Predict possible attack actions

Connectivity features as well as the computational models based on the proposed features allow the discovery of notorious attacking sources and non-trivial attacking sources. On the other hand, one may want to focus on one specific attacking source and predict what next attack action may happen. Sequence modeling and prediction will allow us for such tasks.

As discussed in Chapter 1, Variable Length Markov Model (VLMM) has been used to adaptively extract patterns in cyber attacks [35]. Though showing promising results, the VLMM approach focused on extracting patterns and projecting based on specific attributes of IDS alerts. A projection made based on one attribute may not match to that based on another. Consider a simplified example shown in Table 2.4.

The VLMM predictions will suggest ‘192.168.3.x,’ ‘UDP,’ and ‘WEB-MISC http directory traversal’ as the top choices for attributes, $tip$ (subnet), $prt$, and $dsc$, respectively. A granularity at the subnet level is used for better capturing the movements of the attack across the accessible regions (collision domains that are typically defined similarly by router firewall rules). The per attribute choices, however, cannot be combined directly since: (1) the ‘HTTP request’ does not utilize ‘UDP’ protocol and (2) the subnet 192.168.3.x in the test network does not contain a web server. Therefore, it is necessary to develop an intelligent and robust combination.
Table 2.4: Per attribute prediction example by VLMM.

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Possible Values</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>tip</em> (subnet)</td>
<td>192.168.1.x</td>
<td>0.1602</td>
</tr>
<tr>
<td></td>
<td>192.168.3.x</td>
<td><strong>0.8141</strong></td>
</tr>
<tr>
<td></td>
<td>192.168.20.x</td>
<td>0.0256</td>
</tr>
<tr>
<td><em>prt</em></td>
<td>TCP</td>
<td>0.4318</td>
</tr>
<tr>
<td></td>
<td>UDP</td>
<td><strong>0.5681</strong></td>
</tr>
<tr>
<td><em>dsc</em></td>
<td>SMTP sendmail 5.5.5 exploit</td>
<td>0.4349</td>
</tr>
<tr>
<td></td>
<td>WEB-MISC http directory traversal</td>
<td><strong>0.5513</strong></td>
</tr>
<tr>
<td></td>
<td>FTP adm scan</td>
<td>0.0138</td>
</tr>
</tbody>
</table>

The per alert attribute VLMM predictions need to be transformed to projection scores with respect to the target machines for combination. Two projections scores, *Proj*$_i$ and *Proj*$_d$, are derived based on VLMM predictions based on ‘*tip* (subnet)’ and ‘*dsc*’, respectively. The *Proj*$_i$(h) represents how likely h may be targeted next according to the order of subnets that have been attacked, and is referred to as the IP projection. The *Proj*$_d$(h) is based on the sequence of service exposure being exploited, referred to as the exposure projection. Let $p_i(\cdot)$ and $p_d(\cdot)$ be the probabilities derived based on VLMM with respect to ‘*tip* (subnet)’ and ‘*dsc*’, respectively. Also let $N(h)$ be the subnet h resides, and $E(h)$ be the set of vulnerability exposures h contains.

$$
\begin{align*}
*Proj* _i(h) &= p_i(N(h)) \\
*Proj* _d(h) &= \sum_{i \in E(h)} p_d(i)
\end{align*}
$$

To combine *Proj*$_i$ and *Proj*$_d$, we chose to use fuzzy inference, for its effectiveness as an information fusion tool that mimics human analytics [67, 68]. There are two types of fuzzy inference systems, Mamdani and Sugeno [69]. We utilized Sugeno System since it guarantees continuity of output surface. The system has two components: the membership functions for fuzzifying inputs and the inference rules for combination.

Membership functions can be derived from examining the distribution of the inputs. Figure 2.8 shows a scatter plot of *Proj*$_i$ versus *Proj*$_d$ for all targets throughout a training dataset. It shows that the exposure projection is mostly evenly distributed between 0 and 1,
while there is a concentration of inputs between 0 and 0.1 for $Proj_t$. Interestingly, different experimental datasets are tested and the results scatter plots are similar. This is due to the fact that while many machines are hidden behind firewalls in most instances, the estimation based on service exposure do not account for such and produce uniform distribution with a sufficiently large set of services and machines.

![Figure 2.8: A scatter plot of $Proj_t$ versus $Proj_d$ of all hosts](image)

In order to differentiate within the concentration region of [0,0.1], more membership functions are used for $Proj_t$. Figure 2.9 shows the membership function design: 5 membership functions and 3 membership functions are used to fuzzify the $Proj_t$ and $Proj_d$, respectively.

Given the membership functions, there are 15 inference rules combining $Proj_d$ and $Proj_t$. The rules places a higher emphasis on $Proj_t$ because human analysts typically give more credit to where the attack has reached instead of what services have been attacked. Table 2.5 gives a tabular view of the rules. The elements $(a_{ij})$ in Table 2.5 are aggregated to determine the overall projection score ($projection$) based on the antecedents $(u_{ij})$ defined
Figure 2.9: The fuzzy membership functions

using the fuzzy logic AND operator.

\[ \text{projection} = \frac{\sum_{i=1}^{5} \sum_{j=1}^{3} u_{ij} \cdot a_{ij}}{\sum_{i=1}^{5} \sum_{j=1}^{3} u_{ij}} \]

where

\[ u_{ij} \triangleq \mu_i(\text{Proj}_t) \cdot \mu_j(\text{Proj}_d), \]

and \( \mu_i \) and \( \mu_j \) denote the \( i^{th} \) and the \( j^{th} \) membership function of \( \text{Proj}_t \) and \( \text{Proj}_d \).

Table 2.5: Fuzzy inference rules used to combine VLMM outputs

<table>
<thead>
<tr>
<th>( \text{Proj}_d )</th>
<th>low 1</th>
<th>low 2</th>
<th>low 3</th>
<th>medium</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>low</td>
<td>0.0</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.8</td>
</tr>
<tr>
<td>medium</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.8</td>
<td>1.0</td>
</tr>
<tr>
<td>high</td>
<td>0.4</td>
<td>0.6</td>
<td>0.8</td>
<td>0.8</td>
<td>1.0</td>
</tr>
</tbody>
</table>

The resulting overall system can be illustrated using the input/output surface plot shown in Figure 2.10. Note that the surface quickly rises in the dense region of IP exposure, to differentiate the inputs and reduce false positives. The changes with respect to the exposure projection is gradual to reflect the uniform distribution described earlier.

Attack prediction by VLMM prediction and fuzzy combination (F-VLMM) were tested via simulation. Simulated multi-stage attacks were generated using the simulator developed by Kuhl et al. [70] on two networks, shown in Figures 2.11 and 2.12. The two networks
were designed to represent two types of enterprise networks. Network A (6 subnets, 11 servers and 4 clusters of hosts (24 hosts total), containing 31 services (15 types total), interconnected via 4 routers) represents the case where each service is implemented in only one or few dedicated servers, while Network B implements about 10 instances per service. Network B (9 subnets, 23 servers and 8 clusters of hosts (130 hosts total), containing more than 300 services (37 types total), interconnected via 8 routers) also has more total machines and service types, representing a larger network with a more redundant configuration. Both networks were configured with firewall rules restricting traffic between different parts of the network. The entire sets of rules are too large to be included in this paper, but the general idea is that the departments (shaded boxes in the figures) that have their own servers or reside deeper in the network will have more restricted rules.

A total of 1,000 random attacks containing 6,854 alerts were generated for Network A, and 1,500 attacks, composed of 11,697 alerts, were generated for Network B. The data set contains a mixture of ‘stealthy’ attacks where some steps are not observed, and attacks with different ‘efficiency’ level. An attack is most efficient if it utilizes the minimum number of stepping stones to get to the final target in the network.

For Network B, specific data sets with efficiency level of 0.8, 0.5, and 0.3 were created.
Figure 2.11: Network A used for attack prediction simulation

For the details of the parameter settings, the readers may refer to Kuhl et al. [70]. A number of targets were chosen for both data sets, representing a broad range of servers and hosts in different departments within the corresponding network.

Given the datasets, the algorithms were tested to determine whether they can accurately project the next attacked target given the already observed events. Cyber attack projection aims at providing a ranked list of projected targets, instead of a prediction of what exactly will happen next. Therefore, the performance of the algorithm was evaluated by examining the percentile ranking of the attacked target one step prior to it being attacked. Because there could be ties in the projection scores, the results presented in this paper are shown in the form of \([lower, upper]\), representing the interval of percentile ranking of targets received the same score as the attacked target.

Table 2.6 shows the average percentile rank in the datasets by running the F-VLMM
Figure 2.12: Network B used for attack prediction simulation

The results show that both algorithms work well for Network A and Network B. The F-VLMM performs well for both networks because (1) the VLMM captures the attack patterns in both targeted subnets and targeted service exposures, and (2) the fuzzy combination successfully differentiates the attacked targets from other targets. Figure 2.13 shows the number of targets receiving different threat scores when all the targets are considered (top) and only the attacked targets are considered. The majority of attacked targets receive high threat scores. The same fuzzy functions and rules are used for different datasets.

Table 2.6: Projection performance achieved by F-VLMM for the various datasets

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[81%,88%]</td>
<td>[85%,88%]</td>
<td>[84%,88%]</td>
<td>[81%,84%]</td>
<td></td>
</tr>
</tbody>
</table>

Note that very little simulation or analytical results have been reported for cyber attack projection, and there exist no public real multistage cyber attack dataset. Our discussion of the results, thus, focuses on the benefits and limitations of the presented algorithms, but does not suggest exact performance if used in real networks.
datasets, showing its robustness.

While it may not be obvious from the overall results, the F-VLMM generally performs better for efficient attacks. Essentially, efficient attacks use only specific stepping stones without deviating from reaching the final goal. Therefore, patterns are easier to capture. There are instances where attacks deviate from the identified patterns, and the F-VLMM could perform not as well, at least temporarily. Specific case studies are presented next to illustrate the pros and cons of proposed approaches.

In order to provide a deeper understanding of the algorithms on different types of attacks, we presents a case study with three attacks that target the same mail server residing in Department F in Network B. Table 2.7 shows the attack steps (only sip and tip are shown) of a high efficiency attack. The attack starts with compromising the external mail server (Step 1-3), then tries to access an internal server (Step 4). After that, a member of the host cluster in Department A (Step 5-7) is compromised to access Department C and E, to reach the final target.

Also shown in Table 2.7 are the percentile rank intervals of the targeted machines one
Table 2.7: A high efficiency attack with percentile rank interval achieved by F-VLMM

<table>
<thead>
<tr>
<th>Step</th>
<th>Source IP</th>
<th>Target IP</th>
<th>Percentile Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9.5.231.72</td>
<td>192.168.1.3</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>237.22.202.140</td>
<td>192.168.1.3</td>
<td>[96.78%,100%]</td>
</tr>
<tr>
<td>3</td>
<td>178.87.46.91</td>
<td>192.168.1.3</td>
<td>[93.75%,96.88%]</td>
</tr>
<tr>
<td>4</td>
<td>192.168.1.3</td>
<td>192.168.2.6</td>
<td>[96.88%,100%]</td>
</tr>
<tr>
<td>5</td>
<td>192.168.1.3</td>
<td>192.168.2.8</td>
<td>[96.88%,100%]</td>
</tr>
<tr>
<td>6</td>
<td>192.168.1.3</td>
<td>192.168.2.9</td>
<td>[93.75%,96.88%]</td>
</tr>
<tr>
<td>7</td>
<td>192.168.1.3</td>
<td>192.168.2.18</td>
<td>[93.75%,96.88%]</td>
</tr>
<tr>
<td>8</td>
<td>192.168.2.18</td>
<td>192.168.4.22</td>
<td>[96.75%,96.88%]</td>
</tr>
<tr>
<td>9</td>
<td>192.168.4.22</td>
<td>192.168.6.111</td>
<td>[90.63%,93.75%]</td>
</tr>
<tr>
<td>10</td>
<td>192.168.6.111</td>
<td>192.168.7.9</td>
<td>[93.75%,96.88%]</td>
</tr>
</tbody>
</table>

step before each attack. Note that this is extracted from the overall results shown in overall results. It is clearly evident that F-VLMM consistently performs exceptionally for this multistage attack with the percentile ranked above 90% and many ranked above 95%. This suggests that VLMM can almost perfectly capture the pattern exhibited by attacks that go straight to the target 192.168.7.9 even if it is hidden behind 4 subnets: 192.168.1.x, 192.168.2.x, 192.168.4.x, and 192.168.6.x (i.e., 3 layers of firewalls).

Table 2.8 shows a ‘less efficient’ attack that has the same final target 192.168.7.9 as the previous attack. The projection performance by F-VLMM is significantly lower for Steps 5, 9, 10, 13, and 14. Step 9 attempted to attack the server 192.168.2.2 while it has already penetrated deeper into 192.168.4.x subnet. Typically such an activity is done prior to further penetration attempts. For Steps 10 and 13, the attack probes into Subnets 192.168.5.x and 192.168.3.x even though they do not contain the target machine. The majority of the dataset, however, contains high efficiency attacks and the model adaptively trained will not rank those seemly unrelated victims high. Because a wide variety of subnets have been visited, F-VLMM is not able to project accurately the real target, and, hence, performs poorly in the final step.

Table 2.9 illustrates a stealthy attack that has the same target as the previous cases, but can perform some intermediate steps without being detected. There is one or more missing
Table 2.8: A low efficiency attack with percentile rank interval achieved by F-VLMM

<table>
<thead>
<tr>
<th>Step</th>
<th>Source IP</th>
<th>Target IP</th>
<th>Percentile Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9.5.231.72</td>
<td>192.168.1.3</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>237.22.202.140</td>
<td>192.168.1.3</td>
<td>[90.63%,93.75%]</td>
</tr>
<tr>
<td>3</td>
<td>178.87.46.91</td>
<td>192.168.1.3</td>
<td>[93.75%,96.88%]</td>
</tr>
<tr>
<td>4</td>
<td>192.168.1.3</td>
<td>192.168.2.6</td>
<td>[96.88%,100%]</td>
</tr>
<tr>
<td>5</td>
<td>192.168.1.3</td>
<td>192.168.2.6</td>
<td>[50.00%,53.13%]</td>
</tr>
<tr>
<td>6</td>
<td>192.168.1.3</td>
<td>192.168.2.9</td>
<td>[84.38%,87.50%]</td>
</tr>
<tr>
<td>7</td>
<td>192.168.2.9</td>
<td>192.168.4.35</td>
<td>[81.25%,84.38%]</td>
</tr>
<tr>
<td>8</td>
<td>192.168.2.9</td>
<td>192.168.4.16</td>
<td>[96.88%,100%]</td>
</tr>
<tr>
<td>9</td>
<td>192.168.2.9</td>
<td>192.168.2.2</td>
<td>[37.5%,40.63%]</td>
</tr>
<tr>
<td>10</td>
<td>192.168.2.2</td>
<td>192.168.5.5</td>
<td>[40.63%,43.75%]</td>
</tr>
<tr>
<td>11</td>
<td>192.168.2.2</td>
<td>192.168.4.40</td>
<td>[84.38%,87.5%]</td>
</tr>
<tr>
<td>12</td>
<td>192.168.4.40</td>
<td>192.168.6.5</td>
<td>[84.38%,87.5%]</td>
</tr>
<tr>
<td>13</td>
<td>192.168.2.2</td>
<td>192.168.3.17</td>
<td>[18.75%,21.88%]</td>
</tr>
<tr>
<td>14</td>
<td>192.168.6.5</td>
<td>192.168.7.9</td>
<td>[3.13%,6.25%]</td>
</tr>
</tbody>
</table>

steps between Steps 3 and 4. Step 3 compromised 192.168.2.7, but Step 4 shows an internal machine 192.168.2.9 being used as a stepping stone. Assuming there is no insider threat, 192.168.2.9 must be compromised before it attacks 192.168.4.20, which, in turns, is used to attack other machines. Because the data is generated via a simulator that contains ground truth of the attack, we know there is no insider threat and there is indeed an attack step not being detected. The missing step has affected the F-VLMM in recognizing the pattern and thus the projection is not as accurate as in the first case.

Table 2.9: A stealthy attack with percentile rank interval by F-VLMM

<table>
<thead>
<tr>
<th>Step</th>
<th>Source IP</th>
<th>Target IP</th>
<th>Percentile Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9.5.231.72</td>
<td>192.168.1.3</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>237.22.202.140</td>
<td>192.168.1.4</td>
<td>[96.88%,100%]</td>
</tr>
<tr>
<td>3</td>
<td>192.168.1.3</td>
<td>192.168.2.7</td>
<td>[96.88%,100%]</td>
</tr>
<tr>
<td>4</td>
<td>192.168.2.9</td>
<td>192.168.4.20</td>
<td>[75%,78.13%]</td>
</tr>
<tr>
<td>5</td>
<td>192.168.4.20</td>
<td>192.168.6.113</td>
<td>[75%,78.13%]</td>
</tr>
<tr>
<td>6</td>
<td>192.168.6.113</td>
<td>192.168.7.9</td>
<td>[81.25%,84.38%]</td>
</tr>
</tbody>
</table>
2.3 Technical gap for dealing noisy attack sequences

This chapter presents our works on attack characterization and prediction. Drawing the analogy from social network analysis, an Attack Social Graph is defined to represent the relationship between attack sources. Applying the notion of degree centrality and agglomerative hierarchical clustering, various types of collaborative attack, or spatial patterns are discovered. These spatial patterns enable a labeling scheme for attack sources over time, resulting in an integrated spatial and temporal model for attack sources. Markov models are developed to differentiate and infer cyber attack strategies worthy of further investigation. The experiment results using Network Telescope and ICTF data show that the integrated spatial and temporal analyses can provide additional insights for high impact attacks that are not trivial by applying traditional statistical or anomaly analyses. Our work [26] [63] [51] offers a viable approach to analyze attack strategies by exploring not only the sequential relationship between the attack actions performed by an individual attack source, but also the relationships exhibited in attack actions among attack sources.

On the other hand, based on the features extracted, network security can benefit from projection of multistage cyber attacks, where likely future targets can be identified for timely responses. Projecting cyber attacks requires the history of patterns exhibited in attacks’ progression in the network. While previous work introduced attack assessments based on these characteristics, our work revisits them and presents a fuzzy ensemble techniques to combine the attack projection estimates. Thorough analysis via simulation are presented to provide insights toward ensemble characterization of multistage attacks. The F-VLMM predictor is developed to effectively capture sequential patterns of attack progression and uses Fuzzy inference to combine estimates based on subnet and services visited. Simulation results have shown F-VLMM’s superior performance for ‘high efficiency attacks’. For attacks deviating from the extracted pattern due to observation noise, decoy, or stealthy attack actions, the F-VLMM have limited performance.

Despite making some progress on characterizing, modeling and prediction of network attacks, there is still a technical gap for dealing with noisy attack sequences. In fact, even
evaluating the performance of alert correlation is an open challenge and there are few work compares the performance and robustness of different alert correlation systems [10]. This is partially due to the lack of publicly available data set that contains variations of network structure and vulnerabilities (as discussed in Chapter 1.5). To the best of our knowledge, Haines et al. [10] was the only one who presented a comparative study, showing a large performance variation by different alert correlation systems: the combined attack recognition rate varies between 13.98% and 94.62% and the combined target recognition rate varies between 17.14% and 48.57%. A key reason for the low recognition rates and performance variation comes from the imperfectly observed attack sequences that contain intentionally or unintentionally removed, injected, or altered observations. These noisy action observations can negatively impact the recognition or classification of the attack sequences in accordance to the Common Attack Pattern Enumeration and Classification (CAPEC) [71] or other attack models.

The lessons we learned from attack action prediction and the technical gap of dealing with noisy attacks give us the motivation of investigating attack obfuscations and formally model the attack obfuscation explicitly to enable us to understand and quantitatively assess the impact of obfuscation on current framework of attack modeling, alert correlation and attack prediction. The formal framework of modeling the joint distribution of clean attack sequence and obfuscated attack sequence will be presented in the next chapter. Further, by carefully studying attack obfuscation strategies, different types of obfuscation models will be developed and analyzed, whereas the analysis will be enabled by efficient inference algorithms presented in Chapter 4.
Chapter 3

Modeling obfuscated attack sequences

3.1 Attack obfuscation and countermeasures

3.1.1 Review of network attack obfuscation

Ptacek [72] pointed out that because of the inherent problems of Network Intrusion Detection System (NIDS), attack obfuscations are inevitable. The key problem is that NIDS is only a monitor system that mimic the target system response. It is almost impossible to mimic various types of the operating system, TCP/IP stack implementation as well and different applications. In addition, the widely used NIDSs are signature-based, i.e., keeping a database of malicious signatures and apply pattern matching techniques for classification of observed actions. Therefore, it is very possible to perform attack obfuscation by breaking the pattern-matching engine in NIDS.

Packet-level obfuscation takes advantage of the knowledge of TCP/IP stack to perform stealthy or decoy actions. Source IP spoofing [73] is a widely used technique to hide the real identity of the attacker. Moreover, more and more compromised machines are utilized as stepping stones to conduct complicated attacks [3]. Taking advantage of compromised hosts, the attacker can easily hide crucial actions and/or inject irrelevant actions to distract analysts with a large number of actions from various origination.

In addition, it is also possible for attackers to inject noisy attack actions because of public available malicious signatures. The idea of noisy action injection is trying to break the alert analysis engine (causal relationship) but not NIDS. Separating the casual relationship between attack actions can be effective to mislead security analyst, e.g., mis-classify a
severe intrusion incident into a script kiddie scenario.

Figure 3.1 gives an example of a Snort [53] rule on *RPC sadmind vulnerability attempt*. There are hex code in content attribute to describe the attack signature. Table 3.1 is the payload to exploit RPC Sadmind vulnerability, which explains the hex code shown in Fig. 3.1. In this example, the attacker can use a hex editor to create a binary file that contains the signature. After establishing TCP connection on an open port, loading the crafted payload will trigger the alert and inject noisy observation.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 1024:(msg:"RPC sadmind query with root credentials attempt TCP"; flow:to_server established; content:"|00 01 87 88|"; depth:4; offset:16; content:"|00 00 00 01 00 00 00 01|"; within:8; distance:4; byte_jump:4,8,relative,align; content: "|00 00 00 00|"; within:4; metadata: policy security-ips drop; classtype: misc-attack; sid:2255; rev:10;)
```

![Figure 3.1: An example of RPC Sadmind Snort rule](image)

<table>
<thead>
<tr>
<th>HEX Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 89 9c e2</td>
<td>the request id, a random uint32</td>
</tr>
<tr>
<td>00 00 00 00</td>
<td>rpc type (call = 0, response = 1)</td>
</tr>
<tr>
<td>00 00 00 02</td>
<td>rpc version (2)</td>
</tr>
<tr>
<td>00 01 87 88</td>
<td>rpc program (0x00018788 = sadmind)</td>
</tr>
<tr>
<td>00 00 00 0a</td>
<td>rpc program version (0x0000000A = 10)</td>
</tr>
<tr>
<td>00 00 00 01</td>
<td>rpc procedure (0x00000001 = 1)</td>
</tr>
<tr>
<td>00 00 00 01</td>
<td>credential flavor (1 = auth_unix)</td>
</tr>
</tbody>
</table>

As discussed earlier, the basic obfuscation actions are packet level obfuscations. Next, we want to discuss the obfuscation techniques in strategy level, *i.e.*, going beyond manipulating individual events, how to use basic obfuscation actions together to achieve attack
strategy level deception. This work considers three categories of obfuscation strategies: *action alteration*, *action insertion* and *action removal*. These general categories are based upon experiences in working with security experts during the DARPA cyber insider threat project [74]. These categories will be discussed below with examples. The obfuscation examples assume an enterprise network environment. Specifically the network shown in Fig. 3.2 is used, which is the same as that used in Chapter 2 attack prediction. The observations of the network attack are alerts from Snort IDS.

The term *noise attack sequence* is a general term and can be used to describe an observed alert sequence with intentional (attacker’s obfuscation) or unintentional noise (such as IDS sensor failure). This work only considers the case of attacker’s intentional obfuscation. The term *noise* and *obfuscation* will be interchangeably used. The term *clean sequence* will be used to represent the original attack on selected target without using obfuscation techniques.

Figure 3.2: Network used for illustrating obfuscation techniques
Action alteration

For signature-based detection engine, modifying the payload and craft a signature for intended alerts can be easily done. The attacker can alter alerts to hide the true origination and attack characteristic. Further, sometimes, to achieve the same reconnaissance or intrusion objective, many actions can be interchangeable to be played. Changing the order of attacking actions can create equivalent sequence which can make the whole sequence more versatile and avoid being detected by matching to the classical intrusion sequence pattern. Figure 3.2 is an example of action alteration obfuscation. The attacking source is coming from the Internet and the target is an enterprise web sever.

<table>
<thead>
<tr>
<th>Obs. #</th>
<th>Clean alert sequence</th>
<th>Obfuscated alert sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1*</td>
<td>PROTOCOL-ICMP PING Windows</td>
<td>ICMP PING NMAP</td>
</tr>
<tr>
<td>2*</td>
<td>PROTOCOL-ICMP PING Windows</td>
<td>PROTOCOL-ICMP PING Oracle Solaris</td>
</tr>
<tr>
<td>3*</td>
<td>PROTOCOL-ICMP PING Windows</td>
<td>PROTOCOL-ICMP PING LINUX/*BSD</td>
</tr>
<tr>
<td>4*</td>
<td>SCAN nmap fingerprint attempt</td>
<td>SCAN nmap</td>
</tr>
<tr>
<td>5</td>
<td>WEB-IIS unicode directory traversal attempt</td>
<td>WEB-IIS unicode directory traversal attempt</td>
</tr>
<tr>
<td>6</td>
<td>WEB-IIS header field buffer overflow attempt</td>
<td>WEB-IIS header field buffer overflow attempt</td>
</tr>
</tbody>
</table>

The first three alerts in the clean alert sequence are indicative to the attacker’s OS. By using different tools or changing the time-to-live filed for ICMP ping, the attack can generate alerts that look like to be originated from other OS platform, while achieving the same reconnaissance goal. Such obfuscation can be misleading for some alert correlation systems and cause alert correlation failure.

Action insertion

Inserting overwhelming alerts can separate related attack actions to affect the analysis engine, e.g., increasing miss-classification of attack strategy. Even more, overwhelming alerts can cause Denial-of-Services (DoS) on the analysis engine, because the capacity of all alert analysis engine are limited [10, 72].

There are many ways to perform noise injection. For example, one simple way of injecting alerts would be writing a script to keep performing scanning or getting sensitive file
actions from a target to trigger the corresponding detection rules. Although such activity will easily expose attacker’s IP and can be easily blocked by system administrator, using such simple tricks to injecting alerts on compromised host (attackers will not be worried about expose the IP for compromised host) can be effective to dilute the original attack traces. An example of noise injection scenario is shown in Table 3.3. Refering to the network diagram in Fig. 3.2, the attacking source IP are two external IPs. Actions #1, #9, #14 come from one source IP and the others are originated from another IP. The target is on the external web server (subnet 192.168.1.x). Table 3.3 lists 16 action observation window, and obfuscated actions are labeled with asterisk.

<table>
<thead>
<tr>
<th>Obs. #</th>
<th>Clean alert sequence</th>
<th>Obfuscated alert sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SCAN nmap TCP</td>
<td>SCAN nmap TCP</td>
</tr>
<tr>
<td>2*</td>
<td>SHELLCODE Metasploit meterpreter connection attempt</td>
<td>SERVER-WEBAPP /etc/passwd file access attempt</td>
</tr>
<tr>
<td>3*</td>
<td>WEB-IIS header field buffer overflow attempt</td>
<td>SERVER-WEBAPP /etc/passwd file access attempt</td>
</tr>
<tr>
<td>4*</td>
<td></td>
<td>SERVER-WEBAPP /etc/passwd file access attempt</td>
</tr>
<tr>
<td>5*</td>
<td></td>
<td>SERVER-WEBAPP /etc/passwd file access attempt</td>
</tr>
<tr>
<td>6*</td>
<td></td>
<td>SERVER-WEBAPP /etc/passwd file access attempt</td>
</tr>
<tr>
<td>7*</td>
<td></td>
<td>SERVER-WEBAPP /etc/passwd file access attempt</td>
</tr>
<tr>
<td>8*</td>
<td></td>
<td>SERVER-WEBAPP /etc/passwd file access attempt</td>
</tr>
<tr>
<td>9</td>
<td>SHELLCODE Metasploit meterpreter connection attempt</td>
<td>SERVER-WEBAPP /etc/passwd file access attempt</td>
</tr>
<tr>
<td>10*</td>
<td></td>
<td>SERVER-WEBAPP /etc/passwd file access attempt</td>
</tr>
<tr>
<td>11*</td>
<td></td>
<td>SERVER-WEBAPP /etc/passwd file access attempt</td>
</tr>
<tr>
<td>12*</td>
<td></td>
<td>SERVER-WEBAPP /etc/passwd file access attempt</td>
</tr>
<tr>
<td>13*</td>
<td></td>
<td>SERVER-WEBAPP /etc/passwd file access attempt</td>
</tr>
<tr>
<td>14</td>
<td>WEB-IIS header field buffer overflow attempt</td>
<td>WEB-IIS header field buffer overflow attempt</td>
</tr>
<tr>
<td>15*</td>
<td></td>
<td>SERVER-WEBAPP /etc/passwd file access attempt</td>
</tr>
<tr>
<td>16*</td>
<td></td>
<td>SERVER-WEBAPP /etc/passwd file access attempt</td>
</tr>
</tbody>
</table>

Comparing the clean alert sequence and obfuscated sequence, the critical attack actions, #9 and #14 are buried into the sensitive file assess attempt alerts, which can be easily generated with automatic script using HTTP GET. At the same time, the causal relationship between the clean alert sequence, i.e., reconnaissance then intrusion attempts, are also broken by the noise alerts. Note that, such action injection obfuscation may not be effective for all of the alert correlation engines, because for some alert correlation systems, the first step is group alerts by attacking source IP. However, this approach has its drawbacks, grouping by attacking IP could make the analysis for coordinated attack more difficult.
Therefore some other alert correlation will not group by IP and could suffer from this such alert injection obfuscation.

Another example of noise injection is \textit{self-throttling}, where by replaying the actions happened before, the attacker can have higher chance hide the most recent intrusion state, \textit{e.g.}, host discovering, service scanning, privilege escalation, etc.. An example of self-throttling is shown in Table 3.4. The attacking source IP is from Internet, and the target is the external web server. Self-throttling actions are marked with asterisk.

<table>
<thead>
<tr>
<th>Obs. #</th>
<th>Clean alert sequence</th>
<th>Obfuscated alert sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TCP PROTOCOL-ICMP PING Windows</td>
<td>TCP PROTOCOL-ICMP PING Windows</td>
</tr>
<tr>
<td>2*</td>
<td>ICMP PING NMAP</td>
<td>TCP PROTOCOL-ICMP PING Windows</td>
</tr>
<tr>
<td>3</td>
<td>SCAN nmap fingerprint attempt</td>
<td>ICMP PING NMAP</td>
</tr>
<tr>
<td>4*</td>
<td>WEB-IIS unicode directory traversal attempt</td>
<td>TCP PROTOCOL-ICMP PING Windows</td>
</tr>
<tr>
<td>5*</td>
<td>WEB-IIS header field buffer overflow attempt</td>
<td>ICMP PING NMAP</td>
</tr>
<tr>
<td>6</td>
<td>SCAN nmap fingerprint attempt</td>
<td>SCAN nmap fingerprint attempt</td>
</tr>
<tr>
<td>7*</td>
<td>TCP PROTOCOL-ICMP PING Windows</td>
<td>ICMP PING NMAP</td>
</tr>
<tr>
<td>8*</td>
<td>TCP PROTOCOL-ICMP PING Windows</td>
<td>SCAN nmap fingerprint attempt</td>
</tr>
<tr>
<td>9*</td>
<td>Scan nmap fingerprint attempt</td>
<td>SCAN nmap fingerprint attempt</td>
</tr>
<tr>
<td>10*</td>
<td>WEB-IIS unicode directory traversal attempt</td>
<td>WEB-IIS unicode directory traversal attempt</td>
</tr>
<tr>
<td>11</td>
<td>WEB-IIS header field buffer overflow attempt</td>
<td>WEB-IIS header field buffer overflow attempt</td>
</tr>
<tr>
<td>12*</td>
<td>WEB-IIS unicode directory traversal attempt</td>
<td>SCAN nmap fingerprint attempt</td>
</tr>
<tr>
<td>13*</td>
<td>WEB-IIS unicode directory traversal attempt</td>
<td>SCAN nmap fingerprint attempt</td>
</tr>
<tr>
<td>14</td>
<td>WEB-IIS header field buffer overflow attempt</td>
<td>WEB-IIS header field buffer overflow attempt</td>
</tr>
</tbody>
</table>

The clean attack sequence in Table 3.4 can be viewed as three stages, the first stage is performing host discovery and the second stage is performing scanning services and identifying the vulnerability followed by trying to exploit the buffer overflow vulnerability. On the other hand, the obfuscated sequence can be misleading and telling a different story, it is hard to say the attack made any progression, \textit{e.g.}, the target has been discovered or the vulnerability has been identified, but the attacker keep performing low level probing actions.

Another example of noise injection is activity splitting, which is another obfuscation technique to avoid certain patterns that can be triggered by some detection engines. In the packet level, spiting sensitive payload and send the payload though different packet is a widely used technique to perform detection evasion [75]. Similarly, in the alert level, one
can split a malicious signature into multiple steps, and fragment actions can look normal. For example, a long sequence of failed log-in attempts is indicative to a dictionary based password brute-force attack. To be more stealthy, the attacker can split one long sequence into multiple smaller sequences.

An example of such scenario is shown in Table 3.5. In the reference network, the attacking source is from Department A and the target is internal FTP server. Instead of trying 6 passwords one by one, trying 2 passwords for 3 times, separated by PING probing actions. Suppose there is detection rule suggesting three consecutive attempts will generate a brute-force attack alert, which is a naive detection method. The obfuscated sequence will not trigger such alert.

Table 3.5: An example scenario of activity splitting obfuscation

<table>
<thead>
<tr>
<th>Obs. #</th>
<th>Clean alert sequence</th>
<th>Obfuscated alert sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ET SCAN Potential FTP Brute-Force attempt</td>
<td>ET SCAN Potential FTP Brute-Force attempt</td>
</tr>
<tr>
<td>2</td>
<td>ET SCAN Potential FTP Brute-Force attempt</td>
<td>ET SCAN Potential FTP Brute-Force attempt</td>
</tr>
<tr>
<td>3</td>
<td>ET SCAN Potential FTP Brute-Force attempt</td>
<td>TCP PROTOCOL-ICMP PING Windows</td>
</tr>
<tr>
<td>4</td>
<td>ET SCAN Potential FTP Brute-Force attempt</td>
<td>TCP PROTOCOL-ICMP PING Windows</td>
</tr>
<tr>
<td>5</td>
<td>ET SCAN Potential FTP Brute-Force attempt</td>
<td>ET SCAN Potential FTP Brute-Force attempt</td>
</tr>
<tr>
<td>6</td>
<td>ET SCAN Potential FTP Brute-Force attempt</td>
<td>TCP PROTOCOL-ICMP PING Windows</td>
</tr>
<tr>
<td>7</td>
<td>ET SCAN Potential FTP Brute-Force attempt</td>
<td>ET SCAN Potential FTP Brute-Force attempt</td>
</tr>
<tr>
<td>8</td>
<td>ET SCAN Potential FTP Brute-Force attempt</td>
<td>ET SCAN Potential FTP Brute-Force attempt</td>
</tr>
</tbody>
</table>

**Action removal**

Action removal is the obfuscation technique where attacker is hiding critical actions that are indicative of the intrusion state. Table 3.6 is an example containing attack action removal.

The example scenario assumes an experienced insider attacker attacking internal web server from department C in the reference network. After the target was discovered, the attacker uses idle scanning [73] from an internal printer, and completely hide the services.

---

1There are different level of intrusion detections. The detection engine mentioned in this example is NIDS, and the long sequence of brute force password attack can be interrupted by other actions, such as PING or scanning. On the other hand, it is possible that there are other intrusion detection systems installed on application level. For example, by inspecting FTP logs, the activity splitting obfuscation will not be effective anymore, because FTP application will not see and be interrupted by PING actions.
Table 3.6: An example scenario of action removal obfuscation

<table>
<thead>
<tr>
<th>Obs. #</th>
<th>Clean alert sequence</th>
<th>Obfuscated alert sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PROTOCOL-ICMP PING Windows</td>
<td>PROTOCOL-ICMP PING Windows</td>
</tr>
<tr>
<td>2*</td>
<td>SCAN nmap XMAS</td>
<td>SHELLCODE Metasploit meterpreter connection attempt</td>
</tr>
<tr>
<td>3*</td>
<td>SCAN nmap TCP</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SHELLCODE Metasploit meterpreter connection attempt</td>
<td></td>
</tr>
<tr>
<td>5*</td>
<td>INDICATOR-COMPROMISE c99shell.php command request - ls</td>
<td></td>
</tr>
<tr>
<td>6*</td>
<td>INDICATOR-COMPROMISE c99shell.php command request - ps aux</td>
<td></td>
</tr>
</tbody>
</table>

scanning actions. The last two actions from the clean alert sequence create backdoor access to the running processes in the server, and critical to suggest that the target is already compromised. By carefully choosing encoding or encryption schemes, such alerts can be hidden from being observed and cause the inaccurate assessment.

An example of obfuscated network attack sequence

To summarize the various obfuscation techniques, we define the network attack as an intrusion process that includes malicious probing for information gathering and attack surface exploitation for possible vulnerabilities. An observed network attack is a sequence of events with uncertainties, including possible missing and misleading observations.

Table 3.7 gives an example of attack sequence and one possible way of obfuscation. Snort descriptions are used to illustrate the attack progression, in terms of the exploits attempted by the attacker. The observations with an asterisk represent the obfuscated actions.

The attacker began with host discovery and altered the ICMP packet time-to-live field to hide the signature of the operating system used for reconnaissance. Once the attacker discovered the Web server, various encoding techniques were used to hide the malicious signature in the packet payload. Such actions triggered the obfuscation indicator alerts, notifying the analyst that possible obfuscations were detected, but with uncertainty about the specific actions evaded / obfuscated. Finally, after the attacker successfully compromised the target with FTP vulnerabilities. More stealthy actions are used to avoid the exposure of

---

2Idle scanning takes advantage of predictable TCP sequence number vulnerability and can be completely anonymous when probing target host’s servers.
compromise indicator alerts.

Table 3.7: An example of attack sequence obfuscation

<table>
<thead>
<tr>
<th>Obs. #</th>
<th>Clean alert sequence</th>
<th>Obfuscated alert sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PROTOCOL-ICMP PING Windows</td>
<td>PROTOCOL-ICMP PING Windows</td>
</tr>
<tr>
<td>2*</td>
<td>PROTOCOL-ICMP PING Windows</td>
<td>PROTOCOL-ICMP PING Oracle Solaris</td>
</tr>
<tr>
<td>3*</td>
<td>PROTOCOL-ICMP PING Windows</td>
<td>PROTOCOL-ICMP PING LINUX/*BSD</td>
</tr>
<tr>
<td>4</td>
<td>PROTOCOL-ICMP PING undefined code</td>
<td>PROTOCOL-ICMP PING undefined code</td>
</tr>
<tr>
<td>5</td>
<td>ICMP PING NMAP</td>
<td>ICMP PING NMAP</td>
</tr>
<tr>
<td>6*</td>
<td>SCAN nmap XMAS</td>
<td>SCAN nmap TCP</td>
</tr>
<tr>
<td>7</td>
<td>SCAN nmap TCP</td>
<td>SCAN nmap TCP</td>
</tr>
<tr>
<td>8</td>
<td>SCAN nmap fingerprint attempt</td>
<td>SCAN nmap fingerprint attempt</td>
</tr>
<tr>
<td>9</td>
<td>SHELLCODE Metasploit meterpreter connection attempt</td>
<td>SHELLCODE Metasploit meterpreter connection attempt</td>
</tr>
<tr>
<td>10*</td>
<td>WEB-IIS header field buffer overflow attempt</td>
<td>WEB-IIS header field buffer overflow attempt</td>
</tr>
<tr>
<td>11*</td>
<td>WEB-IIS multiple decode attempt</td>
<td>WEB-IIS multiple decode attempt</td>
</tr>
<tr>
<td>12*</td>
<td>WEB-IIS unicode directory traversal attempt</td>
<td>WEB-IIS unicode directory traversal attempt</td>
</tr>
<tr>
<td>13*</td>
<td>FTP command overflow attempt</td>
<td>FTP command overflow attempt</td>
</tr>
<tr>
<td>14*</td>
<td>FTP EXPLOIT overflow</td>
<td>FTP EXPLOIT overflow</td>
</tr>
<tr>
<td>15*</td>
<td>INDICATOR-COMPROMISE c99shell.php command request - ls</td>
<td>INDICATOR-COMPROMISE c99shell.php command request - ls</td>
</tr>
<tr>
<td>16*</td>
<td>INDICATOR-COMPROMISE c99shell.php command request - ps aux</td>
<td>INDICATOR-COMPROMISE c99shell.php command request - ps aux</td>
</tr>
</tbody>
</table>

The obfuscation example described in Table 3.7 can have a profound impact on network attack situation assessment. In particular, by looking at the clean alert sequence, one can clearly see the intrusion stages, i.e., from reconnaissance, to exploit the vulnerability on Web server, FTP server and eventually compromised the target host. On the other hand, obfuscated alert sequence can lead to a different conclusion, because several alerts are missing and others are replaced with obfuscation indicator. By looking at the obfuscated alert sequence, one may conclude it is more likely to be an ad-hoc probing and vulnerability attempts, e.g., using FastTrack automated attack script [76].

For most alert analysis engines, the key component of analyzing attacks is understanding the casual relationships between attack actions [19]. For example, if an attack exploits a Windows vulnerability, the prior reconnaissance actions are likely to be some kind of OS footprinting, assuming no missing observations. The diverse attack behaviors and obfuscations present uncertainties, making the attack process probabilistic instead of deterministic. Such uncertainties can mislead the alert analysis engines, similar to the examples shown
above.

### 3.1.2 Current attack obfuscation countermeasures

The above provides a review of obfuscation techniques could be used to confuse the alert analysis system. Because of the variation of the system services, e.g., network configurations and NIDS engine, the obfuscation techniques can be effective to certain types of systems but not other.

Facing the different types of obfuscation techniques, many IDSs and alert analysis engines have developed countermeasures. For example, for detecting code-level obfuscations, Snort NIDS employs decoders and pre-processors to decode the payload of the packet before pattern matching was applied. For packet-level obfuscation, there are enhanced rules to detect the malicious manipulation of the packets. For example, in the early years, XMAS scan [73] was considered to be a stealthy scan since it constructs an invalid packet and the OS will usually drop it without logging. In modern NIDSs, these scanning are not stealthy anymore and such signatures are included into the database.

The battle between attack evasion and attack detection has gone on for many years and the techniques have grown on both sides. Unfortunately, we want to emphasize that attack defense is lagging behind because the detection rule enhancements are generally ad-hoc fixes to specific obfuscation techniques after they are discovered. In this continuous battle, one missing component is to analyze the effect of obfuscation without knowing the exact obfuscation technique, by developing an abstraction of the obfuscation techniques: action alteration, insertion and removal. The next section will present an abstraction of attack obfuscations and show the benefit of recovering the attack strategy/model of the attacker, to allow the deployment of preemptive defense mechanisms.

### 3.2 Probabilistic modeling on attack obfuscations

We consider an attack strategy/model as a probabilistic sequence model, e.g., Markov model or Hidden Markov Model (HMM) [77], to describe the different possible attack
actions and captures the casual relationship of attack actions using transition probabilities. More specifically, an attack sequence is mathematically described as a **vector of random variables** and each observation is an instance/sample of the attack model. When obfuscated, the attack sequence is modeled by another vector of random variables, where an *obfuscation model* represents the obfuscation techniques probabilistically. The joint distribution is the overall description for the attack sequence that contains possible obfuscated observations. Because the clean attack sequence and the obfuscated sequence are not independent, one needs to jointly treat the attack model and the obfuscation model for probability inference.

Let \( \Omega \in \{0, 1, 2, 3, \ldots \} \) represents the set of possible attack actions, the attack sequence is defined as a length-\( N \) vector random variable \( X \), where random variable \( X_k \in \Omega, k \in \{1, 2, 3, \ldots, N\} \) is defined as the \( k^{th} \) observed action in the attack sequence \( X \). An attack model is a probabilistic sequence model to specify \( P(X) \), which is shown in (3.1) as a \( L^{th} \) order Markov model.

\[
P(X) = P(X_1, \cdots, X_L) \prod_{k=1}^{N-L} P(X_{L+k} | X_{L+k-1}, X_{L+k-2}, \cdots, X_k)
\]

(3.1)

where \( P(X_1, \cdots, X_L) \) represents the initial distribution of the \( L^{th} \) order Markov model. The Markov property (given \( L \) observations past and further are independent) enables the product form decomposition of \( P(X) \).

The attack model discussed here does not take the obfuscated observations into account and it represents the intended attack strategy of the attacker. The term *clean attack sequence* is used to represent the sequences directly generated from this attack model. Because of the Markov property, the attack model has a chain structure [78]. Figure 3.3 shows the graphical representation of a first order attack model, and the joint distribution of \( X \) can be written in (3.2).

\[
P(X) = P(X_1) \prod_{i=1}^{N-1} P(X_{i+1} | X_i)
\]

(3.2)
Let a random variable vector $Y$ represents obfuscated attack sequence. $Y$ conditional depends on $X$ and the obfuscation model $P(Y|X)$ describes the relationship between clean and obfuscated sequence probabilistically.

According to the sequence length relationship between clean sequence $X$ and obfuscated sequence $Y$, there are two types of models:

- Type-I model, where the length of $X$ is equal to length of $Y$.
- Type-II model, where the length of $X$ is larger than or smaller than length of $Y$.

Type-I model can be used to model attack action alteration obfuscation and Type-II model can be used for noise insertion and action removal.

### 3.2.1 Type-I model for action alteration

#### Action alteration model

For Type-I model, *i.e.*, the clean and noise sequence have same length, one important model is HMM. HMM is a widely used model to describe sequential observation with noise, where the hidden random variables represent the clean sequence, and the observations represent the noise sequence. Figure 3.4 gives the graphical notation of classical HMM for a length-$N$ sequence.

$$P(Y|X) = \prod_{k=1}^{N} P(Y_k|X_k)$$  \hspace{1cm} (3.3)

In HMM, observed event $i$ only directly depends on the corresponding hidden state $i$, $P(Y|X)$ for HMM can be written as (3.3). The term $P(Y_k = y|X_k = x)$ is called
emission probability, which can be described by a discrete function, $g(x, y)$. We define $g(x, y)$ as obfuscation function/matrix.

Additional constrains can and should be placed on the basic framework of HMM to reflect attack obfuscation behavior. Specifically, the classical HMM obfuscation model (3.3) assumes attackers perform obfuscations independently across all attack actions, and the obfuscation of $P(Y_k)$ will directly depend on $P(X_k)$ but have indirect correlations with other obfuscated actions. In real-world attack, it almost never happens, and there are strong correlations between obfuscated actions. Such correlation can be reflected by a more general obfuscation model $P(Y|X)$ by placing explicit constraints on $Y$. For example, a simple extension of (3.3) can be shown in (3.4).

$$P(Y|X) = \frac{1}{\binom{N}{M}} \prod_{k=1}^{N} P(Y_k|X_k) \quad (3.4)$$

where $I(\cdot)$ is the indicator function, $|\cdot|_H$ represents the Hamming distance between two vectors.

The added parameter $M$ can be interpreted as an estimate of the percent ($M$ out of $N$) of attack actions the attacker may change actions. It serves to complement $g(x, y)$, which describes the preference on which obfuscation action is more likely to be chosen. The noise model described in 3.4 is published in [79] and [80].

On the other hand, we also extend the basic HMM structure to finite order on $X$ instead.

---

$^3$The concept of Markov Blanket [78] gives all of the random variables $Y_k$ depends on. Here we mean the connections in graphical notation are only from $Y_k$ to $X_k$. 

---

Figure 3.4: Graphical model notation for HMM
of first order. This extension allows us to use a higher order attack model. Recall our attack sequence modeling work [51] [63] [64] discussed in Chapter 2, cross validation shows the order is larger than one.

The resulting overall joint attack and obfuscation model can be depicted as a graphical model shown in Fig. 3.5 with the dashed nodes representing that $Y_k$ are not only dependent on $X_k$ but can also allow only $M$ changes. This simple extension allows assessing the impact of obfuscation as a function of the percentage of attack actions altered, which will be shown in Chapter 5. In addition, we will also show the inaccurate estimation of parameter $M$ have little effect on the attack classification performance.

![Figure 3.5: An example of proposed Type-I model (second order)](image)

**More discussions about action alteration model and HMM**

This subsection will discuss more about the rational of the action alteration model proposed in (3.4). We will focus on two topics: 1. the intuition behind the indicator function design and 2. the relationship between the classical model, such as HMM.

One simple example will be used for the obfuscation model discussion. Consider there are 5 possible attack actions: $\Omega = \{A, B, C, D, E\}$, for any action, there are equal chance to alter it into other actions. For HMM, the obfuscation matrix $P(Y_k|X_k)$ is shown in Table 3.8, where $a$ is a float number and $0 \leq a \leq 1$. Note that, the parameter $a$ controls *how much obfuscation exists*. For example, if $a = 0.8$, it means 80% of the time, the attacker will keep the original attack action, and 20% of the time will alter the original attack action to other actions.

On the other hand, using the proposed action alteration model, the obfuscation matrix
Table 3.8: An example of $P(Y_k|X_k)$ for HMM

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
</tbody>
</table>

$P(Y_k|X_k)$ will be represented in Table 3.9, the key difference is the diagonal elements of the matrix are always 0 and the parameter $M$ in (3.4) will be used to control how much obfuscation exists. If the diagonal elements of the matrix contain none-zero values, the conditional probability $P(Y|X)$ is not a valid. (probability definition ($P(Y|X)$ is valid, if and only if, for any given $X$, $\sum_Y P(Y|X) = 1.0$).

Table 3.9: An example of $P(Y_k|X_k)$ for proposed action alteration model

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>0.00</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>0.25</td>
<td>0.00</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>0.25</td>
<td>0.25</td>
<td>0.00</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.00</td>
<td>0.25</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Comparing between two models, the key difference is modeling action alteration implicitly vs. explicitly. HMM model treats no alteration as one type of the obfuscation (the diagonal elements of the matrix can be none-zero). HMM controls number of alterations in the whole attack sequence by the obfuscation matrix implicitly. For example, for a length-$N$ sequence, $P(Y_k|X_k)$ given in Table 3.8 will implicitly suggests $(1 - a)N$ number of alterations. And note that, $(1 - a)N$ is only an expected value, and in reality, number of alterations is a random number.

On the other hand, for the alteration model proposed in (3.4), we model the action alteration explicitly and number of alterations can be set into a precise number $M$ (Section
5 will show the inaccurate estimation of $M$ will not cause big impact for the analysis). The reason we have such design is that it is necessary to separate attack model with noise model in security context. The action alteration model proposed also means no alteration is not a obfuscation activity and should NOT be captured in $P(Y_k|X_k)$. Instead, such behavior should be captured by $P(X)$. 

The proposed design is more realistic for security applications, where it may possible to learn classical clean attack patterns and classical obfuscation techniques from data, separately. But it is more difficult to learn both of attack model and obfuscation model at same time. Further, suppose the network analysts want to use domain knowledge to derive the obfuscation model, it is relative easy to ask the analysts how Table 3.9 should be constructed and how much noise actions approximately exists. However, on the other hand, it is more difficult to construct Table 3.8 to reflect obfuscation level implicitly.

### 3.2.2 Type-II model for action insertion and action removal

For Type-II model, $X$ and $Y$ have different length. For action insertion, the length of $Y$ is larger than $X$ and for action removal action insertion, the length of $Y$ is smaller than $X$. Similar to Type-I model, there should be a parameter suggestion how much obfuscation exists. Because the Type-II model have different length on $X$ and $Y$, how much obfuscation exists will be directly reflected in model structure. In our work, we use regularized structure as an example to show the inference design. The model structure can be easily changed and we have more discussions the end of this subsection.

**Action insertion**

Figure 3.6 is an example of action insertion model, where we assume for every one clean action, additional obfuscated action can be injected. And the length of $Y$ is twice times of the length of $X$. For given noisy observation, it conditionally depends on previous action and one clean attack action.

For the model proposed in Fig. 3.6, the join distribution is shown in (3.5) and the
Figure 3.6: Graphical representation for action insertion

The obfuscation model $P(Y|X)$ can be described in (3.6).

\[
P(X, Y) = \left( P(X_1) \prod_{i=1}^{N/2-1} P(X_{i+1}|X_i) \right) \\
\left( P(Y_1|X_1)P(Y_2|Y_1, X_1) \prod_{i=2}^{N/2} P(Y_{2i-1}|Y_{2i-2}, X_i)P(Y_{2i}|Y_{2i-1}, X_i) \right) \\
= P(X)P(Y|X) \quad (3.5)
\]

\[
P(Y|X) = P(Y_1|X_1)P(Y_2|Y_1, X_1) \prod_{i=2}^{N/2} P(Y_{2i-1}|Y_{2i-2}, X_i)P(Y_{2i}|Y_{2i-1}, X_i) \quad (3.6)
\]

**Action removal**

Similarly, for action removal case, if we assume one of two actions can be removed, the model structure is shown in Fig. 3.7. The joint distribution and obfuscation model can be described in (3.7) and (3.8).

\[
P(X, Y) = \left( P(X_1) \prod_{i=1}^{N-1} P(X_{i+1}|X_i) \right) \left( P(Y_1|X_1, X_2) \prod_{i=2}^{N/2} P(Y_{2i}|Y_{2i-1}, X_{2i-1}, X_{2i}) \right) \\
= P(X)P(Y|X) \quad (3.7)
\]
Discussions on model structure

We want to argue that the model structure proposed in our work are general examples to model the attack with obfuscations, but not advocating the proposed models are only the models for specific cases. Security analysts can make changes to the model structure to reflect more customized obfuscation scenarios.

For example, for Type-II model, the action insertion case, one may argue that the injected noise is conditionally independent of any variable, i.e., the attack randomly injects noise and there are no links between the injected noise and the true observations, as shown in Fig. 3.8.

\[
P(Y|X) = P(Y_1|X_1, X_2) \prod_{i=2}^{N/2} P(Y_i|Y_{i-1}, X_{2i-1}, X_{2i})
\] (3.8)

In addition, another example of modifying the structure is adding links to the random variable. For Type-I model, the attack action alteration case, one may argue that the clean
sequence and noise sequence model are auto-regressive HMM [81] as shown in Fig. 3.9.
In such a case, noise action $Y_i$ depends on the previous noise action $Y_{i-1}$.

![Figure 3.9: An example of adding dependencies on Type-I model](image)

Finally, for Type-II model (shown in Fig. 3.10), it is possible to be extended with a higher order on $X$ and additional constrains on $Y$, similar to the Type-I model shown in Fig. 3.5, where clean sequence has a second order model and $Y$ is constrained in certain subset of all possible actions.

![Figure 3.10: An example of adding dependencies and constrains on Type-II model](image)

The model structure can be extended to incorporate aforementioned features from our proposed framework. For example, removing dependencies can be achieved by setting a special parameter of the the general model. Consider a toy example shown in Fig. 3.11. In such structure, assume all random variables are binary, $P(Z|X,Y)$ is a $2 \times 2 \times 2$ table. We can use a two dimensional table shown in Table 3.10 to represent such a three dimensional table, where $0 \geq a, b, c, d \geq 1$.

Now suppose we want to set that $Z$ only depends on one random variable. If $Z$ only
Figure 3.11: An toy example of dependency link removal

Table 3.10: An example of parameter set up for link removal toy example

<table>
<thead>
<tr>
<th>Row</th>
<th>Column 1</th>
<th>Column 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$P(Z = 0</td>
<td>X = 0, Y = 0) = a$</td>
</tr>
<tr>
<td>2</td>
<td>$P(Z = 0</td>
<td>X = 0, Y = 1) = b$</td>
</tr>
<tr>
<td>3</td>
<td>$P(Z = 0</td>
<td>X = 1, Y = 0) = c$</td>
</tr>
<tr>
<td>4</td>
<td>$P(Z = 0</td>
<td>X = 1, Y = 1) = d$</td>
</tr>
</tbody>
</table>

depends on $X$, we can set rows 1 and 2, row 3 and 4 are exactly the same, respectively, i.e., $a = b, c = d$; If we want $Z$ only depends on $Y$, setting $a = c, b = d$ will satisfy the needs. This example shows how the conditional distribution table can be changed to reflect the link removal in the model.

Adding dependencies and constrains are more complicated than removing dependencies. Nevertheless, Chapter 6 will discuss how proposed design can perform probabilistic inference for most extensions efficiently.

### 3.3 Probabilistic inference and impact assessment for obfuscated sequences

Different types of models proposed in Section 3.2 enable us to make the impact assessment for different obfuscation techniques. One of the most important tasks for security analysis is attack plan recognition [19], which is correlating the observed sequence to one of the pre-defined attack models. Such correlation would help security analysts to understand the attack and eventually to predict next possible actions. Correlating observed attack action
sequence to attack models is a sequence classification problem\(^4\) [82] [83]. We first discuss about matching a clean attack sequence \(X\) to pre-defined models. The problem can be described as finding most possible \(C\) for given \(X\), which is \(\arg\max_C P(C|X)\). The attack models which specify \(P(X|C)\), and the prior of the attack model \(P(C)\) are given. Using the Bayes theorem, the classification can be described in (3.9) which is the classical Bayesian classification: calculating posterior from prior and likelihood.

\[
\arg\max_C P(C|X) = \arg\max_C \frac{P(X|C)P(C)}{P(X)} = \arg\max_C P(X|C)P(C)
\]

(3.9)

In order to perform such classification, \(P(X|C)\) and \(P(C)\) need to be known. \(P(C)\) will be given from domain knowledge. On the other hand, \(P(X|C)\) can be calculated from attack model. Knowing \(P(X|C)\) and \(P(C)\) enables us to calculate the conditional probability \(P(C|X)\) and eventually classify the observed sequence in to predefined attack model using Bayesian classification as shown in (3.9).

So far, the attack classification discussed assumes the observed sequences \(X\) are generated by pre-defined attack models and do not contain any noise. To evaluate the impact caused by attack obfuscations, it is necessary to understand the performance limit of the classification when attacks contain obfuscation. Therefore, we need to finding the representative metric for impact assessment with possible obfuscated observations.

The metric we used is expected classification accuracy, which have close relationship with many concepts in statistics literature, such as irreducible error or Bayes error rate [37]. This metric means that, assume the true distribution are known and we have the perfect model to captures the true distribution of the data, there may still be errors when making classification, because the true distribution of the different classes may overlap.

\(^4\)In general, there are different types of the correlation engine, the correlation discussed in following sections are computational approaches and rule based correlation are not included.
The term *irreducible error* or *Bayes error rate* will describe how much the overlap is, and the metric we used is $1 - \text{Bayes error}$.

An example is given here to introduce the performance metric on clean sequence and noise sequence. Figure 3.12 gives an example of sequence classification and performance limit of the classification. Using the two attack model specified in Chapter 5, 200 length-20 attack sequences are generated for each attack model. The prior distribution $P(C)$ is uniform on two attach models. For each sequence, the Log-likelihood for each of the two models are calculated. All the sequences are plotted as a data point in Fig. 3.12, where x-axis and y-axis are the likelihood for model 0 and model 1 respectively. Because of the generative model is known and the prior is equal for two classes, (3.9) tells us the optimal classification is comparing two likelihoods which is shown as the optimal classifier is the diagonal line.

![Figure 3.12: An example of attack sequence classification by comparing the likelihood](image)

As shown in Fig. 3.12, it is easy to classify the given data points (classification can be done by comparing the likelihood), but it is not clear to state how *separable* the data is, *i.e.*, what is the limit that the classifier can do. The metric of *expected classification accuracy*
for an obfuscated sequence \( Y \) is defined as

\[
\sum_Y P(Y) \max_C P(C|Y)
\]  

Equation (3.10) can be explained as follows: for any given obfuscated observation \( Y \), \( P(C|Y) \) can be calculated for all attack models, and the noise sequence \( Y \) can be classified into pre-defined models by using \( \arg \max_C P(C|Y) \). By doing such classification, \( \max_C P(C|Y) \) percent of all time, the classification will be correct. Summing over all possible \( Y \) will give us the mathematical expectation of the classification accuracy.

In sum, the problem addressed in our framework can be described as how to calculate the performance limit for obfuscated attack observations for different obfuscation strategy \( P(X|Y) \), \textit{i.e.}, calculating \( \sum_Y P(Y) \max_C P(C|Y) \), given \( P(X|C) \), \( P(C) \), and \( P(Y|X) \).

As shown in Chapter 4, the calculation, \textit{i.e.}, probabilistic inference, is computational challenging, especially for certain obfuscation strategies, algorithms need to be derived to inference effectively.

Using proposed framework and algorithm, the network analysts would be able to calculate the noise sequence distribution. Figure 3.13 is one example of our published simulation results [79], \( P(Y) \) is shown in Fig. 3.13(b), form the figure we can roughly observe how much impact caused by given obfuscation strategy: only 10% of obfuscated events cause a lot of overlap for two group of points. More importantly, the performance limit shown in (3.10) can be calculated as a benchmark to access specific scenarios.

Table 3.11 summarizes the notations used in our framework and will be heavily used in next chapter’s algorithm design.
Figure 3.13: Comparison of classify attack sequence with/without noise

Table 3.11: Notations used in algorithm design

<table>
<thead>
<tr>
<th>Notation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Omega$</td>
<td>a set of possible attack actions</td>
</tr>
<tr>
<td>$X$</td>
<td>a vector of random variable represents observed actions</td>
</tr>
<tr>
<td>$X_k$</td>
<td>$k^{th}$ action in vector $X$</td>
</tr>
<tr>
<td>$C$</td>
<td>a random variable represents attack model classes</td>
</tr>
<tr>
<td>$f^C(\cdot)$</td>
<td>a function describes for attack model $C$</td>
</tr>
<tr>
<td>$N$</td>
<td>a integer, represents the length of the observation</td>
</tr>
<tr>
<td>$Y$</td>
<td>a vector of r.v. represents noisy observation</td>
</tr>
<tr>
<td>$g(x, y)$</td>
<td>attack altering probability for noise model</td>
</tr>
</tbody>
</table>
Chapter 4

Inference algorithm design

This chapter will discuss the algorithm design for performing probabilistic inference on proposed Type-I and Type-II models. We first introduce the concept of probabilistic inference and computational challenges, then present basic ideas of inference for probabilistic graphical model by two simplified problems. Utilizing the ideas illustrated in simplified problems, theorems and algorithms are derived to inference on the proposed models that reflect attack sequence obfuscations. Finally, the algorithm of evaluating the impact of attack obfuscation is discussed.

According to the problem statement in the last part of Chapter 3 (subsection 3.3), the metric we used to evaluate the expected classification accuracy for obfuscated attack sequence $Y$ is shown in (4.1).

$$\sum_{Y} P(Y) \max_{C} P(C|Y)$$

(4.1)

Distribute $P(Y)$ into max operation, Equation (4.1) is can be written as

$$\sum_{Y} \max_{C} P(C|Y)P(Y)$$

$$= \sum_{Y} \max_{C} P(C, Y)$$

$$= \sum_{Y} \max_{C} P(Y|C)P(C)$$

(4.2)

The probability of clean observation sequence for given attack model $P(X|C)$, the
prior of attack models $P(C)$ and the noise model $P(Y|X)$ \(^1\) are assumed to be known. To calculate $\sum_Y \max_C P(Y|C) P(C)$, two sub-problems need to be solved:

- Given $P(X|C)$ and $P(Y|X)$, calculate $P(Y|C)$.
- Given $P(Y|C)$, calculate $\sum_Y \max_C P(Y|C) P(C)$.

The first subproblem is can be solved by an extension of Message Passing Algorithm [78] using dynamic programming idea, and the second subproblem can be approximated by Monte-Carlo approximation with any desired precision and confidence.

### 4.1 Probabilistic inference algorithm design

#### 4.1.1 Basic probabilistic inference problem

Probabilistic inference is the problem of calculating specific marginal or conditional distributions for given model [78]. The inference is computationally challenging because brute-force calculation needs to take account for the exponential number of terms. For example, let $X$ to be a length-$N$ vector of random variables. Calculate the marginal distribution $P(X_1)$ needs to sum over $X_2, X_3, \ldots, X_N$, which is looping over $\Omega^{N-1}$ terms, where $X_i \in \Omega$.

Performing exact inference for arbitrary model structure of $P(X)$ is a NP-hard problem [81]. However, for some special structure of $P(X)$ (for example we will discuss the chain structure later), efficient algorithm exists.

In HMM literature, there are classical algorithms to perform exact inference. For example, the well-known Viterbi algorithm [77] enables one to efficiently calculate the most probable path of clean sequence, \textit{i.e.}, solving (4.3).

$$\arg \max_X P(X|Y) \quad (4.3)$$

\(^1\)For different types of attack obfuscations, \textit{e.g.}, Type-I and Type-II model, $P(Y|X)$ is different from one to another.
Likewise, one can efficiently calculate $P(Y)$ for HMM, because $P(Y|X)$ is relatively simple for HMM, comparing to the obfuscation model proposed in Chapter 3.

Unfortunately, some of the existing algorithms cannot be directly applied to the Type-I and Type-II model proposed in Chapter 3. For Type-I model, constraint $M$ on $Y$ will affect the possible values of $X$ and the $\arg\max$ operation will only apply to a subset of $X$ that satisfies the constraint. For Type-II model, the length of $X$ and $Y$ are different.

Here we discuss two simplified problems to show the idea of solving the inference problem in our proposed model. After discussing the simplified problem, the algorithm design for different model structure and possible extensions are given.

**Inference on a chain structure**

One example of inference on a chain structure is explained as follows: suppose we want to solve the optimization problem\(^2\) shown in (4.4).

$$\max_X \prod_{k=1}^{N-1} f(X_k, X_{k+1}) \quad (4.4)$$

We can represent the relationship for all the variables with a chain structure shown in Fig. 4.1, where $X_i$ only interact with $X_{i-1}$ and $X_{i+1}$. This simplified problem comes from the first order Markov model described in 2, because the joint distribution of first order Markov model have the term $\prod_{i=1}^{N-1} P(X_{i+1}|X_i)$ and according to Markovian property given $X_i$, $X_{i-1}$ and $X_{i+1}$ are independent.

![Figure 4.1: An example of chain structure](image)

Because the objective function has such a special chain structure, we can take advantage of this structure, and use dynamic programming techniques to solve the optimization.

---

\(^2\)In the beginning of this Chapter, inference example is given by summation operation, and in this example it is given by maximization operation. In fact, they are eventually the same from algorithm design perspective as discussed in Murphy’s dissertation [81].
Define a function

\[ F_i(a) = \max_{X_1 \ldots X_i} \prod_{k=1}^{i-1} f(X_k, X_{k+1}) \]  

s.t. \( X_i = a \)

The function \( F_i(a) \) defines \( F_i(a) \) as the cost of the best length-\( i \) subsequence and end with symbol \( a \). For the chain structure, the only connections between subsequence \( X_1 \) to \( X_i \) and subsequence \( X_i \) to \( X_N \) is the variable \( X_i \). The reason we set the constrain of the subsequence ends with symbol \( a \) is because we want to decouple the interactions between two subsequences. Such constrain will allow us to solve the problem in a smaller scale, and drive recursion rules for extension, which is the idea of dynamic programming. By define the function \( F_i(a) \) we can prove the recursion rule described Theorem 4.1.1.

**Theorem 4.1.1.** Let \( F_i(a) \) be the cost of the best length-\( i \) subsequence and end with symbol \( a \) as shown in (4.5). Equation (4.6) gives the relationship between \( F_i(a) \) and \( F_{i-1}(a) \), that can be used in Algorithm 1 to find the optimal solution for (4.4).

\[ F_i(a) = \max_b F_{i-1}(b) \cdot f(b, a) \]  

\(^3F_i(a)\) is a discrete function of \( a \) for give \( i \). Because it is a discrete function, it can be stored in one dimensional a table / array. In addition, \( F_i(a) \) can also be viewed as a two dimensional table for different \( i \). These tables are also called dynamic programming table. In this work, the term dynamic programming function and dynamic programming table interchangeably.
Proof.

\[
\begin{align*}
\max_b F_{i-1}(b) \cdot f(b, a) \\
= \max_b \left( \max_{X_1, \ldots, X_{i-1}} \prod_{k=1}^{i-2} f(X_k, X_{k+1}) \right) \cdot f(b, a) \\
\text{s.t. } X_{i-1} = b \\
= \max_{X_1, \ldots, X_{i-1}} \prod_{k=1}^{i-2} f(X_k, X_{k+1}) \cdot f(X_{i-1}, a) \\
= \max_{X_1, \ldots, X_{i-1}} \prod_{k=1}^{i-1} f(X_k, X_{k+1}) \\
\text{s.t. } X_i = a \\
= F_i(a)
\end{align*}
\]

Using Theorem 4.1.1, the algorithm to solve the optimization is given in Algorithm 1. The complexity is \( \Theta(N \cdot |\Omega|^2) \), where \( N \) is the length of the sequence, and \( |\Omega| \) is the number of the possible values of the random variables, \( i.e., X_i \in \Omega \). As discussed earlier, the brute-force calculation for searching the max value will consider \( |\Omega|^N \) number of terms.

**Inference with constrains**

As discussed earlier, Type-I model has the additional constrain on \( Y \). In the second example, we add constraint to the chain structure to illustrate the algorithm design.

The revised problem is shown in (4.7). The major difference is that only a subset of \( X \) needs to be considered. The subset of \( X \) is depend on \( M \) and \( Y \), \( i.e., \) only \( M \) number of elements are allowed be different from a given vector \( Y \).

\[
\sum_{X:|X-Y|_{H}=M} \prod_{i=1}^{N-1} f(X_{i+1}, X_i) \tag{4.7}
\]

We represent the chain structure with constrain in Fig. 4.2, the dashed variables represent additional constraints are applied.
**Algorithm 1:** Algorithm to solve the inference on a chain structure

**Input:** Given the sequence length $N$, function $f(x, y)$, $X_i \in \Omega$

**Output:** $\max X \prod_{k=1}^{N-1} f(X_k, X_{k+1})$

// Initialization Step

for $a \in \Omega$ do
  Initialize $F_2(a) = \max_{X_1} f(X_1, a)$
end

// Run dynamic programming according to recursion rule

for $i \in 3, 4, \cdots, N$ do
  for $a \in \Omega$ do
    $F_i(a) = \max_{b} F_{i-1}(b) \cdot f(b, a)$
  end
end

return $\max_{a} (F_N(a))$

---

![Chain Structure](image)

**Figure 4.2:** An example of chain structure with constrain

Comparing to the solution without additional constraint, one can add another dimension to the dynamic programming table to **decompose the dependencies on constrain**. We define the dynamic programming function in (4.8).

$$F_{i,j}(a) = \sum_{X_1, \cdots, X_i} \prod_{k=1}^{i-1} f(X_k, X_{k+1})$$  \hspace{1cm} (4.8)

s.t. \hspace{1cm} $X_i = a$

$$| <X_1, \cdots, X_i> - <Y_1, \cdots, Y_i> |_H = j$$

**Theorem 4.1.2** gives the recursion rules on defined function in (4.8).

**Theorem 4.1.2.** Let $F_{i,j}(a)$ be the sum for (4.7) and sum over the subsequence $X_1, \cdots, X_i$. In addition, the subsequence $X_1, \cdots, X_i$ is different from $Y_1, \cdots, Y_i$ by $j$. Further, $X_i = a$ as defined in (4.8). The relationship between $F_{i,j}(a)$, $F_{i-1,j}(a)$ and $F_{i-1,j-1}(a)$ can be described with (4.9).
\[ F_{i,j}(a) = \begin{cases} 
\sum_b F_{i-1,j}(b) \cdot f(b,a) & \text{if } a = Y_i \\
\sum_b F_{i-1,j-1}(b) \cdot f(b,a) & \text{if } a \neq Y_i 
\end{cases} \tag{4.9} \]

The proof of Theorem 4.1.2 case \( a = Y_i \) is given. Using similar idea, it is easy to show the recursion rules for \( a \neq Y_i \) case.

**Proof.** Case 1, \( a = Y_i \)

\[
\sum_b F_{i-1,j}(b) \cdot f(b,a) = \sum_b \left( \sum_{X_1 \cdots X_{i-1}} \prod_{k=1}^{i-2} f(X_k, X_{k+1}) \right) \cdot f(b,a)
\]

s.t. \( X_{i-1} = b \)

\[ |(X_1, X_2, \cdots, X_{i-1}) - (Y_1, Y_2, \cdots, Y_{i-1})|_H = j \]

\[
= \sum_{X_1 \cdots X_{i-1}} \prod_{k=1}^{i-2} f(X_k, X_{k+1}) \cdot f(X_{i-1}, a)
\]

s.t. \( X_i = a \)

\[ a = Y_i \]

\[ |(X_1, X_2, \cdots, X_{i-1}, Y_i) - (Y_1, Y_2, \cdots, Y_{i-1}, Y_i)|_H = j \]

\[
= \sum_{X_1 \cdots X_i} f_2(X_k, X_{k+1})
\]

s.t. \( X_i = a \)

\[ a = Y_i \]

\[ |(X_1, X_2, \cdots, X_{i-1}, Y_i) - (Y_1, Y_2, \cdots, Y_{i-1}, Y_i)|_H = j \]

\[ = F_{i,j}(a) \]

Using the recursion rules shown in Theorem 4.1.2, the algorithm can be derived to efficiently inference with the constraints.
We further illustrate the intuition behind the algorithm by showing an example of the dynamic programming table in Fig. 4.3. The intuition of designing dynamic programming table entry \( F_{i,j}(a) \) is taking advantage of the chain structure by defining a variable \( a \), and solve the problem in smaller scale. In addition, another dimension gives the decomposition for the constrain: \( j \) element difference from \( Y \). \( F_{N,M} \) can be calculated from bottom up by the recursion rule and \( P(Y) = \sum_a(F_{N,M}(a)) \) is the solution for the original problem.

In general, every entry in the dynamic programming table can be calculated by its left neighbor entry or upper left neighbor entry, \( i.e., F_{i,j} \) can be calculated from \( F_{i-1,j} \) or \( F_{i-1,j-1} \) as shown in theorem. The entries in the first row and diagonal of the table have special recursion rules to calculate from its left neighbor entry and upper left neighbor entry respectively.

<table>
<thead>
<tr>
<th></th>
<th>i=1</th>
<th>i=2</th>
<th>i=3</th>
<th>i=4</th>
<th>i=5</th>
<th>i=6</th>
<th>i=7</th>
<th>i=8</th>
</tr>
</thead>
<tbody>
<tr>
<td>j=1</td>
<td>N/A</td>
<td>INIT</td>
<td>INIT</td>
<td>INIT</td>
<td>INIT</td>
<td>INIT</td>
<td>INIT</td>
<td>INIT</td>
</tr>
<tr>
<td>j=2</td>
<td>N/A</td>
<td>INIT</td>
<td>DP</td>
<td>DP</td>
<td>DP</td>
<td>DP</td>
<td>DP</td>
<td>DP</td>
</tr>
<tr>
<td>j=3</td>
<td>N/A</td>
<td>N/A</td>
<td>INIT</td>
<td>DP</td>
<td>DP</td>
<td>DP</td>
<td>DP</td>
<td>DP</td>
</tr>
<tr>
<td>j=4</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>INIT</td>
<td>DP</td>
<td>DP</td>
<td>DP</td>
<td>DP</td>
</tr>
<tr>
<td>j=5</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>INIT</td>
<td>DP</td>
<td>DP</td>
<td>DP</td>
</tr>
</tbody>
</table>

Figure 4.3: An example of \( 8 \times 5 \) dynamic programming table

The complexity of the algorithm is \( \Theta(N \cdot M \cdot |\Omega|^2) \), where \( N \) is the length of the sequence, \( M \) is the total number of elements that are different between \( X \) and \( Y \). The complexity can be intuitively explained as follows: the dimension of the dynamic programming table is \( N \times M \). For every entry, it stores a function of \( a \). The calculation for an entry requires looping over the \( \Omega \) space. In addition, evaluating the function needs to sum over the variable \( b \) as shown in (4.9), which needs another loop over \( \Omega \).
4.1.2 Inference for different obfuscation models

After discussing inference on a chain structure and inference with additional constraint, the recursion rule and algorithm can be easily derived using the idea for the attack model and obfuscation model proposed in Chapter 3. In this subsection we give the algorithm of calculating obfuscated sequence distribution for Type-I and Type-II model. For a better notation, we drop the random variable $C$ in $P(Y|C)$, which means we focus on specific attack model and obfuscation model and they are already given. In next subsection, the impact assessment for different $C$ will be discussed.

Type-I model for action alteration

For Type-I model, $P(X)$ is finite order Markov model, and $P(Y|X)$ is given in (4.10).

$$P(Y|X) = \frac{1}{N \choose M} I(|Y - X|_H = M) \prod_{k=1}^{N} P(Y_k|X_k)$$

(4.10)

Let $f(x, y) = P(X_{i+1} = y|X_i = x)$, $g(x, y) = P(Y_i = y|X_i = x)$, the noise sequence distribution can be written in (4.11).

$$P(Y) = \sum_X P(X)P(Y|X) = \sum_{X:|Y - X|_H = M} \left( P(X_1, X_2, \ldots, X_L) \prod_{k=1}^{N-L} f(X_k, \ldots, X_{k+L}) \prod_{k=1}^{N} g(X_k, Y_k) \right)$$

(4.11)

The dynamic programing function $F_{i,j}(A)$ is defined in 4.12, where the length of vector $A$ is $L$ and is used to decouple the $L^{th}$ order Markov model.

$$F_{i,j}(A) = \sum_{X: X_1, X_2, \ldots, X_i} \left( P(X_1, X_2, \ldots, X_L) \prod_{k=1}^{i-L} f(X_k, \ldots, X_{k+L}) \prod_{i=1}^{i} g(X_k, Y_k) \right)$$

s.t. $<X_{i-L+1}, \ldots, X_i> = A$

$$|<X_1, \ldots, X_i> - <Y_1, \ldots, Y_i>|_H = j$$

(4.12)
Given the function definition, the recursion rules are shown in (4.13). The proof is given in Appendix B and the algorithm implementation is shown in 2.

\[ F_{i,j}(A) = \begin{cases} 
\sum_B F_{i-1,j}(<B,A'>) \cdot f(<B,A>), & \text{if } A_L = Y_i \\
\sum_B F_{i-1,j-1}(<B,A'>) \cdot f(<B,A>)g(A_L,Y_i), & \text{if } A_L \neq Y_i 
\end{cases} \] (4.13)

**Type-II model for action insertion and removal**

For action insertion case,

\[
P(X, Y) = \left( P(X_1) \prod_{i=1}^{N/2-1} P(X_{i+1}|X_i) \right) \left( P(Y_1|X_1)P(Y_2|Y_1, X_1) \prod_{i=2}^{N/2} P(Y_{2i-1}|Y_{2i-2}, X_i)P(Y_{2i}|Y_{2i-1}, X_i) \right) = P(X)P(Y|X)
\]

\[
P(Y) = \sum_X \left( P(X_1) \prod_{i=1}^{N/2-1} P(X_{i+1}|X_i) \right) \left( P(Y_1|X_1)P(Y_2|Y_1, X_1) \prod_{i=2}^{N/2} P(Y_{2i-1}|Y_{2i-2}, X_i)P(Y_{2i}|Y_{2i-1}, X_i) \right)
\]

Let \( f(x,y) = P(X_{i+1} = y|X_i = x) \), \( g(x,y,z) = P(Y_{2i-1} = z|Y_{2i-2} = x, X_i = y) \), \( \phi(x,y,z) = P(Y_{2i} = z|Y_{2i-1} = x, X_i = y) \). The dynamic programming function can be defined in (4.14).
**Algorithm 2:** Inference algorithm to calculate $P(Y)$

**Data:** Length-$N$ sequence $Y$, $L^{th}$ order Markov model, function $g(x, y)$ and noise level $M$

**Result:** Noise sequence distribution $P(Y)$

Initialize DP Table:

for $i \in 3, 4, \cdots, N$ do

    for $j \in 1, 2, 3, \cdots, \min(M, i)$ do

        if $j = 1$ // Calculate first row in table then

            for $A \in \Omega^L$ do

                if $A_L = Y_i$ then

                    $F_{i,j}(A) = \sum_B F_{i-1,j}(<B,A'>)f(<B,A>)$

                else

                    $F_{i,j}(A) = P(X_1, X_2, \cdots, X_L)_{i-1-L} \prod_{k=1}^{i-L} f(Y_k, \cdots, Y_k+L) f(Y_{i-1-L}, \cdots, Y_{i-1}, A_L) g(A_L, Y_i)$

                end

            end

        end

    else if $i = j$ // Calculate diagonal elements then

        for $A \in \Omega^L$ do

            if $A_L \neq Y_i$ then

                $F_{i,j}(A) = \sum_B F_{i-1,j-1}(<B,A'>)f(<B,A>)g(A_L, Y_i)$

            end

        end

    else

        for $A \in \Omega^L$ do

            if $A_L = Y_i$ then

                $F_{i,j}(A) = \sum_B F_{i-1,j}(<B,A'>)f(<B,A>)$

            else

                // General recursion rule for other elements

                $F_{i,j}(A) = \sum_B F_{i-1,j-1}(<B,A'>)f(<B,A>)g(A_L, Y_i)$

            end

        end

    end

end

return $\sum_A (F_{N,M}(A))$
Based on the definition in (4.14), (4.15) shows the recursion rule. The proof is given in Appendix B.

\[ F_i(a) = \sum_{X_{i-1}} \prod_{k=1}^{i-1} P(X_1) f(X_k, X_{k+1}) \prod_{k=2}^{i} P(Y_1|X_1) P(Y_2|Y_1, X_1) g(Y_{2k-2}, X_k, Y_{2k-1}) \phi(Y_{2k-1}, X_k, Y_{2k}) \]

s.t. \( X_i = a \) (4.14)

Given the recursion rule (4.15), the algorithm can be easily derived, similar to the simplified problem discussed before. To avoid the repetition, we ignore the algorithm implementation for Type-II model.

For action removal case,

\[ P(X, Y) = \left( P(X_1) \prod_{i=1}^{N-1} P(X_{i+1}|X_i) \right) \left( P(Y_1|X_1, X_2) \prod_{i=2}^{N/2} P(Y_i|Y_{i-1}, X_{2i-1}, X_{2i}) \right) \]

\[ = P(X) P(Y|X) \]

\[ P(Y) = P(X_1) P(X_2|X_1) P(Y_1|X_1, X_2) \]

\[ \left( \sum_{X} \left( \prod_{i=2}^{N/2} P(X_{2i-1}|X_{2i-2}) P(X_{2i}|X_{2i-1}) \right) \left( \prod_{i=2}^{N/2} P(Y_i|Y_{i-1}, X_{2i-1}, X_{2i}) \right) \right) \]

Let \( f(x, y) = P(X_{i+1} = y|X_i = x) \), \( g(x, y, z, p) = P(Y_i = p|Y_{i-1} = x, X_{2i-1} = y, X_{2i} = z) \). Then we expand and re-write the product operation. The goal is make the product subscript the same, so we can define the dynamic programming function.
The dynamic programming function can be defined in (4.16).

\[
F_i(a) = P(X_1)P(X_2|X_1)P(Y_1|X_1, X_2) \\
\sum_{X_1, \ldots, X_2} \prod_{k=2}^{i} f(X_{2k-2}, X_{2k-1}) f(X_{2k-1}, X_{2k}) \prod_{k=2}^{i} g(Y_{k-1}, X_{2k-1}, X_{2k}, Y_k) \\
\text{s.t.} \quad X_{2i} = a
\] (4.16)

Recursion rules are shown in (4.17).

\[
F_i(a) = \sum_{b,c} F_{i-1}(b) \cdot f(b, c) \cdot f(c, a) \cdot g(Y_{i-1}, c, a, Y_i) \\
\] (4.17)

**Other structures with additional dependencies**

As discussed in Chapter 3, the model structure can be extended by multiple ways. Chapter 3 already illustrated how to remove dependencies between random variables. In this subsection, we discuss how to extend the model structure by adding additional dependencies on random variables and the corresponding algorithm design (inference recursion rules).

Consider the joint distribution of \(X\) and \(Y\), there are three ways of adding links:

- Adding dependences on clean sequence \(X\)
- Adding dependences on noise sequence \(Y\)
- Adding interactive dependences between \(X\) and \(Y\)

The first case is already addressed by Type-I model, which describe a \(L^{th}\) order Markov model on \(P(X)\). This subsection will give the solutions for the other two cases. Adding the dependencies on \(Y\) and between \(X\) and \(Y\) can be viewed adding another term on the recursion rule. Here, we use two examples to illustrate how to make the modification, the repetitive proofs are not listed.
One example of adding dependencies on $Y$ is shown in Fig. 4.4,

\[
P(X, Y) = \left( \prod_{i=1}^{N-1} P(X_{i+1}|X_i) \right) \left( \prod_{i=1}^{N} P(Y_i|X_i) \prod_{i=1}^{N-1} P(Y_{i+1}|Y_i) \right)
\]

Dynamic programming function $F_i(a)$ is defined in (4.18).

\[
F_i(a) = \sum_{X_1, X_2, \ldots, X_i} \left( \prod_{k=1}^{i-1} P(X_{k+1}|X_k) \right) \left( \prod_{i=1}^{i} P(Y_k|X_k) \prod_{i=1}^{i-1} P(Y_{k+1}|Y_k) \right)
\]

(4.18)

And we can easily prove the recursion rule shown in (4.19), where $f(x, y) = P(X_{i+1} = y|X_i = x)$, $g(x, y) = P(Y_i = y|X_i = x)$ and $\phi(x, y) = P(Y_i = y|Y_{i-1} = x)$.

\[
F_i(a) = \sum_b F_{i-1}(b) \cdot f(b, a) \cdot g(a, Y_i) \cdot \phi(Y_{i-1}, Y_i)
\]

(4.19)

On the other hand, another example of adding dependencies on the interactions between $X$ and $Y$ is shown in Fig. 4.5,
\[
P(X, Y) = \left( P(X_1) \prod_{i=1}^{N-1} P(X_{i+1}|X_i) \right) \left( \prod_{i=1}^{N} P(Y_i|X_i) \prod_{i=1}^{N-1} P(Y_{i+1}|X_i) \right) \\
= P(X) P(Y|X)
\]

Dynamic programming function \( F_i(a) \) is defined in (4.20).

\[
F_i(a) = \sum_{X_1, X_2, \ldots, X_i} P(X_1) \prod_{k=1}^{i-1} P(X_{k+1}|X_k) \left( \prod_{k=1}^{N} P(Y_k|X_k) \prod_{k=1}^{i-1} P(Y_{k+1}|X_k) \right) (4.20)
\]

The recursion rule for the model shown in Fig. 4.5 is listed in (4.21), where \( f(x, y) = P(X_{i+1} = y|X_i = x), g(x, y) = P(Y_i = y|X_i = x) \) and \( \psi(x, y) = P(Y_i = y|X_{i-1} = x) \).

\[
F_i(a) = \sum_b F_{i-1}(b) \cdot f(b, a) \cdot g(a, Y_i) \cdot \psi(X_{i-1}, Y_i) (4.21)
\]

## 4.2 Performance evaluation for sequence classification

This section discusses the second subproblem: how to compute the expected classification accuracy for noise sequence \( Y \), \emph{i.e.}, how to evaluate (4.2), where \( P(Y|C) \) can be calculated with Algorithm 2 and \( P(C) \) is known.

The notation of (4.2) is simple, but calculating the expectation needs to sum over exponential number of terms, respect the observation length-\( N \), similar as before, the reason is that the possible values for vector \( Y \) is \( |\Omega|^N \).

Similar to the problem of calculating \( P(Y|C) \) addressed before, calculating the expected classification accuracy in (4.2) also need to sum over exponential number of terms. However, unlike solving the problem of \( P(Y|C) \), to the best of our knowledge, there is no efficient algorithm to solve 4.2 efficiently.

There are two major reasons. First, \( P(Y|C) \) is calculated from \( P(X|C) \) and \( P(Y|X) \), as discussed earlier, there are analytical expression for \( P(X|C) \) and \( P(Y|X) \). In addition,
they satisfy the *element by element basis* property which can be decoupled by dynamic programing idea.

On the other hand, for calculating the expected classification accuracy from $P(Y|C)$, we do not have the analytical expression for $P(Y|C)$, instead, for any given $Y$, we need to execute dynamic programming in Algorithm 2 to get the value. So we will not be able to use the similar idea to solve it efficiently. Second, max operation makes the problem more complicated and eventually prohibit us to decouple the problem to *element by element basis*.

Although it is difficult to calculate the exact value of expected classification accuracy described equation 4.2, we can efficiently approximate it with arbitrary precision and confidence using Monte-Carlo method.

The reason is that we can easily get samples of $Y$ from the distribution $P(Y)$. With the samples of $Y$, it is possible to use sample average to estimate the true mean which is described in (4.22).

$$
\sum_{Y} P(Y) \psi(Y) \approx \frac{1}{n} \sum_{k=1}^{n} \psi(Y_k)
$$

(4.22)

where $Y_k$ are independent and identically distributed random variables sampled from $P(Y)$, and $\psi(Y)$ is defined as $\max_{C} P(C|Y)$ and can be calculated from $P(Y|C)$ and $P(C)$, i.e.,

$$
\psi(Y) = \max_{C} P(C|Y) = \max_{C} \frac{P(Y|C) P(C)}{P(Y)} = \max_{C} P(Y|C) P(C)
$$

(4.23)

$P(C|Y)$ is a probability value, according to the definition of $\psi(Y)$, $0 \leq \psi(Y) \leq 1$. Therefore according to Hoeffding’s bound [84], we have

$$
P \left( \left\| \frac{1}{n} \sum_{k=1}^{n} \psi(Y_k) - E[\psi(Y)] \right\| > \epsilon \right) \leq \delta
$$

(4.24)

where $n$ is number of samples and $\delta \triangleq 2 \exp(-2n\epsilon^2)$.

Applying Equation (4.24), suppose we choose
Then, with probability at least $1 - \delta$, the difference between the approximation and the true value is at most $\epsilon$.

The values for $\epsilon = 0.01$ and $\delta = 0.01$ need at least 26,492 samples. In the experiments, we use 30,000 samples. The approximation algorithm is given in 3.

**Algorithm 3:** Algorithm to estimate the expected classification accuracy

**Input:** Given $P(X|C)$, $P(C)$, $P(X|Y)$, sample size $n$

**Output:** expected classification accuracy for sequence classification

Set $S = \langle \rangle$

for $i \in \{1, 2, \cdots, n\}$ do

Sample $C_0$ from $P(C)$

Given sample $C_0$, sample $X_0$ from $P(X|C_0)$

Given sample $X_0$, sample $Y_0$ from $P(Y|X_0)$

Set $V = \langle \rangle$

for $c \in \{1, 2, \cdots, |C|\}$ do

Given $Y_0$, calculate $P(Y_0|c)$ using Algorithm 2

Append $V$ with $P(Y_0|c)P(c)$

end

Append $S$ with $\max(V)/\sum(V)$

end

return $\max(S)/\sum(S)$

### 4.3 Algorithm efficiency verification

The last section of the chapter will verify the effectiveness of Monte-Carlo approximation and dynamic programming algorithm we proposed.

For a small observation length $N$, it is possible to calculate the exact expected classification accuracy by summing over all $X$ using (4.1). We did the exact calculation for observation length from 5 to 10. On the other hand, we approximate the expected classification accuracy with different observation length by using proposed algorithm. For each given observation length, 10 repetitions of approximation experiments are conducted. For
each repetition, the 30,000 samples are used. As discussed earlier, the $\epsilon$ and $\delta$ should be smaller than 0.01.

![Figure 4.6: Monte Carlo approximation accuracy](image)

Figure 4.6 shows the box plot of differences between exact value and approximation on 10 repetitions. From the box plot we can observe that the estimation is even better than theoretical bounds. The theoretical bounds tell us the error should be smaller than 0.01 with probability 99%. The experiments of 10 repetitions show 0.003 max error for different observation length. In fact, Hoeffding bound usually is a loose bound, if we take number of samples calculated from Hoeffding bound, the result is usually better than the theoretical value.

Figure 4.6 tells us the approximation is accurate. Figure 4.7, gives the time cost comparison between approximation and exact calculation. The time cost of approximation and exact calculation of expected classification accuracy plotted in log-scale in Fig. 4.7. The exact calculation time cost increases exponentially respect to observation length $N$. This is because the summation needs to be conducted over all possible $X$, and number all possible $X$ is $|\Omega|^N$. On the other hand, the time cost for approximation is almost the same for different observation length. The reason is that no matter how many possible values $X$ can take, we only take fixed number of samples to estimate the expected classification accuracy, 30,000 in our case. For example, for $N = 10$, all possible values for $X$
is $|\Omega|^N = 6^{10} = 60, 466, 176$, we still can use 30,000 samples to estimate the expected classification accuracy with the same accuracy.

To verify the noise inference algorithm proposed, we use brute-force algorithm to calculate $P(Y = Y_0)$ by summing over all the $X$ that satisfy the constrain. The brute force algorithm has three steps. First, we produce all $X$ that satisfy the constrain $|X - Y_0|_H = M$. Then, calculate $P(X)P(Y_0|X)$. Finally, we sum over the values for each $X$. In comparison, we use dynamic programming to calculate $P(Y = Y_0)$. We will get the exact same answers by these two approaches but with different time cost.

Figure 4.8 shows the time cost for brute force inference for different $N$, $M$ and $|\Omega|$. In particular, the values are fixed, so that the time cost with brute force can be within a reasonable range.

Theoretically, brute force needs to sum over $\binom{N}{M} (|\Omega| - 1)^M$ number of terms. From Fig. 4.8, we know the time increases rapidly respect to $N$ and $M$. For sub figure 2, $M$ has a drop, this is because, the term $\binom{N}{M}$. We observe when $M$ in the middle of $N$, the system is very useful. And in security this is the case, of we observe $N$ we would not have the noise is close to $N$ or a small number.

With the same setup, the time costs for Algorithm 2 are around 0.01 second. Therefore we set the values to a much bigger scale to test the performance of dynamic programming.
time cost. Even the problem scale is much larger, the algorithm still gets the results in few seconds. This is important because in real security application, the observation window should contain hundreds of events, and the number of possible noise elements should also be a big number. But the size of $|\Omega|$ should not be too big because of Bias-Variance trade off [37]. Finally, as discussed earlier, the complexity of the algorithm is $\Theta(N \cdot M \cdot |\Omega|^2)$. In the experiments, we do observe these trends. In particular, the time cost increase linearly respect to $N$ and $M$. For a reasonable $|\Omega|$ (around 10) the time cost for proposed algorithm is less than 1.0 second.
Chapter 5

Network attack obfuscation simulation and results

5.1 Simulation framework and set up

5.1.1 Attack action space

In general, one can model network attacks at various levels with a combination of attributes reported by NIDS and host logs. To demonstrate the use of the proposed framework, this paper considers 15 classical and widely used attack actions from five categories selected from MITRE’s common attack pattern enumeration and classification [71]. The attack categories and action space $\Omega$ are shown in Fig. 5.1.

The five categories (C-0 to C-4) show different levels (stages) of the intrusion process. Abuse of functionality is a low-profile information-gathering step; taking advantage of the function provided by the target system can achieve a certain level of information collection without leaving much malicious trace, because such functions are designed to serve normal requests. For example, instead of scanning the target web server to get the server version, one can try to access a non-existing web page and observe the HTTP-404-ERROR generated by the server, which can expose the server platform and version. Network reconnaissance is a category of high-profile scanning in addition to abuse of functionality. Actions in this category are essentially taking advantage of the TCP/IP protocol, e.g., TRACEROUTE, PING, NMAP etc., to explore unknown environment. Probabilistic techniques represent another type of exploration, using a number of trials to identify vulnerabilities. For
- Abuse of functionality (C-0)
  - Detect unpublicised web pages and services (A-0-1)
  - Directory traversal (A-0-2)
  - Web server application fingerprinting (A-0-3)

- Network reconnaissance (C-1)
  - Infrastructure-based footprinting (A-1-1)
  - Host discovery (A-1-2)
  - Scanning for vulnerable software (A-1-3)

- Probabilistic techniques (C-2)
  - Fuzzing (A-2-1)
  - Screen temporary files for information (A-2-2)
  - Client-server protocol manipulation (A-2-3)
  - Dictionary-based password attack (A-2-4)

- Buffer overflow and code injection (C-3)
  - Manipulating user-controlled variables (A-3-1)
  - Command or script injection (A-3-2)
  - Hijacking a privileged process or thread (A-3-3)

- Data leakage attacks (C-4)
  - Data excavation attacks (A-4-1)
  - Data interception / sniffer attacks (A-4-2)

Figure 5.1: An example of action space (attack patterns)

Example, fuzzing is widely used in software testing by feeding the system with invalid, unexpected random inputs. By observing the system feedback, an experienced attacker can discover possible design flaws, including the chance of getting buffer overflow or code injection vulnerability, which is part of category C-3 and can eventually compromise the
target machine. After compromising the target the ultimate goal of attack can be stealing sensitive data or data excavation, e.g., generic cross-browser cross-domain thefts [71], or data interception/sniffer.

### 5.1.2 Attack models

Four attack models are defined based on the attack action space shown in Fig. 5.1. Two of them are first order and the other two are second order\(^1\). The two first order attack models are shown in Tables 5.1 and 5.2; only non-zero rows are listed. These two models are inspired from real attacks in ICTF hacking competition data set [61] [43] and CAIDA data set [65] [62]. The \((i, j)\) element in the table denotes the transition probability \(P(X_{i+1} = j|X_i = i)\).

The strategy described in Table 5.1 can be explained as a two phases attack: reconnaissance and intrusion phase. The attacker is more hesitant to switch between phases than stay within a phase. The specific probability numbers in the table can reflect the characteristics of the attack, e.g., the automatic script the attacker is using. In fact, our experience suggests that the probabilities of action transitions are quite reliable for detecting the attack tools such as Metasploit [76] or Nessus [85]. The model shown in Table 5.2 reflects a different attack strategy: the attacker utilizes the reconnaissance actions throughout the attack process, and perform specific exploits only sporadically.

Table 5.3 shows Model-3, the first of the two second order attack models used for the experiments. Every row in the Table 5.3 represents a combination of two past actions, \(X_{i-1}, X_i\). There are a total of \(|\Omega|^3\) number of parameters in the model and most parameters are zero. Table 5.3 only shows a fraction of the non-zero parameters. The design of Model-3 is describing attack behavior more specific. As an example, the first row indicates that the attacker is doing host scanning to discover live hosts and live services. This usually takes a long time in order to obtain sufficient information. Therefore, the attacker may keep doing

\(^1\)As pointed out by Fava et al. [35] and Du et al. [51], most attack behaviors can be captured with first and second order Markov models. Furthermore, higher order models can be too specific with high complexity and perform poorly because of Bias-variance trade-off [37].
Table 5.1: Attack model 1 (partial) used in simulation

<table>
<thead>
<tr>
<th></th>
<th>A-0-1</th>
<th>A-1-2</th>
<th>A-2-1</th>
<th>A-3-1</th>
<th>A-4-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-0-1</td>
<td>0.30</td>
<td>0.30</td>
<td>0.20</td>
<td>0.10</td>
<td>0.10</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>A-1-2</td>
<td>0.30</td>
<td>0.20</td>
<td>0.30</td>
<td>0.10</td>
<td>0.10</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>A-2-1</td>
<td>0.20</td>
<td>0.25</td>
<td>0.40</td>
<td>0.10</td>
<td>0.05</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>A-3-1</td>
<td>0.10</td>
<td>0.10</td>
<td>0.10</td>
<td>0.40</td>
<td>0.30</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>A-4-2</td>
<td>0.10</td>
<td>0.10</td>
<td>0.10</td>
<td>0.10</td>
<td>0.60</td>
</tr>
</tbody>
</table>

Table 5.2: Attack model 2 (partial) used in simulation

<table>
<thead>
<tr>
<th></th>
<th>A-0-1</th>
<th>A-1-2</th>
<th>A-2-1</th>
<th>A-3-1</th>
<th>A-4-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-0-1</td>
<td>0.30</td>
<td>0.30</td>
<td>0.30</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>A-1-2</td>
<td>0.40</td>
<td>0.30</td>
<td>0.20</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>A-2-1</td>
<td>0.20</td>
<td>0.20</td>
<td>0.30</td>
<td>0.15</td>
<td>0.15</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>A-3-1</td>
<td>0.20</td>
<td>0.20</td>
<td>0.30</td>
<td>0.20</td>
<td>0.10</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>A-4-2</td>
<td>0.20</td>
<td>0.20</td>
<td>0.20</td>
<td>0.20</td>
<td>0.20</td>
</tr>
</tbody>
</table>

A-0-1 following previous two A-0-1. In reality, it is very likely to observe a long sequence of A-0-1. The second of the two second order model, Model-4, reflects similar behavior on the high-profile phase but much more specific on describing the long sequence of certain vulnerability attempts.

### 5.1.3 Obfuscation models

Both Type-I and Type-II obfuscation models discussed in Chapter 3 are used in the simulation. Table 5.4 shows a subset of the function $g(x, y)$ for action alteration, which defines the obfuscation behavior and will be used for the subsequent action alteration simulations. The
Table 5.3: Attack model 3 (partial) used in simulation

<table>
<thead>
<tr>
<th></th>
<th>A-0-1</th>
<th>A-1-2</th>
<th>A-2-1</th>
<th>A-3-1</th>
<th>A-4-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-0-1,A-0-1</td>
<td>0.96</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>A-0-1,A-1-2</td>
<td>0.35</td>
<td>0.66</td>
<td>0.03</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>A-3-1,A-0-1</td>
<td>0.30</td>
<td>0.10</td>
<td>0.10</td>
<td>0.40</td>
<td>0.10</td>
</tr>
<tr>
<td>A-3-1,A-1-2</td>
<td>0.10</td>
<td>0.30</td>
<td>0.10</td>
<td>0.40</td>
<td>0.10</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(i,j)th element gives the value of \( g(x = i, y = j) \). This function reflects that the obfuscation behavior will mostly alter attack actions within the same category, but occasionally change from one to another category.

Table 5.4: Attack obfuscation model (partial) used in simulation

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>A-0-1</th>
<th>A-1-2</th>
<th>A-2-1</th>
<th>A-3-1</th>
<th>A-4-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-0-1</td>
<td>0.00</td>
<td>0.40</td>
<td>0.40</td>
<td>0.10</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>A-1-2</td>
<td>0.40</td>
<td>0.00</td>
<td>0.40</td>
<td>0.10</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>A-2-1</td>
<td>0.40</td>
<td>0.40</td>
<td>0.00</td>
<td>0.10</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>A-3-1</td>
<td>0.10</td>
<td>0.20</td>
<td>0.30</td>
<td>0.00</td>
<td>0.50</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>A-4-2</td>
<td>0.10</td>
<td>0.10</td>
<td>0.30</td>
<td>0.50</td>
<td>0.00</td>
<td></td>
</tr>
</tbody>
</table>

5.1.4 Simulation overview

The simulation overview is shown in Fig. 5.2. From the security domain knowledge, we have multiple attack models. These models are in different orders and have different parameters. The attack sequences are generated from these attack models. After attack sequence generation, the mixed attack sequences will be feed into different attack obfuscation techniques discussed in Chapter 3. The sequence classifiers for clean sequence and
noise sequence assume the full knowledge of the attack model but not attack obfuscation model. They compare the likelihood for a given observed sequence cross different generative models to perform classification. On the other hand, noise inference algorithm utilize the knowledge of attack obfuscation module (as shown in Section 5.2.3, inaccurate estimation of the attack obfuscation can also be acceptable) to calculate the distribution of the obfuscated sequence eventually to make the sequence classifier better. We will compare the classification results for clean attack sequence, noise attack sequence and using the proposed inference algorithm. The performance metric is defined as the optimal classification rate which is specified in (4.1).

![Diagram](image)

Figure 5.2: Simulation overview for attack sequence classification with obfuscations

### 5.2 Type-I model simulation and results

#### 5.2.1 A case study of action alteration obfuscation

In this section, a case study of obfuscated multistage attack is studied to show how the algorithm proposed in this paper can help security analysts to understand the obfuscated attack sequence better. Figure 5.3 gives the network diagram used for the case study. It
describes a small enterprise network with six subnets, eleven servers and four clusters of hosts (24 hosts in total). The whole network has 31 open services (15 types total) and interconnected via four routers.

The intrusion scenario is described in Table 5.5. The attacker began the attack from compromised hosts in the Internet. The external servers (web server and file server) were first explored with abuse of functionality. After few steps, the attacker obtained the vulnerability information and performed a buffer overflow attack on the file server and compromised the external file server 192.168.1.3. Using this stepping stone, the attacker compromised the internal server (Domain controller 192.168.3.1) and use it to probe the hosts in Department C (192.168.30.x). The actual attack sequence was obfuscated with a few observed actions altered. In particular, actions #5,#8,#13,#15, were observed as low-profile explorations, i.e., abuse of functionality and network reconnaissance.

Given the attack models and the observed sequence, one can calculate the likelihood
Table 5.5: Network attack sequence used in the case study

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9.5.231.72</td>
<td>192.168.1.3</td>
<td>A-1-1</td>
<td>A-1-1</td>
</tr>
<tr>
<td>3</td>
<td>9.5.231.72</td>
<td>192.168.1.3</td>
<td>A-2-1</td>
<td>A-2-1</td>
</tr>
<tr>
<td>4</td>
<td>9.5.231.72</td>
<td>192.168.1.2</td>
<td>A-2-2</td>
<td>A-2-2</td>
</tr>
<tr>
<td>5*</td>
<td>9.5.231.72</td>
<td>192.168.1.3</td>
<td>A-3-1</td>
<td>A-0-3*</td>
</tr>
<tr>
<td>6</td>
<td>237.21.22.14</td>
<td>192.168.1.3</td>
<td>A-3-1</td>
<td>A-3-1</td>
</tr>
<tr>
<td>7</td>
<td>9.5.231.72</td>
<td>192.168.1.3</td>
<td>A-3-2</td>
<td>A-3-2</td>
</tr>
<tr>
<td>8*</td>
<td>192.168.1.3</td>
<td>192.168.3.1</td>
<td>A-3-1</td>
<td>A-1-3*</td>
</tr>
<tr>
<td>9</td>
<td>192.168.1.3</td>
<td>192.168.3.1</td>
<td>A-3-1</td>
<td>A-3-1</td>
</tr>
<tr>
<td>10</td>
<td>192.168.3.1</td>
<td>192.168.30.1</td>
<td>A-3-2</td>
<td>A-3-2</td>
</tr>
<tr>
<td>11</td>
<td>192.168.3.1</td>
<td>192.168.30.1</td>
<td>A-3-1</td>
<td>A-3-1</td>
</tr>
<tr>
<td>12</td>
<td>192.168.3.1</td>
<td>192.168.30.1</td>
<td>A-3-2</td>
<td>A-3-2</td>
</tr>
<tr>
<td>13*</td>
<td>192.168.3.1</td>
<td>192.168.30.1</td>
<td>A-3-1</td>
<td>A-1-3*</td>
</tr>
<tr>
<td>14</td>
<td>192.168.3.1</td>
<td>192.168.30.1</td>
<td>A-3-2</td>
<td>A-3-2</td>
</tr>
<tr>
<td>15*</td>
<td>192.168.3.1</td>
<td>192.168.30.1</td>
<td>A-3-1</td>
<td>A-1-2*</td>
</tr>
<tr>
<td>16</td>
<td>192.168.3.1</td>
<td>192.168.30.1</td>
<td>A-3-2</td>
<td>A-3-2</td>
</tr>
</tbody>
</table>

for each model, i.e., how much the observed sequence fit for specific strategies. Figure 5.4 gives the likelihood comparison with respect to the four aforementioned attack models, when the observed sequence is (1) consisted of the original attack actions without obfuscation (Clean), (2) altered with the obfuscated actions (Noise), and (3) recovered with the inference algorithm discussed in Chapter 4 (InfAlg).

The original clean attack sequence shown in Table 5.5 includes a sequence of actions in C-3 (high-profile vulnerability attempts). Comparing the four models, Model-4 describes such behavior the best, and hence the likelihood to fit with Model-4 in Fig. 5.4(a) is the highest. However, the obfuscated actions break the pattern in C-3 and the likelihood to fit for the four models is completely different; Model-1 become the most likely model as shown in Figure 5.4(b). This is because Model-1 describes a scenario of bouncing between reconnaissance and vulnerability attempts to discover potential targets instead of the focused attacks. In real-world application, such mis-classification could impact the analyst’s decision making, e.g., having an inaccurate estimation of attacker’s capability
The impact of obfuscation in this case is clearly profound, and needs an efficient way to recover the likelihood of matching to Model-4 even in the presence of the obfuscated actions. The proposed method does not recover the exact original sequence; instead, it allows efficiently calculate $P(Y|C)$ for each model $C$, so as to determine the best match. In this case, if one knows that there are 4 actions out of 16 altered, there are $\binom{16}{4} = 1820$ possible combinations of the 4 altered positions. For each altered position combination, there are 465,920 possible noise sequences. The proposed algorithm efficiently recover, or re-distribute the likelihood of matching to the four models as shown in Fig. 5.4(c). The resulting likelihood distribution allows the analyst to conclude (recover) Model-4 as the best match.

\subsection{Action alteration impact evaluation}

The algorithm proposed in our framework enables us to assess the impact for attack obfuscations. This subsection expands on the example shown above to treat a large number of obfuscated sequences with random obfuscation behaviors. We shall first describe more
specifically how the use of the algorithm systematically determine the best match to one of the four models.

For a given sequence, whether it is the original sequence $X$ or the obfuscated sequence $Y$, the optimal match one can do is to find $\max_{C} P(C|X)$ or $\max_{C} P(C|Y)$, which can be found with $P(X)$ and $P(Y)$ based on Bayes rule, respectively. The algorithm discussed calculates $P(Y)$ efficiently, and $P(X)$ can be directly derived based on the given attack models. Note that $\max_{C} P(C|Y)$ represents the best likelihood one can match a given sequence $Y$ to a model $C$, and the $1 - \max_{C} P(C|Y)$ also implies the least error for the given sequence and the models. To assess the overall impact to any sequence, clean or obfuscated, that can occur under the attack models, one will need to calculate $\sum_{X} P(X) \max_{C} P(C|X)$ and $\sum_{Y} P(Y) \max_{C} P(C|Y)$, which are termed Optimal Classification Rate in this paper. To efficiently assess the optimal classification rate over a large number of possible sequences under different scenarios, this paper uses Monte Carlo method to approximate the summation. According to Hoeffding’s bound [84], at least 26,492 samples are needed to keep the approximation error within 1%; for each scenario, 30,000 samples are used in the simulation shown below.

Similar to the specific case study shown earlier, the simulation shown here also considers the case where no inference or recovery is done for the obfuscated sequences - referred to as the “noise” case, in contrast to the “clean” case and the “InfAlg” case. The optimal classification rate for the different cases are plotted when the first-order and the second-order models are used to generate and classify attack sequences, respectively. The first-order and second-order cases are separately considered to evaluate the effect of the order of the model on the classification performance, and, thus, errors.

Figure 5.5 shows the impact of attack obfuscation for different sequence lengths observed when the obfuscation level ($M/N$) is fixed at 20%. The clean curves show the optimal classification rate when the original attack sequences are classified to the models. The noise curves give the performance on obfuscated sequences, but without assuming the knowledge of these sequences were obfuscated. Finally, the InfAlg curves take advantage
of the obfuscation model information and utilizes the inference algorithm proposed in this paper to perform classification.

The performance for all curves increases as the observation length increases. This is intuitive, the more observations exhibited from the same attack behavior, the easier one can differentiate sequences among attack models. Second, the “InfAlg” curve is always above the “noise” curve and below the “clean” curve for both the first-order and the second-order cases; this shows quantitatively how much performance recovery one can optimally achieve with the (limited) knowledge of the obfuscation behavior. In addition, the plot shows that using the second order model gives better results than using the first order. This is also expected because differentiating among more specific second-order models is easier than doing so among first-order models. One should note that mixing first and second order models, however, do not necessarily make it easier as evidenced by the case study shown in the previous section; it depends on the characteristics of the models considered.
This experiment result provides not only insights that verifies intuitions but also provide quantitative assessment of the optimal performance one can achieve. For example, Fig. 5.5 shows that with the use of the inference algorithm and an observation length of 30, one can recover from 82% classification rate to 85% for first-order models, and 90% to 95% for second-order models. Comparing these numbers to the clean curves, one can assess how much the best one can do with the given knowledge of obfuscation. With this framework, one can test out the impact of obfuscation for different combinations of attack models and obfuscation models.

A key property of the obfuscation model considered in Type-I model is the estimated level of obfuscation $M/N$. Figure 5.5 shows the performance for different obfuscation level when the sequence length is fixed at 40. For both first and second order-model cases, the optimal classification rates for the original clean sequences are around 99% and 97%, respectively; these numbers give the performance limit one can ever aim at for each scenario. When the obfuscation level increases, the performance drops, especially for the second-order model case without inference. At around the obfuscation level of 28%, the obfuscated sequences without inference for the second-order case actually begin to exhibit worse performance than that for the first-order case. Fortunately, with the limited knowledge of obfuscation, the optimal classification rate can be recovered, e.g., from 60% to 90% when the obfuscation level is at 40%. Interestingly, the performance recovered through inference for the second-order model case remains better than that for the first-order case, at least up to the 45% obfuscation level, which is very high. Generally speaking, the higher the obfuscation level, the more improvements one can achieve, for both first and second-order cases. The performance recovered through inference is closer to the absolute limit exhibited by the clean curves for the second-order model case that that for the first-order model case, at least when the obfuscation level is not too high.
5.2.3 Evaluation the impact inaccurate obfuscation model

As discussed in subsection 5.2.2, one important parameter in obfuscation model for action alteration is the obfuscation level, \( i.e. \), how much action alteration exists in the sequence. In order to run the noise inference algorithm, this parameter \( M \) is assumed to be known. In real application cases, one may argue that it is not reasonable for security analysts to know how attacker perform obfuscation in such a detailed level, \( e.g. \), number of actions changed. Therefore, in this subsection, we want to investigate how the parameter \( M \) impact the proposed algorithm, and how much impact the inaccurate estimation of \( M \) can cause empirically. We will show that, only an approximation of noise level is needed to get a reasonably good inference results, which means even the analysts only have a rough estimation of how much obfuscation exists, the proposed algorithm will still be very useful for helping classify attack sequences.
Using the same experiment design shown in Section 5.1, the obfuscated attack sequences were created using the true obfuscation level value $M_{true}$. On the other hand, we intentionally set the inaccurate estimation of obfuscation level used for the inference. The algorithm is executed based on different $M_{est}$ values.

Figure 5.7 and Figure 5.8 shows the results of inaccurate obfuscation level impact estimation. Figure 5.7 shows the inaccurate $M$ respect to observation length. Clean sequence length $N$ from 10 to 60, and the real obfuscation level $M_{true}$ is 40%. We estimated inaccurate $M_{est}$ from 20% and 60%. From the figure we can observe, correct $M=40\%$ have the best performance, the curve is on the top of other curves. At the same time, the more $M_{est}$ close to real value $M_{true}$, the better performance it is, i.e., $|M_{true} - M_{est}| = 10\%$ are better than $|M_{true} - M_{est}| = 20\%$. The most important information is that the inaccurate $M$ would not affect the performance too much. Note that, in this experiment even the obfuscated sequences have changed 60% comparing to the original clean sequence, we shall
can observe the big improvement comparing to the noise line. Finally, these trends remain the same for different observation length.

Figure 5.8 evaluates the inaccurate $M$ estimation respect to noise level. $M_{true}$ changes from 30% to 55%, and on different noise level, the $M_{est}$ deviate $M_{true}$ from 10% and 20%. From the figure we can observe the similar trends comparing to Fig. 5.7. The more obfuscation exists, the worse the performance is, as shown in the decreasing noise curve. At the same time, the noise inference curves are not decreasing too much comparing to the noise curve. Further, the impact of the inaccurate $M$ is still relative small: all of the noise inference curves are close to each other, although $M_{est} = M_{true}$ is the best. Finally, these trends remain the same for different noise levels from 30% to 55%.

Figure 5.9 and Fig. 5.10 gives when noise level $M$ approaches extreme cases, i.e., $M = 0.0$ and $M = 1.0$. Note that, for $M = 0.0$ the inference algorithm will not be able to work, because according to the noise model definition, $P(Y|X) = 0$ (detailed discussion...
can be found in Chapter 3.2.1). From the simulation results, one can conclude that with less obfuscated events, the expected performance increases for the noise curve. And the InfAlg curve improvement decreases. On the other hand, at the small noise level, inaccurate estimation of $M$, will be worse than directly use the noise sequence for classification.

For $M$ approach 1.0 case, the simulation results are shown in Fig. 5.10. Similar to previous observations, the nosieINF can always make a bit performance improvement and the inaccurate $M$ estimation does not affect the performance too much. Further, one interesting observation is that the curves are not monotonically decrease, i.e., the performance for changing all of the actions is even better than changing 95% of the actions. This counterintuitive observations can be explained with the combinatorial number of possible noise sequences. Specifically, as discussed earlier, the possible noise sequences for given $M$ is $\binom{N}{M} (|\Omega| - 1)^M$. Therefore number of possible sequences will reduce when $M = 1.0$, i.e.,
number of changes equal to sequence length case. Similar trend can also be observed in the time cost evaluation figure (Fig. 4.8).

5.3 Type-II model simulation and results

5.3.1 Action insertion simulation

For Type-II model action insertion case, two specific scenarios are designed for simulations. Based security domain knowledge, we believe they are representative to real action insertion in attack.

Case one describes an attacker injecting independent noise observations on the attack sequences, which means, the injected action have nothing to do with previous attack action.

---

Here we want to clarify the notation of $M$. In the action alteration obfuscation model, $M$ is defined as a positive integer and represent number of changes in the sequence. For better presentation, we changed noise level into percentile for all of the figures. In the term $\binom{N}{M}$, $M$ represent number of changes.
and the "clean attack actions". This means the injected action has its own distribution conditionally independent to other random variables. For example, the injected noisy actions can have 80% of abuse of service action, and 20% of network reconnaissance action.

On the other hand, case two describes a more complicated action injection plan, the injected action actually have some relationship with the clean actions. For example, in the network reconnaissance stage and vulnerability attempt stage, the attacker would have different preference to inject more actions in some categories than other categories. Such noise injection plan can be effective to confuse the alert analysis engine on the intrusion stage assessment and eventually cover the intrusion stage and hide the real intent.

According to the attack scenario description and dependencies between random variables, two action injection scenarios can be described graphically in Fig. 5.11. Comparing to the general noise insertion model described in Fig. 3.6, conditional dependencies (links in the graph) need to be removed to reflect aforementioned scenarios. As discussed in Chapter 3, section 3.2.2, link removing can be easily done by setting special values in the model parameters.

![Graphical representation of action insertion case 1](image)

Figure 5.11: Graphical representation of action insertion case 1

The simulation results are different than want we expected but very reasonable to explain. Original we thought action insertion scenario one is much simpler comparing to scenario two (injection pure random noise vs. well planned action injection obfuscation) and should be easily treated. The inference result for scenario one should have better performance than two. However, the results are opposite: using the same simulation set up
(action space, network configuration, attack models), the inference performance for scenario two is much better than scenario one as shown in Fig. 5.13. The InfAlg curve for insertion scenario one overlaps with the noise curve and shows almost nothing improved. This means that by only injecting the independent noisy observations, if the alert analysis system did not filter unrelated action ahead and feed the noise sequence into the pattern match engine, such case would have a big impact on the sequence classification.

Such results can be explained intuitively with the evaluation framework. In the experiment set up, we assume we know the full knowledge of attack model and noise model\(^3\), i.e., the joint distribution of clean sequence \(X\) and obfuscated sequence \(Y\) and our contribution is derive efficient algorithm to compute specific probabilities. Because we know the joint distribution, the more information we know about the obfuscation model, the better performance we can have for the proposed inference algorithm. On the other hand, in the action insertion scenario one is more uncertain (the probability distribution has a high entropy) because of the independent distribution on inserted actions, the inference algorithm will help little. As a result, we can conclude that, the more we know and the more accurate we know about the relationship between action with the clean action, the better performance we can achieve. In addition, it is very important to exclude or filter out the *unrelated noisy attack actions* before feeding the sequence into the pattern match engine.

\(^3\)As shown in section 5.2.3, inaccurate estimation of model can also be acceptable.
5.3.2 Action removal simulation

For action removal simulation, we also evaluated two different cases. Case one can be described in Fig. 5.14. The model does not have connections in $Y$ and one of two clean actions can be removed by attacker. The security context behind this model is that the attacker would perform attempts on different services and vulnerabilities, and sometimes using high profile attack actions such buffer overflow attempt, and sometimes using low profile attack actions such as abuse of functionality. And the attacker is keep doing such attacks on different services. Therefore, suppose the observations will depend on sensor configuration, for some services, high profile actions can be observed, and for some other services low profile actions can be observed. We spent some time to select the attack actions and set parameters to be more realistic to reflect the actions can be observed.

Another type of action removal is shown in Fig. 5.15. it does not have the connection from $X_{2i}$, $i \in \{1, 2, 3, \cdots\}$ (subscript is even number) to $Y$. And this model represents a
more stealthy and decoy attack. There are covering up actions with the real actions. We assume the covering up actions should have relationship (conditional dependencies) with the real actions which is different from the case of attack generate "random actions", but generate "related actions" to mislead security analyst. On the other hand, because the goal of such action is misleading, the $Y$ is also related to previous actions which are the links within $Y$.

The simulations set up are as same as previous cases (action space, network, attack models), and the results are shown in Fig. 5.16.

From the results we can conclude that it is easier for network analyst to deal with the case one than case two. The noise curve and InfAlg curve for case one are much higher than case two. This observation is intuitive because our according to our design the case two is more advanced obfuscation in terms of the attacker intentionally mis-leads the analyst. Further, the noise curve is around 0.5, which is the lower bound of classify two models. This tells us that if the attacker can really do the decoy obfuscation described before, that is
Figure 5.16: Simulation results for two different action removal (AR) models

actually very misleading and we can do nothing, if we only have noise sequence. However, the noise inference for the case two line has some gaps between the noise line and noise inference line. It tells us that if we know the attack model and noise model to some level, the noise inference algorithm would still be very useful. Finally, the we notice the InfAlg curve for case two and noise curve for case one are very similar. This tells the best results for case have is approximately equal to the worse case for case one. This is because for both cases, we are trying to inference from one of the two actions. And our inference in case one can be viewed as removing the decoy/misleading factors (edges between Y) but case two would have both stealthy and decoy factors.

Finally, we compare the impact for different types of obfuscations: noise insertion (NI), action alteration (AA) and action removal (AR). The aggregated view of performance limit for different scenarios are shown in 5.17. Two first order attack models listed in 5.1 and 5.1 are selected to run the simulation. For Type-I model, noise level was set to 20%. For
Figure 5.17: Comparison for different attack obfuscation techniques

Type-II model, obfuscation technique listed in Fig. 5.12 and Fig. 5.14 were used. From the simulation results we can conclude that, comparing to action alteration, action insertion and removal could cause more impact on the system, as shown in the figure the noise curve for action and removal are in the bottom. On the other hand, the improvement of the noise insertion case can be big if we know some information about the relationship between the injected noise actions with clean attack actions.
Chapter 6

Conclusion

The mixture of organized cyber crimes and random attacks against enterprise and government networks has led to asymmetric cyber battlefields filled with large-scale cyber attacks. To obtain a timely situation awareness from overwhelming, diverse and evolving data, analysts can benefit from effective computational techniques performing intrusion detection, alert correlation, attack characterization and prediction, host clustering. Drawing the analogy from social network analysis, we define an attack social graph to represent the relationship between attack sources. Applying the notion of degree centrality and agglomerative hierarchical clustering, various types of collaborative attack, or spatial patterns are discovered. These spatial patterns enable a labeling scheme for attack sources over time, resulting in an integrated spatial and temporal model for collaborative attack sources. Markov models are developed to differentiate and infer cyber attack strategies worthy of further investigation. The experiment results using Network Telescope and ICTF data show that the integrated spatial and temporal analyses can provide additional insights for high impact attacks that are not trivial by applying traditional statistical or anomaly analyses.

Further, moving beyond intrusion detection, network security can benefit from projection of multistage attacks, where likely future targets can be identified for timely responses. Projecting cyber attacks requires extraction and analysis of various characteristics, including the history of patterns exhibited in attacks' progression in the network. While previous work introduced attack assessments based on these characteristics, we revisited them and presented fuzzy VLMM framework to combine the attack projection estimates. Thorough
analysis via simulation were presented to provide insights toward ensemble characterization of multistage attacks. The analysis reveals that fuzzy VLMM framework can effectively capture sequential patterns of attack progression and the combined estimates can be very useful for action prediction. More importantly we demonstrated the impact can be large for different types of noises exist in the attack sequence.

Finally, we reviewed attack obfuscation and countermeasures and propose and solve the inference problem for obfuscated attack sequences, which enables the study of the benefits and limitations of attack sequence modeling and classification. Recovering the most likely original attack sequence and revealing the optimal classification rates under different attack models allow assessing the value of developing and using attack models that can be used to correlate observed events. In real applications, long observation window of attack sequences, large number of noisy observations, large number of possible attack actions or high order of generative Markov models can all lead to overwhelming computation time. The inference algorithm developed in this paper provides a mean to determine the optimal classification rates under different scenarios efficiently.

The framework and methods developed in our work can also be applied to other contexts beyond network security. Any behavior sequences that might suffer from noise and require matching to pre-defined models can use this work to recover the most likely clean sequence or evaluate quantitatively the optimal performance one can achieve to separate the instances.
Bibliography


[74] DARPA Cyber Insider Threat (CINDER) Program. (Access Date: May. 2014). [Online]. Available: https://www.fbo.gov/index?s=opportunity&mode=form&tab=core&id=585e02a51f77af5cb3c9e06b9cc82c48&cvview=1


Appendix A

Snort alerts explanation

- Alert: ICMP PING NMAP.
  - Summary: This event is generated when an ICMP ping typically generated by nmap is detected.
  - Impact: This could indicate a full scan by nmap which is sometimes indicative of potentially malicious behavior.
  - Detailed information: Nmap’s ICMP ping, by default, sends zero data as part of the ping. Nmap typically pings the host via icmp if the user has root privileges, and uses a tcp-ping otherwise.

- Alert: WEB-MISC cat%20 access
  - Summary: This event is generated when an attempt is made to exploit a known vulnerability on a web server or a web application resident on a web server.
  - Impact: Information gathering and system integrity compromise. Possible unauthorized administrative access to the server. Possible execution of arbitrary code of the attackers choosing in some cases.
  - Detailed information: This event is generated when an attempt is made to compromise a host running a Web server or a vulnerable application on a web server. Many known vulnerabilities exist for each implementation and the attack scenarios are legion. Some applications do not perform stringent checks when validating the credentials of a client host connecting to the services offered on a host server. This can lead to unauthorized access and possibly escalated privileges to that of the administrator. Data stored on the machine can be compromised and trust relationships between the victim server and other hosts can be exploited by the attacker.
• Alert: WEB-MISC http directory traversal
  – Summary: This event is generated when an attempt is made to execute a directory traversal attack.
  – Impact: Information disclosure. This is a directory traversal attempt which can lead to information disclosure and possible exposure of sensitive system information.
  – Detailed information: Directory traversal attacks usually target web, web applications and ftp servers that do not correctly check the path to a file when requested by the client. This can lead to the disclosure of sensitive system information which may be used by an attacker to further compromise the system.

• Alert: WEB-MISC /etc/passwd
  – Summary: This event is generated when an attempt is made to retrieve a protected system file on a host via a web request.
  – Detailed information: The passwd file usually found in the /etc/ directory on UNIX based systems, contains login information for users of a host. If shadow password files are not being used, an attacker could obtain valid login information for the system by using widely available password cracking tools on the file. The file may also be used to garner information that may be used in brute force password guessing attacks against the host.

• Alert: SHELLCODE x86 NOOP
  – Summary: A series of NOP instructions for Intel’s x86 architecture was detected.
  – Impact: As part of an attack on a remote service, an attacker may attempt to take advantage of insecure coding practices in hopes of executing arbitrary code. This procedure generally makes use of NOPs.
  – Detailed information: The NOP allows an attacker to fill an address space with a large number of NOPs followed by his or her code of choice. This allows "sledding" into the attackers shellcode.

• Alert: WEB-MISC robots.txt access
Summary: This event is generated when an attempt is made to access the file robots.txt directly.

Impact: Information gathering.

Detailed information: Robots.txt access is usually made by search robots for site indexing. A webmaster sometimes adds information for areas of the site that should not be indexed by the engine. This can include user directories and files and directories used in administration of the server. The information gathered from robots.txt could be used for system compromise and control of the web server.

Alert: WEB-ATTACKS /bin/ls command attempt

Summary: Attempted ps command access via web

Impact: Attempt to gain information on system files and filestructure

Detailed information: This is an attempt to gain intelligence on the filesystem on a webserver. The ls command lists the files and filesystem layout on a UNIX or Linux based system. The attacker could possibly gain information needed for other attacks on the host.

Alert: WEB-MISC ls%20-l

Summary: This event is generated when an attempt is made to exploit a known vulnerability on a web server or a web application resident on a web server.

Impact: Information gathering and system integrity compromise. Possible unauthorized administrative access to the server. Possible execution of arbitrary code of the attackers choosing in some cases.

Detailed information: This event is generated when an attempt is made to compromise a host running a Web server or a vulnerable application on a web server. Many known vulnerabilities exist for each implementation and the attack scenarios are legion. Some applications do not perform stringent checks when validating the credentials of a client host connecting to the services offered on a host server. This can lead to unauthorized access and possibly escalated privileges to that of the administrator. Data stored on the machine can be compromised and trust relationships between the victim server and other hosts can be exploited by the attacker.
• Alert: INDICATOR-OBFUSCATION base64-encoded uri data object found
  – Summary: This event is generated when network traffic that indicates a base64-encoded uri data object found has been detected in network traffic.
  – Impact: Unknown.
  – Detailed information: This event indicates that network traffic indicating that base64-encoded uri data object has been detected. Attackers may exploit systems by embedding certain filetypes within other files or by using encoding schemes.

• Alert: WEB-IIS header field buffer overflow attempt
  – Summary: This event is generated when an attempt is made to overflow a buffer in HTTP header field handler of Microsoft Internet Information Server (IIS) versions 4.0, 5.0, and 5.1.
  – Impact: Denial of Service, arbitrary code execution. Full administrative control is possible.
  – Detailed information: A vulnerability exists in HTTP header process in ASP.DLL, a specially crafted packet sent to this processor will allow an attacker to disrupt the ISS service or run any arbitrary commands with the privileges of the ASP ISAPI extension.

• Alert: SCAN nmap fingerprint attempt
  – Summary: This event is generated when the nmap port scanner and reconnaissance tool is used against a host. When run with the ’-O’ option, it attempts to identify the remote operating system.
  – Impact: Can provide useful reconnaissance information to an attacker. Has been known to cause a denial of service on some older hosts.
  – Detailed Information: Nmap attempts to identify the remote operating system by looking for different services that are common or specific to particular operating systems. It also sends a variety of abnormal packets that are often handled differently by different operating systems so that it can differentiate between them based on the responses.

• Alert: SCAN nmap TCP
– Summary: This event is generated when the nmap port scanner and reconnaissance tool is used against a host.

– Impact: This could be part of a full scan by nmap and could indicate potential malicious reconnaissance of the targeted network or host.

– Detailed Information: Some versions of Nmap’s TCP ping, if selected, sends a TCP ACK with an ACK number = 0. Nmap can use TCP ping as a second alternative to ICMP Ping.

• Alert: SCAN nmap XMAS

– Summary: A nmap XMAS scan was detected.

– Impact: System reconnaissance that may include open/closed/firewalled ports, ACLs.

– Detailed Information: Nmap sets the URG PSH and FIN bits as part of it’s XMAS scan. Typically, a closed port will respond with an ACK RST, whereas an open port may not respond at all. However, this varies from machine to machine, and also depends on what (if any) filtering policies are in place between the hosts in question.

• Alert: FTP command overflow attempt

– Summary: This event is generated when an attempt is made to send an overly long FTP command, possibly with the intent to cause of denial of service or buffer overflow in the 3CDaemon FTP server.

– Impact: Attempted remote access or denial of service. Successful execution of this attack can cause a denial of service or buffer overflow, allowing the execution of arbitrary commands on the vulnerable FTP server.

– Detailed Information: 3CDaemon is an FTP server for Windows hosts. A buffer overflow vulnerability exists in 3CDaemon revision 10. The exploit is caused by sending an FTP command that is 400 bytes or longer, causing the server to crash or permitting a buffer overflow that may allow the execution of arbitrary commands with the privileges of the process running the FTP server. This attack does not require login access to the FTP server.

• Alert: INDICATOR-COMPROMISE c99shell.php command request - ls

– Summary: This event is generated when activity relating to the “c99shell.php” Trojan Horse program is detected.
Impact: Possible theft of data and control of the targeted machine leading to a compromise of all resources the machine is connected to.

Detailed Information: Trojan horse programs can be used by an attacker to steal data from the infected machine, they can also be used to control the infected host. This event indicates that activity relating to the trojan horse program c99shell.php has been detected in network traffic. In particular this event indicates that the software detected is a Remote Access Trojan. RAT programs allow full control of the target system using a client on the attackers machine that connects to the server on the client host.
Appendix B

Recursion rule theorem proof

According to definition expand $\sum_B F_{i-1,j}(<B, A_1 >, \ldots, A_{L-1} >)$, we have

$$\sum_B F_{i-1,j}(<B, A_1 >, \ldots, A_{L-1} >) \cdot f(<B, A >)$$

$$= \sum_B \left( \sum_{X_1 \cdots X_{i-1}} P(X_1, X_2, \cdots, X_L) \prod_{k=1}^{i-1-L} f(X_k, \cdots, X_{k+L}) \prod_{k=1}^{i-1} g(X_k, Y_k) \right) f(<B, A >)$$

s.t. $<X_{i-L}, X_{i-L+1}, \cdots, X_{i-1} > = <B, A_1, \cdots, A_{L-1} >$  

$| <X_1, X_2, \cdots, X_{i-1} > - <Y_1, Y_2, \cdots, Y_{i-1} > |_H = j$  

(B.1)

According to constraint (B.2), we know $B = X_{i-L}$ and $<A_1, \cdots, A_{L-1} > = <X_{i-L+1}, \cdots, X_{i-1} >$. Substitute these two terms in (B.1) we have

$$\sum_{X_1 \cdots X_{i-1}} P(X_1, X_2, \cdots, X_L) \prod_{k=1}^{i-1-L} f(X_k, \cdots, X_{k+L})$$

$$\prod_{k=1}^{i-1} g(X_k, Y_k) \cdot f(X_{i-L}, X_{i-L+1}, \cdots, X_{i-1}, A_L)$$

s.t. $<X_{i-L+1}, \cdots, X_{i-1} > = <A_1, \cdots, A_{L-1} >$  

$| <X_1, X_2, \cdots, X_{i-1} > - <Y_1, Y_2, \cdots, Y_{i-1} > |_H = j$  

(B.3)

We can combine term $f(X_{i-L}, \cdots, X_{i-1}, A_L)$ to product $\prod_{k=1}^{i-1-L} f(X_k, \cdots, X_{k+L})$ we have (B.5). At same time, adding the constraint $X_{i} = Y_i$, $A_{L} = Y_i$ we can also expand the Hamming distances constraint (B.4) to (B.8). At the same time $\prod_{k=1}^{i-1-L} g(X_k, Y_k)$ can also be expanded one element. Then we have
\[
\sum_{X_1 \cdots X_i} P(X_1, X_2, \cdots, X_L) \prod_{k=1}^{i-L} f(X_k, \cdots, X_{k+L}) \prod_{k=1}^{i} g(X_k, Y_k) \tag{B.5}
\]

s.t.  \[<X_{i-L+1}, \cdots, X_{i-1}> = <A_1, \cdots, A_{L-1}>\] \tag{B.6}

\[X_i = A_L\] \tag{B.7}

\[|<X_1, X_2, \cdots, X_{i-1}, Y_i> - <Y_1, Y_2, \cdots, Y_{i-1}, Y_i>|_H = j\] \tag{B.8}

\[A_L = Y_i\]

Combine the constraint (B.6) and (B.7), we have

\[<X_{i-L+1}, \cdots, X_i> = A\] \tag{B.9}

Put out \[A_L = Y_i\] into condition, (B.5) with constraint (B.9) is the definition of \(F_{i,j}(A)\).
\[
\sum_{b} F_{i-1}(b) \cdot f(b, a) \cdot g(Y_{2i-2}, a, Y_{2i-1}) \cdot \phi(Y_{2i-1}, a, Y_{2i})
\]

\[
= \sum_{b} \left( \sum_{X_1, \hdots, X_{i-1}} \prod_{k=1}^{i-2} f(X_k, X_{k+1}) \prod_{k=2}^{i-1} g(Y_{2k-2}, X_k, Y_{2k-1}) \phi(Y_{2k-1}, X_k, Y_{2k}) \right) \\
\cdot f(b, a) \cdot g(Y_{2i-2}, a, Y_{2i-1}) \cdot \phi(Y_{2i-1}, a, Y_{2i})
\]

s.t. \( X_{i-1} = b \)

\[
= \left( \sum_{X_1, \hdots, X_{i-1}} \prod_{k=1}^{i-2} f(X_k, X_{k+1}) \prod_{k=2}^{i-1} g(Y_{2k-2}, X_k, Y_{2k-1}) \phi(Y_{2k-1}, X_k, Y_{2k}) \right) \\
\cdot f(X_{i-1}, a) \cdot g(Y_{2i-2}, a, Y_{2i-1}) \cdot \phi(Y_{2i-1}, a, Y_{2i})
\]

\[
= \sum_{X_1, \hdots, X_{i-1}} \prod_{k=1}^{i-2} f(X_k, X_{k+1}) \cdot f(X_{i-1}, a) \\
\prod_{k=2}^{i-1} g(Y_{2k-2}, X_k, Y_{2k-1}) \cdot g(Y_{2i-2}, a, Y_{2i-1}) \phi(Y_{2i-1}, X_k, Y_{2k}) \phi(Y_{2i-1}, a, Y_{2i})
\]

\[
= \sum_{X_1, \hdots, X_i} \prod_{k=1}^{i-1} f(X_k, X_{k+1}) \prod_{k=2}^{i} g(Y_{2k-2}, X_k, Y_{2k-1}) \phi(Y_{2k-1}, X_k, Y_{2k})
\]

s.t. \( X_i = a \)

\[
= F_i(a)
\]
\[
\sum_{b,c} F_{i-1}(b) \cdot f(b, c) \cdot f(c, a) \cdot g(Y_{i-1}, c, a, Y_i)
\]
\[
= \sum_{b,c} \left( \sum_{X_1,\ldots,X_{2i-2}} \prod_{k=2}^{i-1} f(X_{2k-2}, X_{2k-1}) f(X_{2k-1}, X_{2k}) \prod_{k=2}^{i-1} g(Y_{k-1}, X_{2k-1}, X_{2k}, Y_k) \right)
\cdot f(b, c) \cdot f(c, a) \cdot g(Y_{i-1}, c, a, Y_i)
\]
s.t. \(X_{2i-2} = b\)
\[
= \left( \sum_{X_1,\ldots,X_{2i-2}} \prod_{k=2}^{i-1} f(X_{2k-2}, X_{2k-1}) f(X_{2k-1}, X_{2k}) \prod_{k=2}^{i-1} g(Y_{k-1}, X_{2k-1}, X_{2k}, Y_k) \right)
\cdot f(X_{2i-2}, X_{2i-1}) \cdot f(X_{2i-1}, a) \cdot g(Y_{i-1}, X_{2i-1}, a, Y_i)
\]
\[
= \sum_{X_1,\ldots,X_{2i-2}} \prod_{k=2}^{i-1} f(X_{2k-2}, X_{2k-1}) f(X_{2k-1}, X_{2k}) f(X_{2i-2}, X_{2i-1}) f(X_{2i-1}, a)
\prod_{k=2}^{i} g(Y_{k-1}, X_{2k-1}, X_{2k}, Y_k) g(Y_{i-1}, X_{2i-1}, c, Y_i)
\]
s.t. \(X_{2i} = a\)
\[
= F_i(a)
\]