ABSTRACT

Most existing security decisions for both defending and attacking are made based on some deterministic approaches that only give binary answers. Even though these approaches can achieve low false positive rate for decision making, they have high false negative rates due to the lack of accommodations to new attack methods and defense techniques. In this dissertation, I study how to discover and use patterns with uncertainty and randomness to counter security challenges. By extracting and modeling patterns in security events, I am able to handle previously unknown security events with quantified confidence, rather than simply making binary decisions. In particular, I cope with the following four real-world security challenges by modeling and analyzing with pattern-based approaches: 1) How to detect and attribute previously unknown shellcode? I propose instruction sequence abstraction that extracts coarse-grained patterns from an instruction sequence and use Markov-chain-based model and support vector machines to detect and attribute shellcode; 2) How to safely mitigate routing attacks in mobile ad hoc networks? I identify routing table change patterns caused by attacks, propose an extended Dempster-Shafer theory to measure the risk of such changes, and use a risk-aware response mechanism to mitigate routing attacks; 3) How to model, understand, and guess human-chosen picture passwords? I analyze collected human-chosen picture passwords, propose selection function that models patterns in password selection, and design two algorithms to optimize password guessing paths; and 4) How to identify influential figures and events in underground social networks? I analyze collected underground social network data, identify user interaction patterns, and propose a suite of measures for systematically discovering and mining adversarial evidence. By solving these four problems, I demonstrate that discovering and using patterns could help deal with challenges in computer security, network security, human-computer interaction security, and social network security.
Dedicated to my family
ACKNOWLEDGEMENTS

First, I want to thank my wife Yiqi Zhao, my parents Dejun Zhao and Youjun Yuan, my mother-in-law Rong Zhao, my cousins Ye Yuan and Xin Yuan. I am the person I am today because of them. Without the support, understanding, and encouragement from them, I could never finish this degree.

I am grateful to my PhD advisor Prof. Gail-Joon Ahn. Prof. Ahn supported me to pursue my own research interest professionally, financially, and emotionally. He is an extremely visionary researcher, and discussions with him have always been rewarding. He helped me with every step of the way to my dissertation and my degree. Prof. Ahn is a true gentleman who treated me as an equal since the first day we met and opened a lot of doors for me in both academia and industry.

I am grateful to my PhD committee members, Prof. Stephen S. Yau, Prof. Dijiang Huang, and Prof. Raghu Santanam. I have learned so much from the classes they taught and the questions they asked for my comprehensive exam. I appreciate the assistance and guidance they provided in preparation of my dissertation proposal and this dissertation. I am grateful to Prof. Partha Dasgupta, Prof. Guoliang Xue, and Prof. Rida Bazzi for their guidance and encouragement.

This dissertation would not be possible without the support of many people at SEFCOM. I would like to express my gratitude to my coauthors, collaborators, labmates and friends at SEFCOM, in particular, Prof. Hongxin Hu (Clemson University), Ruoyu Wu, Yiming Jing, Michael Mabey, Jeong-Jin Seo, Wonkyu Han, Mukund Sankaran, Deepinder Mahi, Dr. Dae-il Jang, Darin Tupper, Dominic Chen, Justin Paglierani, Sowmya Challa, Monika Szalamacha, James Holmes, Clinton D’souza, Jeremy Whitaker, Carlos Rubio, Hadi Sharifi, Michael Sanchez, Patrick Trang, Pradeep Sekar, Ketan Kulkarni, Prof. Tae-Sung Kim (Chungbuk National University), Prof.
Namje Park (Jeju National University), Prof. Juan Wang (Wuhan University), Yusef Shaban, and Joshua Frisby.

I would like to thank my colleagues in ASU but outside of SEFCOM. The discussions with them have helped my research tremendously. They are Dr. Xi Fang, Prof. Dejun Yang (Colorado School of Mines), Dr. Jin Zhang, Dr. Qiang Zhang, Prof. Rui Zhang (University of Hawaii), Dr. Lingjun Li, Xinxin Zhao, Xiang Zhang, Yashu Liu, Jun Shen, and Chuan Huang.

I thank my friends who are not mentioned above. They are Yadong Shen, Fengze Xie, Prof. Su Dong (Fayetteville State University), Wu Li, Xinhui Hu, Wei Huang, Tianyi Xing, Yuan Wang, Abdullah Alshalan, Jian Cai, Heng Chen, Meng Chen, Ke Bai, Jing Lu, Jing Huang, Che Liu, Jie Zhu, Xiaoping Li, Ruozhou Yu, Albion Hargrave, and Jo-Nell Hargrave. Thank you for making my PhD journey very enjoyable.
# TABLE OF CONTENTS

| LIST OF TABLES ................................................................. | x  |
| LIST OF FIGURES .............................................................. | xi |

## CHAPTER

1 INTRODUCTION ................................................................. 1
   1.1 Proposed Pattern-based Approach Framework ..................... 2
   1.2 Contributions of This Dissertation ............................. 5
   1.3 Previous Publications .................................................. 9

2 MATHEMATICAL PREREQUISITES ............................................ 11
   2.1 Probability Theory ...................................................... 11
   2.2 Dempster-Shafer Theory ................................................. 13
   2.3 Markov Models ............................................................ 15
   2.4 Support Vector Machines ............................................... 16

3 DETECTING AND ATTRIBUTING SHELLCODE ............................... 18
   3.1 Introduction ............................................................. 18
   3.2 Background ............................................................... 22
      3.2.1 Differentiating Code from Data ................................. 22
      3.2.2 IA-32 Instruction Format ....................................... 23
   3.3 Motivating Examples of Instruction Patterns .................... 24
   3.4 Overview of Extracting and Using Instruction Patterns ........ 27
   3.5 Automatic Extraction of Instruction Patterns .................. 28
      3.5.1 Customized Linear Sweep Disassembly ......................... 28
      3.5.2 Feature Selection ................................................... 29
      3.5.3 Instruction Sequence Abstraction ............................... 33
   3.6 Using Instruction Patterns to Detect and Attribute Shellcode ... 34
3.6.1 Detecting Shellcode ........................................ 35
3.6.2 Attributing Encoded Shellcode........................... 36
3.7 Implementation .............................................. 37
  3.7.1 Data Collection and Implementation .................... 37
  3.7.2 Instruction Patterns Identification .................... 39
3.8 Evaluation ................................................... 39
  3.8.1 Evaluation of Detecting Shellcode ...................... 39
  3.8.2 Evaluation of Attributing Encoded Shellcode .......... 42
  3.8.3 The Strength of Encoding Engines ..................... 47
  3.8.4 Performance Evaluation ................................ 50
3.9 Discussion .................................................. 50
  3.9.1 The Feasibility of Using One Model .................... 50
  3.9.2 Using Opcode Functionality Sequence to Detect Shellcode . 51
  3.9.3 The Arms Race and Future Work ....................... 52
3.10 Related Work .............................................. 52
3.11 Summary .................................................... 54

4 MITIGATING MANET ROUTING ATTACKS ...................... 56
4.1 Introduction ................................................ 56
4.2 Background .................................................. 58
  4.2.1 OLSR Protocol ........................................ 58
  4.2.2 Routing Attack on OLSR ............................... 60
4.3 Extended Dempster-Shafer Theory of Evidence ............. 61
  4.3.1 Importance Factors and Belief Function ............... 62
<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3.2</td>
<td>Expected Properties for Our Dempster’s Rule of Combination with Importance Factors</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Dempster’s Rule of Combination with Importance Factors</td>
</tr>
<tr>
<td>4.4</td>
<td>Overview of Risk-Aware Response Mitigation for MANET Routing Attacks</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Response to Routing Attacks</td>
</tr>
<tr>
<td>4.5</td>
<td>Risk Assessment Based on Routing Table Change Patterns</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Discovering Routing Table Change Patterns</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Combination of Evidence</td>
</tr>
<tr>
<td>4.6</td>
<td>Adaptive Decision Making</td>
</tr>
<tr>
<td>4.7</td>
<td>Case Study and Evaluation</td>
</tr>
<tr>
<td>4.7.1</td>
<td>Methodology and Metrics</td>
</tr>
<tr>
<td>4.7.2</td>
<td>Case Study</td>
</tr>
<tr>
<td>4.7.3</td>
<td>Evaluation with Random Network Topologies</td>
</tr>
<tr>
<td>4.8</td>
<td>Related Work</td>
</tr>
<tr>
<td>4.9</td>
<td>Summary</td>
</tr>
<tr>
<td>5</td>
<td>GUESSING PICTURE PASSWORDS</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>5.2</td>
<td>Background: Picture Gesture Authentication</td>
</tr>
<tr>
<td>5.3</td>
<td>User Choice Patterns in Picture Gesture Authentication Passwords</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Experiment Design</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Findings</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Memorability and Usability Analysis</td>
</tr>
<tr>
<td>5.4</td>
<td>Using User Choice Patterns to Attack PGA Passwords</td>
</tr>
<tr>
<td>CHAPTER</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Attack Models ............................. 101</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Password and PoI Representations .................. 102</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Location-dependent Gesture Selection Functions .......... 103</td>
</tr>
<tr>
<td>5.4.4</td>
<td>LdGSF Sequence List Generation and Ordering ............. 107</td>
</tr>
<tr>
<td>5.4.5</td>
<td>Password Dictionary Generation ..................... 111</td>
</tr>
<tr>
<td>5.5</td>
<td>Implementation .................................. 112</td>
</tr>
<tr>
<td>5.5.1</td>
<td>PoI Identification ................................ 112</td>
</tr>
<tr>
<td>5.5.2</td>
<td>LdGSF Identification ........................... 114</td>
</tr>
<tr>
<td>5.6</td>
<td>Attack Evaluation .............................. 116</td>
</tr>
<tr>
<td>5.6.1</td>
<td>Nontargeted Attack Evaluation ..................... 116</td>
</tr>
<tr>
<td>5.6.2</td>
<td>Targeted Attack Evaluation ....................... 126</td>
</tr>
<tr>
<td>5.7</td>
<td>Discussion .................................... 128</td>
</tr>
<tr>
<td>5.7.1</td>
<td>Picture-Password-Strength Meter .................... 128</td>
</tr>
<tr>
<td>5.7.2</td>
<td>Other Attacks on PGA ............................ 129</td>
</tr>
<tr>
<td>5.7.3</td>
<td>Limitations of Our Study ........................ 129</td>
</tr>
<tr>
<td>5.8</td>
<td>Related Work ................................... 130</td>
</tr>
<tr>
<td>5.9</td>
<td>Summary ....................................... 131</td>
</tr>
<tr>
<td>6</td>
<td>DISCOVERING UNDERGROUND INTELLIGENCE .......... 132</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction ................................... 132</td>
</tr>
<tr>
<td>6.2</td>
<td>Patterns in Cybercrime .......................... 135</td>
</tr>
<tr>
<td>6.3</td>
<td>Using Social Interaction Patterns to Discover Intelligence ... 138</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Online Underground Social Dynamics Model .............. 139</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Principles of Metric Design and Definitions .......... 141</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Ranking Metrics ............................... 144</td>
</tr>
</tbody>
</table>
6.4 System Design and Implementation ........................................... 147
  6.4.1 Challenges from Real-world Data ................................. 147
  6.4.2 System Architecture and Implementation ...................... 148
  6.4.3 Visualization Interfaces of Our Tool ............................ 150
6.5 Experiments on Synthetic Social Dynamics Data ................. 151
6.6 A Case Study on Real-world Online Underground Social Dynamics . 153
  6.6.1 Post, User and Group Analysis .................................... 154
  6.6.2 Evidence Mining by Correlating Social Dynamics with Ad-
   versarial Events .......................................................... 158
  6.6.3 Comparison with HITS Algorithm ................................ 162
6.7 Related Work .................................................................... 164
6.8 Summary ............................................................................ 165
7 CONCLUSION ................................................................. 166
  7.1 Contributions ................................................................. 166
  7.2 Future Work ................................................................. 168
REFERENCES ......................................................................... 169
<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Shellcode Dataset Comparison</td>
<td>36</td>
</tr>
<tr>
<td>3.2 Top 50 Opcode Transition Patterns</td>
<td>40</td>
</tr>
<tr>
<td>3.3 The Strength of Encoders</td>
<td>48</td>
</tr>
<tr>
<td>3.4 Shellcode Attribution Time Cost (millisecond)</td>
<td>49</td>
</tr>
<tr>
<td>4.1 Risk Assessment and Decision Making</td>
<td>80</td>
</tr>
<tr>
<td>5.1 Survey Question: Which of the following best describes what you are considering when you choose locations to perform gestures?</td>
<td>93</td>
</tr>
<tr>
<td>5.2 Attributes of Most Frequently Used PoIs</td>
<td>96</td>
</tr>
<tr>
<td>5.3 Numbers of Gesture-order Patterns</td>
<td>98</td>
</tr>
<tr>
<td>5.4 Numbers of Gesture Type Combinations and Average Time Spent on Creating Them</td>
<td>99</td>
</tr>
<tr>
<td>5.5 Top 10 Identified LdGSFs Using $P_{L-15}^2$</td>
<td>115</td>
</tr>
<tr>
<td>5.6 Top 10 Identified LdGSFs Using $P_{A-40}^2$</td>
<td>116</td>
</tr>
<tr>
<td>6.1 Top Five Influential/Active Users</td>
<td>155</td>
</tr>
<tr>
<td>6.2 Top Five Influential/Active Groups</td>
<td>155</td>
</tr>
<tr>
<td>6.3 Results from Our Tool for Queries</td>
<td>159</td>
</tr>
<tr>
<td>6.4 Selected Top Relevant Articles</td>
<td>160</td>
</tr>
<tr>
<td>6.5 Selected Articles by crx.ur and Her/His Information</td>
<td>161</td>
</tr>
<tr>
<td>6.6 Information about dx.ur</td>
<td>161</td>
</tr>
<tr>
<td>6.7 Selected Top Relevant Articles</td>
<td>163</td>
</tr>
<tr>
<td>6.8 Top Five Authorities and Hubs by HITS</td>
<td>163</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>The Security Cycle (a) Defenders’ Perspective (b) Attackers’ Perspective</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Proposed Pattern-based Approach Framework</td>
<td>4</td>
</tr>
<tr>
<td>3.1</td>
<td>IA-32 Architecture Instruction Format</td>
<td>24</td>
</tr>
<tr>
<td>3.2</td>
<td>Motivating Examples of Instruction Patterns</td>
<td>25</td>
</tr>
<tr>
<td>3.3</td>
<td>Overview of the Approach of Using Instruction Patterns to Detect and</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>Attribute Shellcode</td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>Instruction Sequence Abstraction Example</td>
<td>34</td>
</tr>
<tr>
<td>3.5</td>
<td>Encoded Shellcode Samples Collection</td>
<td>38</td>
</tr>
<tr>
<td>3.6</td>
<td>The S-score Distribution with Different k</td>
<td>41</td>
</tr>
<tr>
<td>3.7</td>
<td>Shellcode Radar Charts</td>
<td>43</td>
</tr>
<tr>
<td>3.8</td>
<td>Attribution Accuracy with Four Kernel Functions</td>
<td>45</td>
</tr>
<tr>
<td>3.9</td>
<td>Accuracy with Increasing Number of Shellcode Classes</td>
<td>47</td>
</tr>
<tr>
<td>4.1</td>
<td>Risk-Aware Response Mechanism</td>
<td>69</td>
</tr>
<tr>
<td>4.2</td>
<td>An Example of Link Spoofing Attack</td>
<td>70</td>
</tr>
<tr>
<td>4.3</td>
<td>Adaptive Decision Making</td>
<td>75</td>
</tr>
<tr>
<td>4.4</td>
<td>Routing Tables</td>
<td>76</td>
</tr>
<tr>
<td>4.5</td>
<td>Performance Results in Three Stages Comparing DRCIF with Binary Isolation and</td>
<td>79</td>
</tr>
<tr>
<td></td>
<td>DRC</td>
<td></td>
</tr>
<tr>
<td>4.6</td>
<td>Performance Results in Five Random Topologies Comparing DRCIF with Binary</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>Isolation and DRC</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>Background Pictures Used in Dataset-2</td>
<td>92</td>
</tr>
<tr>
<td>5.2</td>
<td>Two Versions of Starry Night and Corresponding Passwords</td>
<td>97</td>
</tr>
<tr>
<td>5.3</td>
<td>Memorability and Usability</td>
<td>100</td>
</tr>
</tbody>
</table>
5.4 (a) Background Picture and Password (b) User’s Selection Processes that Were Taken From Pace (2011b) (c) Corresponding LdGSFs that Simulate User’s Selection Processes .................. 104

5.5 (a) Background Picture and Identified PoIs (b) Identified PoIs (c) Password Representations (Colors are used to indicate the connections between the PoIs in (b) and LdGs in (c)) ...................... 105

5.6 PoI Identification on Example Pictures in Dataset-2: (a) Original Pictures (b) Circle Detection with Hough Transform (c) Contour Detection (d) Objectness Measure (e) Object Detection .................. 113

5.7 (a) Percentage of passwords cracked vs. number of password guesses, per condition. (b) Percentage of LdGs cracked vs. number of password guesses, per condition. For Dataset-1, there are 86 passwords that include 258 LdGs. For Dataset-2, there are 10,039 passwords that have 30,117 LdGs. ............................................. 117

5.8 (a) Percentage of passwords cracked vs. number of password guesses, per condition. (b) Percentage of LdGs cracked vs. number of password guesses, per condition. Only the first chosen password by each subject in Dataset-2 was considered. There are 762 passwords that have 2,286 LdGs. ..................................................... 119

5.9 (a) Percentage of passwords cracked vs. number of password guesses, per condition. (b) Percentage of LdGs cracked vs. number of password guesses, per condition. Only passwords for pictures 243, 1116, 2057, 4054, 6467, and 9899 were considered. There are 4,003 passwords that have 12,009 LdGs. ............................................. 120
5.10 Online attacks. (a) Number of passwords cracked within five guesses, per condition. (b) Number of LdGs cracked within five guesses, per condition. ................................................................. 122

5.11 (a) Average number of passwords cracked vs. different training data sizes. (b) Average number of LdGs cracked vs. different training data sizes. $P_{A-40}^2$ is used for this analysis. Average over 3 analyses, with one standard deviation shown. ................................................................. 123

5.12 (a) pictures with fewer PoIs (b) portraits (c) pictures with people in them (d) pictures with lots of PoIs. Unbiased algorithm on $P_{A-40}^2$ is used for this analysis. ................................................................. 125

5.13 Average runtime in seconds to order LdGSF sequences using BestCover and Unbiased. Average over 15 pictures in Dataset-2 with one standard deviation shown. ................................................................. 126

5.14 Offline attacks. There are 9,974 passwords from 697 accounts in this experiment. The average size of training data sets is around 13. (a) Percentage of passwords cracked vs. number of password guesses, per condition. (b) Passwords cracked per account. Each horizontal bar represents a condition. Regions within each bar show the fraction of accounts for which the indicated number of passwords were cracked. . . . . 127

6.1 Cybercrime Workflow ................................................................. 135

6.2 Motivating Example of Malware Distribution and Evolution in Social Dynamics (All screenshots taken from Holt (2008)) .................. 137

6.3 Online Underground Social Dynamics Model: Entities and Relationships139

6.4 SOCIALIMPACT: Systematic Ranking Indices ......................... 144
<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.5</td>
<td>149</td>
</tr>
<tr>
<td>6.6</td>
<td>150</td>
</tr>
<tr>
<td>6.7</td>
<td>152</td>
</tr>
<tr>
<td>6.8</td>
<td>153</td>
</tr>
<tr>
<td>6.9</td>
<td>153</td>
</tr>
<tr>
<td>6.10</td>
<td>156</td>
</tr>
<tr>
<td>6.11</td>
<td>157</td>
</tr>
</tbody>
</table>

6.10 Correlation Coefficient of \textit{UserActiveness} & \textit{UserInfluence} and \textit{GroupActiveness} & \textit{GroupInfluence}
Chapter 1

INTRODUCTION

Discovering and using patterns in data is the process of discovering *regularities* in either a manual or an automatic way and using discovered regularities as supporting *evidence* to assist decision making such as predicting the outcomes of available actions. Patterns are everywhere, and the history of discovering and using patterns has been successful long before the invention of computers. The most famous story about the discovery of patterns in all times may be how Issac Newton defined the law of universal gravitation which was inspired by the *observations* of falling apples. Unfortunately, not all patterns can be described by a deterministic equation such as the equation of the law of universal gravitation. Most patterns follow a *skewed probability distribution* and their representations need the mathematical foundations of multiple theories. After we entered the era of computers, algorithms have been designed to automate the process of both discovering and using patterns with randomness and uncertainty. The research fields to improve such algorithms are known as pattern recognition, data mining, and machine learning which have achieved successes in many domains, such as face recognition, voice control, and text search.

The typical workflow of discovering patterns starts from observing the behaviors of the targets, a process known as signal acquisition. After signals are collected, preprocessing and feature extraction take place to remove redundant data and highlight salient characteristics, respectively. Even though there exists some pure statistical ways to do preprocessing and feature extraction, both of them require some *domain knowledge* of the targets and their signals to obtain more reliable and explainable results. The next step, which requires domain knowledge as well, is to select and build
models to best fit the extracted features. The generated models are used for future predictions and decision making, aka using patterns. Through the process of pattern discovery and modeling, regularities are encoded as mathematical models which are ready for using.

Security researchers and practitioners have been using two types of approaches to discover and use patterns in security events to solve security problems for decades. The first type of approaches uses predetermined patterns. Commercially popular anti-virus systems and intrusion detection systems are using malware signatures, a predefined sequence of bits, to tell if the captured samples are known malwares. Even if such misuse-based systems could minimize the false positive rate of malware detection, they could not detect new malware variants that do not contain that predefined bit sequences; hence they introduce high false negative rate. The other type of approaches uses statistical patterns. The first instance may be the intrusion detection model Denning (1986) proposed in 1986 by Dorothy E. Denning who designed to monitor a system’s audit records and detect abnormal patterns of system usages with statistical models. However, pure statistical approaches overlook the root cause of the problems, and their results are usually inexplicable.

1.1 Proposed Pattern-based Approach Framework

In order to find out stages where discovering and using patterns might be applicable and beneficial in the security cycles, it is important to model the defense and attack processes from defenders’ and attackers’ perspectives. As shown in Figure 1.1(a), we model the security cycle of defenders as three stages for simplicity: protection, detection, and response. Protection is the stage where security policies are enforced. Detection is used to check if the protection mechanisms are breached and security policies are violated. When policy violations are confirmed, responses are
taken place to mitigate the threats. Obviously, patterns can be discovered and used in the detection stage, which is the practice adopted in Denning (1986). In Chapter 3, we show how to discover and use instruction patterns for shellcode detection and attribution. In Chapter 4, we show that routing table change patterns can be used to measure the risk of MANET attacks and guide response.

On the other hand, the security cycle of attackers consists of reconnaissance, planning, and attack as shown in Figure 1.1(b). Reconnaissance is the process that attackers gather information of potential targets and choose targets based on it. Planning is the stage where attack tactics and approaches are designed and prepared. The actual attacks are then carried out on the targets for attackers’ benefits. Since direct intelligence is not available in many cases, patterns in side channel information are used to infer target intelligence in reconnaissance Al-Saleh and Crandall (2011). In Chapter 5, we show that user-choice patterns in picture passwords can be used by attackers for planning. In Chapter 6, we show that how to use social interaction patterns in the underground dynamics for reconnaissance.

We propose a pattern-based approach framework as shown in Figure 1.2, whose goal is to handle previously unknown security events by considering their root causes, randomness and uncertainty and at the same time quantify the confidence of decision...
making. Different from pre-determined pattern-based approaches, our approach does not rely on fixed bit sequences, but considers randomness and uncertainty. Thus, our approach can cope with previously unknown security events. Different from statistical pattern-based approaches that leave the feature discovery to statistical methods, our pattern-based approaches focuses on using the domain knowledge in each security problem to discover regularities and tackle root causes directly. Therefore, the reasons for such regularities to occur are explainable and interpreted in a security-related way.

Our pattern-based approach framework has four major steps: i) signal acquisition, in which we collect samples from both benign and malicious sources. The instances of applying this step to solve different challenges can be found in Section 3.3, Section 4.5, Section 5.3, and Section 6.2; ii) feature extraction, in which we extract the features that are related to the root cause of each problem. The instances of applying this step to solve different challenges can be found in Section 3.5.3, Section 4.5.1, Section 5.3, and Section 6.3; iii) model building, in which we use the tools in our mathematical repository to describe the discovered security-related regularities. The instances of applying this step to solve different challenges can be found in Section 3.6, Section 4.3,
Section 5.4.3, and Section 6.3.1; and iv) decision making, in which we design weighted decision making to quantify our decision confidence. The instances of applying this step to solve different challenges can be found in Section 3.6, Section 4.6, Section 5.4.4, and Section 6.3.3.

1.2 Contributions of This Dissertation

This dissertation studies and addresses four security challenges: i) how to detect and attribute shellcode; ii) how to safely mitigate network routing attacks; iii) how to guess picture passwords; and iv) how to discover underground social intelligence. These four problems represent security challenges in four different domains: computer security, network security, human-computer interaction security, and human social network security. The common thread throughout the research on these different problems is applying the techniques of discovering and using patterns to model, analyze, and solve security-related issues. A brief introduction of each problem and the contributions from this dissertation are summarized as follows:

Among the list of 2011 CWE/SANS most widespread and dangerous software errors, fifteen out of twenty-five could lead to remote code injection and execution attacks Mitre (2011). Among all kinds of code injection attacks, binary code injection is the most destructive one. A successfully exploited system gives the direct control of its CPU, the binary code interpreter, to the adversaries that have all the privileges of the subverted processes One (1996). The malicious injected codes in this category is also named shellcode since they used to return a command shell to the attackers. For shellcode detection and attribution, we apply the proposed pattern-based approach framework and make the following contributions:

- (Signal Acquisition) We collect benign and malicious binary code samples and look into the instruction patterns shown in these sample.
• (Feature Extraction) We analyze the reason for such patterns and propose instruction sequence abstraction, a coarse-grained feature extraction method on the instruction sequence that reduces the size of input data dimension, removes confusing byte characteristics, and keeps distinguishable instruction features.

• (Model Building and Decision Making) We design a Markov-chain-based model to capture the observed sequential patterns for shellcode detection. Our detection approach is location-independent and length-independent, hence it supports on-line shellcode detection on suspicious data streams.

• (Model Building and Decision Making) We apply support vector machines to capture distributional patterns shown in the instruction sequence abstraction for understanding encoded shellcode attribution.

Mobile Ad hoc Networks (MANET) are utilized to set up wireless communication in improvised environments without a predefined infrastructure or centralized administration. The network topologies of MANET are frequently changed due to the unpredictable mobility of nodes. Furthermore, each mobile node in MANET plays a router role while transmitting data over the network. Hence, any compromised nodes under an adversary’s control could cause significant damage to the functionality and security of its network since the impact would propagate in performing routing tasks. For mobile ad hoc network routing attack mitigation, we apply the proposed pattern-based approach framework and make the following contributions:

• (Signal Acquisition) We look into the routing table change patterns when routing attacks happen in MANET environment.

• (Feature Extraction) We identify several categories of routing table change patterns and measure the impact of each pattern category.
• (Model Building) We propose an extended Dempster-Shafer evidence model with importance factors and articulate expected properties for Dempster’s rule of combination with importance factors. Our Dempster’s rule of combination with importance factors is nonassociative and weighted.

• (Decision Making) We propose an adaptive risk-aware response mechanism with the extended D-S evidence model, considering damages caused by both attacks and countermeasures. The adaptiveness of our mechanism allows us to systematically cope with MANET routing attacks.

Picture Gesture Authentication Johnson (2012) has been recently introduced as an alternative login experience to text-based password on such devices. In particular, the new Microsoft Windows 8™ operating system adopts such an alternative authentication to complement traditional text-based authentication. This new authentication mechanism hit the market with miscellaneous computing devices including personal computers and tablets Microsoft (2013). Consequently, it is imperative to examine and explore potential attacks on picture gesture authentication in such a prevalent operating system for further understanding user experiences and enhancing this commercially popular picture password system. For picture gesture password guessing, we apply the proposed pattern-based approach framework and make the following contributions:

• (Signal Acquisition) We compile two datasets of PGA usage from user studies and perform an empirical analysis on collected data to understand user choice in background picture, gesture location, gesture order, and gesture type.

• (Feature Extraction) We identify the most popular user-choice patterns in gesture location, gesture order, and gesture type.
• (Model Building) We introduce the concept of selection function that abstracts and models users’ selection processes when selecting their picture passwords. We demonstrate how selection functions can be automatically identified from training datasets.

• (Decision Making) We propose a novel attack framework that is able to generate ranked password dictionaries based on selection functions.

Existing research on net-centric attacks has focused on the detection of attack events on network side and the removal of rogue programs from client side. However, such approaches largely overlook the way on how attack tools and unwanted programs are developed and distributed. Recent studies in underground economy reveal that suspicious attackers heavily utilize online social networks to form special interest groups and distribute malicious code. Consequently, examining social dynamics, as a novel way to complement existing research efforts, is imperative to systematically identify attackers and tactically cope with net-centric threats. For underground social intelligence discovery, we apply the proposed pattern-based approach framework and make the following contributions:

• (Signal Acquisition) We study the role of underground social dynamics in the whole underground economy. We collect a dataset of user interactions of an underground social network.

• (Feature Extraction and Model Building) We study the user interactions in this dataset. And, we formulate an online underground social dynamics model considering both social relationships and user-generated contents.

• (Model Building and Decision Making) We propose a suite of measures to systematically quantify social impacts of individuals and groups along with their
online conversations which facilitate adversarial evidence acquisition and investigation.

1.3 Previous Publications

This dissertation incorporates materials from several of my previous conference papers, a journal paper and a book chapter. The concepts and techniques of using instruction patterns to detect and attribute shellcode in Chapter 3 were discussed in conference papers:


The ideas of using routing table change patterns to assess and mitigate routing attack risks in Chapter 4 were discussed in a conference and a journal paper:


The ideas of using user choice patterns to guess picture passwords in Chapter 5 were presented in a conference paper:

The ideas of using social interaction patterns to model underground social dynamics and predict future cyber attacks in Chapter 6 were discussed in two conference papers and a book chapter:


The rest of this dissertation is organized as follows. In Chapter 2, we overview the mathematical foundations of our research. In Chapter 3, we present how to detect and attribute shellcode using patterns extracted from instruction sequences. In Chapter 4, we present how to mitigate MANET routing attacks using evidence collected and combined from routing tables. In Chapter 5, we present how to guess picture gesture passwords by modeling the user-choices patterns exhibited in collected passwords. In Chapter 6, we present how to discover underground social intelligence using social patterns. Chapter 7 concludes this dissertation.
MATHEMATICAL PREREQUISITES

There are many theories to model and describe patterns with uncertainty and randomness. This chapter is not meant to be a comprehensive introduction or summary of all existing techniques of discovering and using patterns. For more comprehensive coverage of existing techniques, please refer to Bishop and Nasrabadi (2006); Tan et al. (2007); Theodoridis and Koutroumbas (2008). This chapter only introduces the mathematical foundations for the techniques we will use and we will develop in this dissertation.

2.1 Probability Theory

Probability theory uses the concept of random variable to describe a variable whose value is subject to variations due to chance. A random variable can take on a set of discrete or continuous values, each with an associated probability.

In this dissertation, we denote the probability that a random variable $X$ will take the value $x_i$ as $Pr(X = x_i)$ for discrete random variables. In the cases where two discrete random variables are involved, we use $Pr(X = x_i, Y = y_i)$ to denote the probability that $X$ will take the value $x_i$ and $Y$ will take the value $y_i$. Since this probability describes multiple random variables, it is called joint probability of $X$ and $Y$. We use $Pr(Y = y_i|X = x_i)$ to denote the conditional probability that $Y$ will take $y_i$ given $X$ takes $x_i$.

Suppose $X$ and $Y$ are two random variables, the sum rule states that

$$ Pr(X) = \sum_Y Pr(X, Y) $$

(2.1)
Suppose $X$ and $Y$ are two random variables, the product rule states that

$$Pr(X, Y) = Pr(Y|X)Pr(X)$$  \hspace{1cm} (2.2)

Since $Pr(X, Y) = Pr(Y, X)$, Bayes’ theorem states that

$$Pr(Y|X) = \frac{Pr(X|Y)Pr(Y)}{Pr(X)}$$  \hspace{1cm} (2.3)

Probability theory uses the concept of mathematical expectation to denote the value of a random variable one would expect to find if one could repeat the random variable process an infinite number of times and take the average of the values obtained. Obviously, the random variable could be some mathematical transformation from other random variable represented as $f(x)$. The expectation of $f(x)$ denoted as $\mathbb{E}[f]$ under a discrete probability distribution $Pr(X)$ is given by

$$\mathbb{E}[f(x)] = \sum_x Pr(x)f(x)$$  \hspace{1cm} (2.4)

The variance of $f(x)$ that provides a measure of how much variability there is in $f(x)$ around its expectation $\mathbb{E}[f]$ is given by

$$var[f(x)] = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2]$$  \hspace{1cm} (2.5)

For two random variables, the covariance that expresses the extent to which two variables vary together is defined by

$$cov[x, y] = \mathbb{E}_{x,y}[(x - \mathbb{E}[x])(y - \mathbb{E}[y])]$$  \hspace{1cm} (2.6)

If $X$ and $Y$ are said to be independent of each other,

$$Pr(X, Y) = Pr(X)Pr(Y)$$  \hspace{1cm} (2.7)
and

$$Pr(Y|X) = Pr(Y)$$  \hspace{1cm} (2.8)$$

When $X$ and $Y$ are independent to each other, their covariance $cov[X,Y]$ vanishes and is equal to 0. This means $X$ and $Y$ vary independently. When $X$ and $Y$ are not independent, we use Pearson’s product-moment coefficient, a normalized representation of covariance, to denote how related they are.

$$\rho_{x,y} = \frac{cov[x, y]}{\sqrt{var(x)}\sqrt{var(y)}} = \frac{\mathbb{E}_{x,y}[(x - \mathbb{E}[x])(y - \mathbb{E}[y])]}{\mathbb{E}[(x - \mathbb{E}[x])^2]\mathbb{E}[(y - \mathbb{E}[y])^2]}$$  \hspace{1cm} (2.9)$$

2.2 Dempster-Shafer Theory

The Dempster-Shafer theory begins with the familiar idea of using a number between zero and one to indicate the degree of support a body of evidence provides for a proposition. But unlike past attempts to develop this idea, the theory does not focus on the act of judgment by which such a number is determined. It focuses instead on something more amenable to mathematical analysis: the combination of degrees of belief or support based on one body of evidence with those based on an entirely distinct body of evidence. The heart of the theory is Dempster’s rule for effecting this combination.

Mathematically, Dempster’s rule is simply a rule for computing, from two or more belief functions over the same set $\Theta$, a new belief function called their orthogonal sum. The burden of this theory is that this rule corresponds to the pooling of evidence: if the belief functions being combined are based on entirely distinct bodies of evidence and the set $\Theta$ discerns the relevant interaction between those bodies of evidence, then the orthogonal sum gives degrees of belief that are appropriate on the basis of the combined evidence.
In Dempster-Shafer theory, propositions are represented as subsets of a given set. Suppose \( \Theta \) is a finite set, and let \( 2^\Theta \) denote the set of all subsets of \( \Theta \). \( \Theta \) will acquire its meaning from what we know or think we know. In order to emphasize this epistemic nature of the set of possibilities \( \Theta \), \( \Theta \) will be called frame of discernment. When a proposition corresponds to a subset of a frame of discernment, it is said that the frame discerns that proposition.

If \( \Theta \) is a frame of discernment, then a function \( m : 2^\Theta \to [0,1] \) is called a basic probability assignment whenever

\[
m(\emptyset) = 0 \quad (2.10)
\]

and

\[
\sum_{A \subseteq \Theta} m(A) = 1 \quad (2.11)
\]

The quantity \( m(A) \) is called \( A \)'s basic probability number, and it is understood to be the measure of the belief that is committed exactly to \( A \). Condition (1) reflects the fact that no belief ought to be committed to \( \emptyset \), while (2) reflects the convention that one's total belief has measure one. To obtain the measure of the total belief committed to \( A \), one must add to \( m(A) \) the quantities \( m(B) \) for all proper subsets \( B \) of \( A \).

A function \( Bel : 2^\Theta \to [0,1] \) is called a belief function over \( \Theta \) if it is given by (3) for some basic probability assignment \( m : 2^\Theta \to [0,1] \).

\[
Bel(A) = \sum_{B \subset A} m(B) \quad (2.12)
\]

Given several belief functions over the same frame of discernment but based on distinct bodies of evidence, Dempster’s rule of combination enables us to compute their orthogonal sum, a new belief function based on the combined evidence. Suppose \( Bel_1 \) and \( Bel_2 \) are belief functions over the same frame \( \Theta \), with basic probability
assignments $m_1$ and $m_2$. Then the function $m : 2^\Theta \rightarrow [0, 1]$ defined by $m(\emptyset) = 0$ and

$$m(A) = \frac{\sum_{A_i \cap B_j = A} m_1(A_i)m_2(B_j)}{1 - \sum_{A_i \cap B_j = \emptyset} m_1(A_i)m_2(B_j)} \quad (2.13)$$

for all non-empty $A \subset \Theta$ is a basic probability assignment.

### 2.3 Markov Models

Markov models are used to treat sequential data by assuming it has Markov property. Markov property states that the conditional probability distribution of future states of the process depends only the most recent states, not on the sequence of events that preceded them. Assume $x_1, ..., x_N$ are $N$ consecutive states of a process, $Pr(x_1, ..., x_N)$ represents the joint probability for a sequence of observations. According to the product rule, we have

$$Pr(x_1, ..., x_N) = \prod_{n=1}^N Pr(x_n|x_1, ..., x_{n-1}) \quad (2.14)$$

If we assume the process has first-order Markov property, which means only the most recent state contributes to the future state. We can rewrite the joint probability for a sequence of observations to

$$Pr(x_1, ..., x_N) = Pr(x_1)\prod_{n=2}^N Pr(x_n|x_{n-1}) \quad (2.15)$$

Transition matrix is a matrix used to describe the transitions of a Markov chain. Each of its entries is a nonnegative real number representing a probability. Taking first-order Markov model as an example, $Pr(i|j) = p_{i,j}$ is the probability of state
moving from $j$ to $i$. The transition matrix looks like

$$P = \begin{pmatrix}
p_{1,1} & p_{1,2} & \cdots & p_{1,j} & \cdots \\
p_{2,1} & p_{2,2} & \cdots & p_{2,j} & \cdots \\
\vdots & \vdots & \ddots & \vdots & \cdots \\
p_{i,1} & p_{i,2} & \cdots & p_{i,j} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \ddots
\end{pmatrix}$$  \hspace{1cm} (2.16)

for which $\Sigma_j p_{i,j} = 1$.

2.4 Support Vector Machines

Support vector machines are decision machines that became popular for solving problems in classification, regression, and detection. Support vector machine approaches the problem of classification through the concept of margin, which is defined as the smallest distance between the decision boundary and any of the samples. In support vector machines, the decision boundary, that is determined by model parameters, is chosen to be the one for which the margin is maximized. Therefore, determination of the model parameters is transformed to a convex optimization problem.

There are many support vector machine variants. Here we only discuss $C$-support vector classification. Given $l$ training vectors $\{x_i, y_i\}, \ i = 1, ..., l$ in two classes, where each vector is represented as $x_i$, and a class label $y_i$ with one of two values $\{-1|1\}$. The SVM requires the solution of the following optimization problem Boser et al. (1992); Cortes and Vapnik (1995); Chang and Lin (2011):
\[
\min_{\mathbf{w}, b, \xi} \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^{l} \xi_i
\]
Subject to \( y_i(\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \)
\[
\xi_i \geq 0
\] (2.17)

where feature vectors \( \mathbf{x}_i \) are mapped to higher dimensional space by a function \( \phi \) and \( C > 0 \) is the penalty parameter of the error term. In SVM, \( K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_i) \), called the kernel function, determines how the feature vector maps data to higher dimensional space.

We list some popular kernel functions here. The first is radial basis function (RBF). A RBF kernel takes the form of
\[
K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \| \mathbf{x}_i - \mathbf{x}_j \|^2), \quad \gamma > 0
\] (2.18)
Therefore, there are two parameters to tune: the penalty parameter \( C \) and \( \gamma \).

Linear function. A linear function takes the form of
\[
K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j
\] (2.19)
Therefore, the only parameter for tuning is \( C \).

Sigmoid function. A sigmoid function takes the form of
\[
K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^T \mathbf{x}_j + r), \quad \gamma > 0
\] (2.20)
Therefore, there are three parameters to tune: the penalty parameter \( C \), \( \gamma \) and \( r \).

Polynomial function. A polynomial function takes the form of
\[
K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \mathbf{x}_j + r)^d, \quad \gamma > 0
\] (2.21)
Therefore, there are four parameters to tune: the penalty parameter \( C \), \( \gamma \), \( r \), and \( d \).
3.1 Introduction

Malicious code injection is still an unsolved problem that threatens critical net-centric production systems. According to the recent report from SANS, fifteen out of twenty-five most widespread and dangerous software errors could lead to remote code injection and execution attacks Mitre (2011). Even though research efforts have been invested on such a devastating issue, production systems, where known vulnerabilities are not mitigated and potential vulnerabilities are introduced with newly-deployed modules, remain highly vulnerable to these threats.

The targets of code injection attacks could be both script and native language interpreters which depends on the type of the inserting code. HTML injection, cross-site scripting (XSS), SQL injection, and PHP inclusion fall into the first category where the corresponding language interpreter, such as browser, JavaScript engine, database management system and etc., is a software application. Therefore, code injection in this catalog is also called script injection, which could result in malicious manipulation of privileges and resources accessible within the environment, such as session record and account information Vogt et al. (2007).

Compared with script injection, binary code injection is the most destructive injection attack. A successfully exploited system gives the direct control of its CPU, the binary code interpreter, to the adversaries that have all the privileges of the subverted processes One (1996). If the underlying operating system or privileged processes are broken down, adversaries could take full control of the entire system.
which renders all the active services unreliable and distrust. The injected malicious binary code is also known as shellcode since they used to return a command shell to the attackers.

Misuse of informational data as executable code is the root cause of code injection attacks. Sanitization techniques that remove and escape reserved characters in the corresponding programming language are proven to be effective and efficient in scripting languages Weinberger et al. (2011). However, these techniques are less successful in defending against binary code injection attacks due to the fact that there is no special meaningful token in binary to differentiate code from data. For instance, in IA-32 instruction set, the only invalid byte to start an instruction is \texttt{0xF1} and all other 255 possible values could be interpreted as the starting byte of a valid encoding Chinchani and Van Den Berg (2006).

One research body in defeating malicious code injection is concerned about the prevention of its execution in an architectural way rather than the detection of its existence. Instruction set randomization Barrantes et al. (2003), data execution prevention, and address space layout randomization, and structural exceptional handler overwrite protection all fall into this category by disrupting the successful running of shellcode. Data execution prevention (DEP) is intended to prevent a process from executing code from non-executable memory regions, such as process stack and system heap Microsoft (2006). Address space layout randomization (ASLR) randomly arranges the locations of data sections such as position of libraries, stack and heap for each process and makes it difficult for injected code to execute itself Shacham et al. (2004). Even if these prevention solutions are effective, widely supported by modern hardware, and popularly adopted in state-of-the-art operating systems, they could only be triggered after the attack event occurred. There is no way for these techniques to predict and detect exploit attempts before it reaches the real attack
surface. The reason behind this shortcoming is that these solutions are not able to address the aforementioned root cause of code injection attacks. Although they could serve as the last line of code injection defense, it is imperative to have other defense techniques which are able to gather tactical intelligence for future shellcode detection and response.

Although signature-based methods remain the most effective ways to defeat known malware, they could be effective only when malicious samples have been acquired and signatures have been obtained. It is less useful against new samples and may be vulnerable to code encoding techniques, such as polymorphism and metamorphism. Several static analysis solutions Toth and Kruegel (2002a); Christodorescu et al. (2005); Chinchani and Van Den Berg (2006); Wang et al. (2006b) construct some distinguishable criteria, such as the length of the instruction sequence, to identify previously unknown malicious code. Since the heuristics in these methods are dependent on the existing knowledge of shellcode, they fail to accommodate the new trends of shellcode evolution. There also exist some discussions about utilizing both sequential and distributional byte patterns to model malicious code Wang et al. (2006a); Kolter and Maloof (2006). However, statistically modeling the byte patterns is vulnerable to deliberate byte cramming Detristan (2003) and blending attacks Fogla et al. (2006). Moreover, quantitative analysis of the byte strength of different shellcode engines presented in Song et al. (2007) concludes that modeling the byte patterns in shellcode is infeasible.

Besides all the investments on binary code detection, it is important to discover binary code attribution that automatically attributes binary code to its originating tools. Encoded shellcode attribution that tells whether the newly-captured shellcode sample is generated by a known shellcode engine provides security analysts and practitioners with more intelligence about the attack event and the adversaries behind the
scene than simplistic detection approaches. However, existing work on this topic Kong et al. (2011) that utilize byte characteristics still face the same challenges as binary code detection with byte anomaly analysis does.

Consequently, systematic techniques that can confront the root cause of code injection, detect unseen malicious attacks and attribute newly-detected malicious code samples are imperative to cope with emerging rogue code threats and gather tactical intelligence for a more comprehensive knowledge base on cyberattacks. These solutions should be resilient to known attacks, such as byte cramming and blending, that undermine existing techniques. To achieve these goals, in this chapter, we propose a novel solution based on static analysis and supervised machine learning techniques. Instead of using byte patterns, we propose instruction sequence abstraction to extract coarse-grained but distinguishable features from disassembled instruction sequence.

We design a Markov-chain-based model and apply support vector machines for unknown shellcode detection and classification. The rest of this chapter is organized as follows. Section 3.2 discusses the background. Section 3.3 discusses the observed motivating instruction patterns. Section 3.4 presents the overview of our approach to extract and use instruction patterns. Section 3.5 illustrates instruction sequence abstraction to automatically extract instruction patterns. In Section 3.6, we propose two learning models for shellcode detection and attribution. Section 3.7 discusses the implementation details and automated identified instruction patterns. We show our experimental results in Section 3.8 and discuss some research issues in Section 3.9. Section 3.10 overviews the related work and Section 3.11 concludes the chapter.
3.2 Background

In this section, we address the theoretical limit in differentiating code from data in IA-32 architecture. We then briefly describe the instruction encoding method of IA-32. We discuss why byte pattern analysis on this architecture is difficult.

3.2.1 Differentiating Code from Data

Accurate and robust methodology for differentiating code from data in a static way has a profound value for both programming language and computer security research. Disassembler and decompiler could use such technique to determine the boundary between code and data in file and memory for better understanding of programs without source code. Intrusion detection system could also make use of it to alert unseen cyberattack. For some computer architectures, this problem is trivial due to their instruction representations and memory alignments. However, some CISC architectures, such as IA-32, adopt variable-length instruction sets and permit interleaving of code and data. That is, instructions and data are stored together in the memory and have indistinguishable representations. The problem is even harder to solve when we consider self-modifying code and indirect jump (\texttt{jmp eax})—a control transfer approach whose destination can only be calculated at execution time—in binary. Therefore, statically separating code from data in such architectures may lead to the halting problem that is undecidable in general Cifuentes and Fraboulet (1997); Wartell \textit{et al.} (2011); Horspool and Marovac (1980); Landi (1992).

A major reason for the failure of byte pattern analysis on binary code is that even a slight change on assembly syntax would cause tremendous changes in byte syntax. Consequently, pattern detection and recognition on byte level is not resilient to assembly-level syntax change that is prevalent in shellcode encoding. For example,
different shellcode instances from the same metamorphic engine could use different
general registers to perform semantically equivalent operations. While one could use
two instructions \texttt{pop edx; mov ebx, [edx]} (byte sequence: \texttt{5A8B1A}) to move a value
from memory to register for computing, another may use \texttt{pop eax; mov esi, [eax]}
(byte sequence: \texttt{588B30}). Even though the instruction sequences look similar, the
byte sequences only share one single byte (8B). With more sophisticated encoding
technique, this shared byte could be further avoided.

3.2.2 IA-32 Instruction Format

We briefly discuss the instruction encoding scheme in IA-32 architecture that
serves as the cornerstone of our analysis. IA-32 instructions follow the encoding ap-
proach shown in Figure 3.1. Each instruction consists of up to 4 prefix bytes, up to
3 opcode bytes, and up to 10 operand bytes. The prefix bytes determine modifiers
of this instruction, such as \texttt{lock} (F0h) and \texttt{repeat} (F2h). The opcode bytes define
the operation of this instruction, such as \texttt{mov} or \texttt{add}. The operands part consists
of optional ModR/M (Address-Form specifier), SIB (Scale-Index-Base), displacement
and immediate that are used to specify the operand combinations. ModR/M byte
contains fields of information to specify the address form of following operands: If re-
quired, a SIB byte follows to detail the scale, index, and base fields of base-plus-index
and scale-plus-index addressing form. A displacement or an immediate follows if re-
quired. For instance, instruction \texttt{imul [ebp+esi+20], 616D2061} references a signed
integer multiplication operation. The addressing mode specifies a \texttt{base-plus-index}
form. A displacement 20 and immediate \texttt{616D2061} appear in this instruction as well.
3.3 Motivating Examples of Instruction Patterns

We describe several motivating patterns observed in both benign and malicious binary samples as shown in 3.2. Different from previous solutions that concentrate on byte patterns, we pay more attention on the instruction patterns shown in disassembly. We address instruction patterns of both sequences and distributions. For patterns in the sequence, we focus on the ones that are found in binary code but less likely to be found in text string, video files, or any other forms of data. For patterns of distributions, we concentrate on the ones that are found across different shellcode instances generated by the same encoder. Note that these patterns only serve as the motivation of our approach that is not dependent on any specific pattern mentioned in this section, but has the ability to extract human-observable and unobservable patterns in binary disassembly.

A push-call sequential pattern consists of several push instructions followed by a call instruction. In IA-32 architecture, the parameters to a function call are stored temporarily on stack usually by a push operation. Depending on the prototype of the function, it is obvious to observe several push instructions before a call instruction. Therefore, in a valid sequence of instructions, the subsequent instruction of push is more likely to be another push or call. SigFree Wang et al. (2006b) used the number of push-call in the instruction sequence as threshold to determine whether a package has code.

A function-prologue sequential pattern normally exists in bp-based function, a
procedure that uses ebp as the frame pointer to mark the bottom of stack for the called function and esp as the stack pointer to track the top of stack. Therefore, as the prologue of bp-based function, the frame pointer of the caller function is stored on stack by push ebp and then updated to the current frame pointer by mov ebp, esp.

Figure 3.2: Motivating Examples of Instruction Patterns
Frequently, an `add esp, immediate` is followed to reserve bytes for local variables in this pattern.

A *function epilogue pattern* exists in `_cdecl` calling conventions. In `_cdecl` calling conventions, it is the caller’s responsibility to clean the stack and maintain stack balance. Therefore, in this sample we observe that there are equal bytes of parameters pushed on stack before calling and popped from stack after calling. Also in IA-32, the return value of a function is normally stored in `EAX`, which is why `EAX` is access after function call.

A *group-of-arithmetic-shift* distributional pattern groups several bitwise and arithmetic instructions, such as `xor` (exclusive or) and `sub` (subtraction), to perform some sophisticated calculations. Instances of this pattern are mostly found in decryption routines that contain the inordinate number of such instructions to decode protected resources Caballero *et al.* (2009).

In a *test-jmp pattern*, there is a comparison or test instruction first, such as `cmp`, `test` which sets the corresponding bits of the `EFLAGS` register. Then, a conditional jump instruction, such as `ja` (jump if above), `jnz` (jump if not zero), is followed which transfers the control to different destinations based on the bit pattern set in `EFLAGS`.

A *getPC pattern* is widely adopted in shellcode Polychronakis *et al.* (2007). Since there is no way to predict the absolution address a shellcode would be located, shellcode always need methods to get its location in runtime. However, IA-32 does not provide any build-in method to retrieve the value of `EIP`, adversaries devise some techniques to get this value in runtime. In this example, `call 7h` pushes the absolute address of `pop esi` on stack, then `pop esi` stores this value to `esi` and restores stack balance at the same time.

A *alphabet pattern* is mostly found in shellcode generated by *English_shellcode* Mason *et al.* (2009) and *alpha_mixed_encoder* engines, whose outputs only consist of dis-
Figure 3.3: Overview of the Approach of Using Instruction Patterns to Detect and Attribute Shellcode

playable ASCII values, such as English letters, blank space and punctuation marks. Because of the narrow choices these engines have, generated shellcode have significantly more unconditional jumps, stack and arithmetic operations than samples generated by other encoders.

However, there are several obstacles in using these instruction observations directly in statistical models. For example, the trained model may overfit instruction patterns shown in the training set. Adversaries and encoding engines could easily choose other semantically-equivalent instructions to replace existing ones which render the modeling of any specific instruction ineffective. In order to solve these challenges, we propose instruction sequence abstraction, a coarse-grained feature extraction method, to tackle the problem of overfitting by mapping high-dimensional byte sequence representations to low-dimensional instances.

3.4 Overview of Extracting and Using Instruction Patterns

Our approach is based on static analysis and supervised machine learning as shown in Figure 3.3. Our solution consists of three major components: i) input processor, that is presented in Section 3.5, preprocesses binary code or suspicious data with customized linear disassembly that outputs the instruction sequence; ii) feature extrac-
tor, that is presented in Section 3.5, reveals distinguishable features from the instruction sequence and outputs its two corresponding data representations named opcode mnemonic sequence and binary finite-dimensional representation; and iii) trainer, that is presented in Section 3.6, utilizes Markov-chain-based model and support vector machines to train detection and classification models based on training set and classifiers that use trained models to detect and classify suspicious data.

3.5 Automatic Extraction of Instruction Patterns

In this section, we propose customized linear sweep disassembly algorithm. To remove any confusion between byte and instruction characteristics that hinder previous research efforts, we then present our feature selection and instruction sequence abstraction to extract representative characteristics from the instruction sequence.

3.5.1 Customized Linear Sweep Disassembly

The first step of our approach is to disassemble binary, a method that extracts semantics out of binary code and outputs machine-understandable disassembly. There exist two basic disassembly algorithms: i) linear sweep disassembly and ii) recursive descent disassembly. Linear sweep disassembly decodes new instruction at the location where the previous one ends. If the current byte could not be decoded as a valid starting byte of the instruction, linear sweep disassembly stops. The disadvantage of linear sweep disassembly is that it may mistakenly disassemble data as code if the starting location is wrong. Recursive descent disassembly determines whether a location should be decoded with the references by other instructions. In other words, recursive descent disassembly follows the control flow of the instruction sequence. Recursive descent disassembly stops when it could not determine the location of the next
instruction, such as when an indirect jump is encountered. The major disadvantage of recursive descent disassembly is that it cannot cover the entire code section.

In our solution, we need the most comprehensive coverage of disassembly. Therefore, we modify linear sweep disassembly and propose a customized linear sweep disassembly $\text{CLSD} : (b_1, ..., b_n) \rightarrow (i_1, ..., i_m)$ as shown in Algorithm 1. Unlike linear sweep disassembly, $\text{CLSD}$ does not stop when it reaches an undecodable address. Instead, it moves to the next address and perform linear sweep until it reaches the end of file. The complexity of this algorithm is $O(n)$. Although $\text{CLSD}$ may mistakenly decode some data section, such as encrypted payload in polymorphic shellcode, and incorrectly disassemble some instructions with a wrong starting location, this algorithm would disassemble the major portion of code over the data stream with the help of the self-repairing ability of IA-32 instruction set Linn and Debray (2003).

3.5.2 Feature Selection

In this section, we present our coarse-grained feature extraction method to reveal representative features from the instruction sequence $(i_1, ..., i_m)$ generated from $\text{CLSD}$. We try to introduce as many features as possible to reduce the possibility that the learned model is over-fitting the training dataset. We inspect both opcode and operand of an instruction as the sources for features. The opcode part of an instruction reveals the functionality of the disassembly statement, while operand part tells which object the effect is enforced on. We design opcode features based on two aspects: functionality and origin. Functionality captures the basic behavior and effect of a given opcode, and origin describes the source of instruction set that a given opcode was first introduced. Although the operand part of an instruction includes several fields such as addressing form and immediate, for simplicity, we only analyze the usage of eight general purpose registers in an instruction as representative char-
Algorithm 1: CLSD

**Input:** Byte sequence $b = (b_1, ..., b_n)$ with $n$ bytes

**Output:** Instruction sequence $i = (i_1, ..., i_m)$

1. Initially $i$ is empty;
2. Set $p = 1$;
3. while $p$ is not greater than $n$ do
   4. if $b_p$ is a valid instruction starting byte then
      5. Do linear sweep disassembly on $b_p$;
      6. When its stops, append its output $(i_1, ..., i_k)$ to $i$;
      7. Set $p$;
   8. else
      9. Set $p = p + 1$;
10. end
4. end
12. return the instruction sequence $i$

acteristics for operand features. We also use the length of instruction as a feature that represents the instruction in general.

*Opcode functionality:* We categorize each opcode into one of the following ten groups in terms of its functionality:

1. *Arithmetic.* Opcodes that provide arithmetic operations, such as addition, multiplication, and some miscellaneous conversion instructions. Examples include `add`, `adc`, `sub`, `sbb`, `mul`, and `imul`.

2. *Shift, rotate and logical.* Opcodes that provide shift, rotate and logical operations, such as bitwise and left shift with the carry-over. Examples include `and`, `or`, `ror`, and `xor`. 
3. **Unconditional data transfer.** Operations that move data among memory locations and registers without querying flag register. Examples include `mov`, `in`, and `out`;

4. **Conditional data transfer.** Operations that move data among memory locations and registers based on the status indicated in a flag register. Examples include `seta` and `setnl`;

5. **Processor control.** Opcodes that manipulate the status of processor by modifying flag, loading and saving system registers, and synchronizing external devices. Examples include `arpl`, `hlt`, and `lgdt`;

6. **Stack operation.** Opcodes that manipulate a program stack. Examples include `push`, `pop`, `enter`, and `leave`;

7. **Unconditional program transfer.** Opcodes that change the program counter register without querying flag register. Examples include `call`, `int`, `jmp`, and `ret`;

8. **Conditional program transfer.** Opcodes that make transfer decisions based on specific bit combinations in flag register. Examples include `ja`, `jne`, and `loopw`;

9. **Test and compare.** Opcodes that compare the values of operands and store the result in some predefined register. Examples include `test`, `cmp`, and `scas`;

10. **Other operation.** Opcodes that are not included in the aforementioned categories.

**Opcode origin:** We also categorize each opcode into one of the following six instruction sets:
1. 8086 set, a group of instructions that were introduced with 8086 family CPUs Intel (1979);

2. 80286, 80386, and 80486 sets. We combine these three instruction sets together because there do not exist many instances in each of these instruction sets;

3. Pentium and Pentium II sets;

4. 80387 and MMX, instruction sets that control and manipulate floating point coprocessors and MMX processors;

5. Pentium III and Pentium IV; and

6. Other sets.

General register usage: For each instruction $i$ in the instruction sequence, we analyze whether any of the eight general registers or any part of them is explicitly used in this instruction. These eight general registers are: 1) eax; 2) ebx; 3) ecx; 4) edx; 5) esi; 6) edi; 7) ebp; and 8) esp. For instance, in an instruction pop eax, the only explicitly mentioned general register is eax. We do not count the usage of esp because it is not explicitly mentioned in the operand part. For an instruction mov esi, [eax], both esi and eax appear in this statement. For an instruction add al, ch, we count one occurrence for both eax and ecx in this statement for the reason that al is part of eax and ch is part of ecx.

Length of instruction: For each instruction $i$ in the instruction sequence, we calculate its length. This feature is necessary because even instructions with the same opcode may vary in length. We split instructions into eight categories: Instructions are categorized into the first seven categories by using the length as an identifier if their lengths are not greater than seven bytes and instructions with longer than seven bytes fall into the eighth category. For example, mov ebp, esp (8BEC) is 2-byte long.
and classified in category ‘two’, and `mov dword ptr [eax+ebp+14h], 0CCCCCCCCCh (C7442814CCCCCCCC)` is 8-byte long and falls into category ‘eight’.

### 3.5.3 Instruction Sequence Abstraction

We now present *instruction sequence abstraction* which includes two representation methods to model instruction sequence: opcode mnemonic sequence (OMS) and binary finite-dimensional representation (BFR). Since both methods map *n*-byte data sequence into much lower dimensional space as coarse-grained feature extraction approaches, they are abstractions of the original byte and the instruction sequence. While OMS maps instances in $256^n$ byte sequence space to their counterparts in $a^m$ space ($a$ is the number of mnemonics in IA-32 instruction set and $m$ is the length of the instruction sequence), BFR represents instances in $\mathbb{Z}^{32}$ space. For the mathematical notations, we use lower case bold roman letters such as $f$ to denote vectors, subscript such as $f_i$ to denote the $i$-th component of $f$, superscript such as $f(i)$ to denote the $i$-th sample in dataset, and $f_{ij}(i)$ to denote the $i$-th sample’s $j$-th component. We assume all vectors to be column vectors and a superscript T to denote the transposition of a matrix or vector.

**Definition 3.1. Opcode Mnemonic Sequence (OMS).** For a given instruction sequence $i = (i_1, ..., i_m)$, its opcode mnemonic sequence is represented as $o^T = (o_1, ..., o_m)$, where $o_k \in \{\text{aaa, aad, ...}\}$, which is the valid opcode mnemonic set of IA-32 architecture.

**Definition 3.2. Binary Finite-dimensional Representation (BFR).** For a given instruction sequence $i = (i_1, ..., i_m)$, its binary finite-dimensional representation is a 32-dimensional vector $f^T = (f_1, ..., f_{32})$, where $f_i$, $i \in \{1, ..., 10\}$, is the number of instructions in the $i$-th opcode functionality category, $f_i$, $i \in \{11, ..., 16\}$, is the num-
55    push ebp
8BEC  mov ebp, esp
8B7608 mov esi, dword ptr [ebp+08]
85F6  test esi, esi
743B  je 401045
8B7E0C mov edi, dword ptr [ebp+0c]
09FF  or edi, edi
7434  je 401045
31D2  xor edx, edx

(a) Byte sequence   (b) Instruction sequence

\[ \mathbf{\alpha}^T = \text{(push, mov, mov, test, je, mov, or, je, xor)} \]

(c) OMS

\[ \mathbf{f}^T = (0, 2, 3, 0, 0, 1, 0, 2, 1, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \]

(d) BFR

**Figure 3.4:** Instruction Sequence Abstraction Example

ber of instructions from the corresponding opcode origins, \( f_i \), \( i \in \{17, ..., 24\} \), is the occurrence of corresponding general register, and \( f_i \), \( i \in \{25, ..., 32\} \), is the number of instructions with the corresponding length. Figure 3.4(b) shows the CLSD output of the byte sequence shown in Figure 3.4(a), and Figures 3.4(c) and 3.4(d) show the OMS and BFR representations of the example.

3.6 Using Instruction Patterns to Detect and Attribute Shellcode

In this section, we articulate our approach to detect and attribute suspicious byte sequences. We propose two classifiers for shellcode detection and attribution, respectively.
3.6.1 Detecting Shellcode

Based on the observation that certain instruction sequences are more likely to exist in some binary code rather than others, we propose to use the existence possibility of opcode mnemonic sequence to identify whether the suspicious byte stream contains shellcode. We choose first-order Markov chain, which is a discrete random process, to model the opcode mnemonic sequence by assigning each opcode mnemonic as a Markov state and computing the transition matrix of this Markov chain.

Like other supervised machine learning techniques, our method has training and evaluation phases. Given OMSs of $l$ shellcode training samples $\{o^{(i)}\}, i = \{1, ..., l\}$, a transition matrix $P \in \mathbb{R}^{a \times a}$ can be trained, with the $(i, j)$-th element of $P$ implies $p_{ij} = Pr(o_{k+1} = j|o_k = i)$ indicating the probability of opcode state transition from $o_k$ to $o_{k+1}$. In the evaluation phase, we calculate shellcode probability score ($S$-score) of suspicious data stream, which is defined in Definition 3 and determine whether it contains shellcode based on a threshold value $t$, which is also learned from the training set.

**Definition 3.3. Shellcode Probability Score (S-score).** Given the transition matrix $P$ trained from shellcode dataset and a suspicious OMS $o^T = (o_1, ..., o_m)$, the S-score of this OMS is defined as follows:

$$S\text{-}score(o) = \sqrt[\text{k}]{\max_{i=1, ..., m-k} \prod_{j=i}^{i+k} Pr(o_{j+1}|o_j)}$$  \hspace{1cm} (3.1)

where $k$ is the length of calculation window. To calculate the $S$-score of $o$, a value for each of $m - k$ opcode mnemonic subsequences with $k$-length is computed as the multiplications of the transition probabilities. Then, the maximum value among all $m - k$ opcode mnemonic subsequences is chosen whose $k$-th root is defined as the
Table 3.1: Shellcode Dataset Comparison

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>styx</td>
</tr>
<tr>
<td># of shellcode engines</td>
<td>2</td>
</tr>
<tr>
<td># of shellcode samples</td>
<td>2</td>
</tr>
</tbody>
</table>

*S-score* of o. If \( S\text{-score}(o) > t \), we say the byte sequence where o is generated from is a shellcode and *vice versa*.

Our shellcode detection approach is length-independent and location-independent on the byte sequence for two reasons: i) CLSD outputs the most comprehensive coverage of code, hence it has the ability to disassemble most shellcode bytes in a package, no matter where it is started; and ii) only the subsequent \( k \) instructions are used to calculate the *S-score*, hence the length of the byte sequence is not vital. Therefore, our approach is able to monitor on-line data stream, where the length and location of interesting points are unknown.

### 3.6.2 Attributing Encoded Shellcode

We propose to use support vector machines (SVM) to attribute encoded shellcode’s BFR to its originating encoding engine. SVM maps feature vectors into a higher dimensional space and computes a hyperplane to separate instances from different groups by maximizing the margin between them. Therefore, SVM is the largest margin classifier. The problem of attributing shellcode to its originating engine is a multi-class classification problem. However, the basic SVM only supports binary classification problems. Therefore, we use algorithms that supports a one-vs-all approach to extend SVM for classifying multi-class problems. Here, we only discuss how to use SVM for the binary classification problem that checks whether a shellcode sample is from a specific engine \( e \).
Given \( l \) shellcode training samples \( \{\mathbf{f}^{(i)}, y^{(i)}\}, \ i = 1, \ldots, l \), where each sample is denoted by its BFR that has 32 features, represented as \( \mathbf{f}^{(i)} \), and a class label \( y_i \) with one of two values \{-1, 1\}. -1 means it is not generated by an engine \( e \), while 1 confirms a specific engine. The SVM requires the solution of the following optimization problem Boser et al. (1992); Cortes and Vapnik (1995); Chang and Lin (2011):

\[
\min_{w,b,\xi} \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^{l} \xi_i \\
\text{Subject to } y_i (\mathbf{w}^T \phi(\mathbf{f}^{(i)}) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0
\]

(3.2)

where feature vectors \( \mathbf{f}^{(i)} \) are mapped to higher dimensional space by a function \( \phi \) and \( C \) is the penalty parameter of the error term. In SVM, \( K(\mathbf{f}^{(i)}, \mathbf{f}^{(j)}) = \phi(\mathbf{f}^{(i)})^T \phi(\mathbf{f}^{(i)}) \), called the kernel function, determines how the feature vector maps data to higher dimensional space. It can be noted that the kernel function maps feature vectors into higher dimensional space only to search for a separate hyperplane so it does not rebuild more complicated features. Hence, it does not conflict with our approach to abstract features from the instruction sequence.

### 3.7 Implementation

We first discuss the data collection and implementation details of our proof-of-concept system. Then, we present the automatically identified sequential instruction patterns in shellcode.

#### 3.7.1 Data Collection and Implementation

We utilized Metasploit Moore (2009)–a penetration framework that hosts exploits and tools from a variety of sources–to collect shellcode samples. We collected 140
**Figure 3.5:** Encoded Shellcode Samples Collection

*unencoded shellcode* samples, all of which are executable on IA-32 architecture, across different operating system platforms including Windows (85), Unix (11), Linux (21), FreeBSD (10), OSX (12), and Solaris (1). These samples are used to train our Markov-chain-based model and test the effectiveness of our approach. For the evaluation of *encoded shellcode* attribution, we chose 21 different payloads that are targeted at Windows, then used 14 different engines to encode these payloads (see Table 3.3 for the full list of engines). We tried to generate 50 unique shellcode instances for each pair of payload and encoder. Even though some payloads are not compatible with specific encoders, we successfully collected 13,176 encoded shellcode samples for attribution analysis. Compared with existing research bodies in both shellcode detection and attribution Chinchani and Van Den Berg (2006); Wang et al. (2006b); Song et al. (2007); Wang et al. (2008); Kong et al. (2011), our shellcode dataset covers a more comprehensive set of samples in term of underlying platform, payload functionality, and encoder class.

We implemented the customized linear sweep disassembly algorithm and feature extractor as an IDA Pro Rescue (2006) plug-in that outputs the OMS and BFR in separate files for each byte sequence input. We also developed the shellcode detection
3.7.2 Instruction Patterns Identification

In the learning phase, we trained our Markov model with aforementioned shellcode samples to generate the transition matrix of opcode. The top 50 highest opcode transition probabilities are shown in Table 3.2. Some transition patterns, such as $Pr(\text{xor}|\text{aaa})= 1.00$, $Pr(\text{push}|\text{push})= 0.61$, $Pr(\text{jz}|\text{test})= 0.57$, and $Pr(\text{jnz}|\text{cmp})= 0.56$, can be human-observable as we mentioned in Section 2.3 but other patterns cannot be easily identified. The results show that our approach is able to extract underlying and implicit machine code characteristics.

3.8 Evaluation

We show shellcode detection and attribution results followed by an analysis approach for measuring the strength of 14 popular shellcode encoding engines. We conclude our evaluation with the performance of our system.

3.8.1 Evaluation of Detecting Shellcode

In the detection phase, $S$-score uses a calculation window $k$ to compute the shellcode probability of the given input. A threshold value $t$ is also used to determine if a given byte sequence is executable or not. To find out the appropriate length of calculation window and threshold value, we tested 140 shellcode samples, 1,280 random data samples, 250 gif files, 250 png files, and 660 benign code pieces (we split ntoskrnl.exe which is the kernel image of Windows NT into 660 pieces). For a given sample, its $S$-score decreases as the calculation window increases, because $Pr(o_{j+1}|o_j)$ is always less than or equal to 1. We calculated the $S$-score of all of the collected
Table 3.2: Top 50 Opcode Transition Patterns

<table>
<thead>
<tr>
<th>Top 1 - 17</th>
<th>Top 18 - 34</th>
<th>Top 35 - 50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pr(xor</td>
<td>aaa) 1.00</td>
<td>Pr(adc</td>
</tr>
<tr>
<td>Pr(push</td>
<td>aam) 1.00</td>
<td>Pr(cm</td>
</tr>
<tr>
<td>Pr(sal</td>
<td>cwde) 1.00</td>
<td>Pr(cm</td>
</tr>
<tr>
<td>Pr(std</td>
<td>clc) 1.00</td>
<td>Pr(in</td>
</tr>
<tr>
<td>Pr(sbb</td>
<td>cm</td>
<td>fistp) 1.00</td>
</tr>
<tr>
<td>Pr(std</td>
<td>clid) 1.00</td>
<td>Pr(fstenv</td>
</tr>
<tr>
<td>Pr(std</td>
<td>idiv) 1.00</td>
<td>Pr(xor</td>
</tr>
<tr>
<td>Pr(dec</td>
<td>jecxz) 1.00</td>
<td>Pr(add</td>
</tr>
<tr>
<td>Pr(add</td>
<td>jg) 1.00</td>
<td>Pr(sub</td>
</tr>
<tr>
<td>Pr(outs</td>
<td>jge) 1.00</td>
<td>Pr(xor</td>
</tr>
<tr>
<td>Pr(outs</td>
<td>jle) 1.00</td>
<td>Pr(push</td>
</tr>
<tr>
<td>Pr(cli</td>
<td>loope) 1.00</td>
<td>Pr(push</td>
</tr>
<tr>
<td>Pr(push</td>
<td>mul) 1.00</td>
<td>Pr(call</td>
</tr>
<tr>
<td>Pr(loope</td>
<td>neg) 1.00</td>
<td>Pr(mov</td>
</tr>
<tr>
<td>Pr(push</td>
<td>or) 1.00</td>
<td>Pr(add</td>
</tr>
<tr>
<td>Pr(cl</td>
<td>sgdt) 1.00</td>
<td>Pr(push</td>
</tr>
<tr>
<td>Pr(cmc</td>
<td>fcomip) 1.00</td>
<td>Pr(push</td>
</tr>
</tbody>
</table>

samples with the length of calculation windows from 8 to 40 to find the appropriate value.

Figure 3.6 presents the S-score distribution of each sample category with three different values of the calculation window length. Figure 3.6(a) shows that if the length of calculation window \( k \) is set to 8, all shellcodes’ S-score is greater than 0.1, and a significant portion of shellcode have S-score greater than 0.3. However, only a small number of random data samples have S-score greater than 0 as shown in Figure 3.6(d). Figures 3.6(g),(j) show the S-score distribution of gif and png files,
Figure 3.6: The S-score Distribution with Different $k$
where only a small portion of samples have $S$-score greater than 0.1. By comparing Figure 3.6(a) and Figure 3.6(m), it is clear that benign code samples have much lower $S$-score than shellcode. Figure 3.6(b) shows that if the length of calculation window $k$ is set to 20, the $S$-score of every shellcode reduces. But, most shellcode samples still have $S$-score greater than 0.1. However, as shown in Figure 3.6(c), if the length of calculation window is set to 28, 3 shellcode samples have $S$-score with ‘null’. On the other hand, the $S$-score of all other samples are reduced close to 0 if $k$ is greater than 20, as shown in Figure 3.6. The results suggest that the combination of calculation window length $k = 28$ and threshold $t = 0.1$ be sufficient to identify shellcode with 97.9% accuracy and 0.82% false positive rate in our dataset.

### 3.8.2 Evaluation of Attributing Encoded Shellcode

We evaluate our shellcode representation and attribution analysis technique from several different aspects including visualization, correlation analysis of selected features, accuracy of attributing, and quantification of encoder strength.

**Data Visualization**

The visualization of shellcode samples could tell us the differences of shellcode generated from various engines in an intuitive way. We propose to visualize shellcode sample in BFR form with a radar chart graph, in which a circle is equally divided by 32 invisible lines. Each of these 32 lines represents an axis for each corresponding feature in BFR form, where the center of circle represents 0 and the periphery represents 1. Since, in the BFR form, a shellcode sample is represented as $f \in \mathbb{Z}^{32}$, we used data scaling on all the samples in our dataset to transform each feature into the range of $[0, 1]$. The value of each feature is marked by a dot on its corresponding line.
Figure 3.7: Shellcode Radar Charts

Then, the dots from each pair of consequent features are connected together and the contained area is marked black.

Figure 3.7 shows six radar charts of instances from six different shellcode engines. As we can notice, the shellcode generated by alpha_mixed has strong similarity with the shellcode generated by alpha_upper. They both generate shellcode with longer instructions (features 25 to 32). Shellcode generated by context_time or fnstenv_mov has more unconditional data transfer instructions than others (feature 3). fnstenv_mov tends to use registers esi, edi and ebp more often (features 21, 22, and 23), while alpha_mixed prefers eax, ecx, and edx (feature 17, 19, and 20). Because the data scaling is performed over the whole dataset, we could also notice that the size of black area differs significantly. The major reason behind this phenomenon is that some engines tend to generate much longer data sequence even if the input payload is
the same. Obviously, the shellcode generated by \texttt{call\_dword\_xor} or \texttt{avoid\_utf8\_tolower} is smaller than its counterpart generated by \texttt{alpha\_mixed} in size.

**Effectiveness of Feature Selection**

In order to prove our feature selection approach in BFR is effective and the extracted features are not redundant, we utilize *Pearson product-moment correlation coefficient* to measure the linear relationships between each pair of features in BFR form. Given two features $f_i$ and $f_j$ in BFR, the correlation coefficient $\rho_{f_i,f_j}$ is a measure of the linear dependency between them that is defined as $\rho_{f_i,f_j} = \frac{E[(f_i-\mu_{f_i})(f_j-\mu_{f_j})]}{\sigma_{f_i}\sigma_{f_j}}$, where $\mu_{f_i}$ is the mean and $\sigma_{f_i}$ is the standard deviation of this feature value over our dataset.

The maximum value for correlation coefficient, which is 1, represents a perfect positive correlation between two variables, and the minimum value -1 indicates a perfect negative correlation. If $|\rho_{f_i,f_j}|$ is close to 1, it means one of the selected features could be linearly represented by another one, hence it clearly indicates redundancy.

We calculated the correlation coefficient for each pair of features over our dataset, and computed the average of their absolute values defined as $P = \frac{\sum_{i \neq j} |\rho_{f_i,f_j}|}{496}$, where 496 is the number of feature pairs $(32 \times 31)/2$. The result was $P = 0.3428$ that indicates our feature selection is not redundant.

**Parameter Tuning**

In the SVM model, the factors that affect the classification result include the penalty parameter $C$, the kernel function, and corresponding parameters in the kernel function. We randomly divided our shellcode dataset into a training set and a testing set, which is a standard approach in machine learning. The training set has 60% of samples from each class and the testing set consists of the rest of samples, 40% of samples. To find the best kernel function and parameters, we used a grid-search for
possible parameter combinations on the training set to learn SVM model with four popular kernel functions Chang and Lin (2011) and to evaluate it on the testing set.

Radial basis function (RBF): A RBF kernel takes the form of $K(f^{(i)}, f^{(j)}) = \exp(-\gamma \| f^{(i)} - f^{(j)} \|^2)$, $\gamma > 0$. Therefore, there are two parameters to tune: the penalty parameter $C$ and $\gamma$. We used a grid-search to test exponentially growing sequence of $C = 2^{-3}, 2^{-2}, ..., 2^8$ and $\gamma = 2^{-7}, 2^{-6}, ..., 2^5$. Figure 3.8(a) shows the accuracy of testing when different parameter combinations are used. The results show that, when $C$ is fixed, the best $r$ is in the range $[2^{-4}, ..., 2^{-2}]$. On the other hand, when $r$ is fixed, the best $C$ is above 8. We found the best $(C, \gamma)$ combination $(8, 0.25)$ with the accuracy of 85.02% in attributing testing samples to its class;

Linear function: A linear function takes the form of $K(f^{(i)}, f^{(j)}) = f^{(i)T}f^{(j)}$. There-
fore, the only parameter for tuning is $C$. We tested $C = 2^{-3}, ..., 2^5$ and found the best penalty parameter $C = 8$ with 82.11% accuracy as shown in Figure 3.8(b);

**Sigmoid function**: A sigmoid function takes the form of $K(f^{(i)}, f^{(j)}) = \tanh(\gamma f^{(i)} T f^{(j)} + r)$, $\gamma > 0$. We found the best $(C, \gamma, r)$ combination $(2^{-4}, 2^{-5}, -2^{-3})$ with the accuracy of 63.70% in attributing testing samples to its class as shown in Figure 3.8(c);

**Polynomial function**: A polynomial function takes the form of $K(f^{(i)}, f^{(j)}) = (\gamma f^{(i)} T f^{(j)} + r)^d$, $\gamma > 0$. We evaluated the combinations of $C = 2^{-1}, ..., 2^3$, $\gamma = 2^{-5}, ..., 2^{-1}$, $r = 2^{-3}, ..., 2^{-1}$ and $d = 2, 3, 4$. We found the best $(C, \gamma, r, d)$ combination $(4, 0.25, 0.125, 3)$ with the accuracy of 84.57% as shown in Figure 3.8(d). In summary, our results suggest that RBF, linear, and polynomial kernels be appropriate for attributing shellcode samples in terms of accuracy. However, the computation cost for each kernel is different. We discuss the system performance using different kernels in Section 3.8.4.

**The Hardness of Multi-class Attributing**

We tested a radius basis function—the kernel function with the highest accuracy—with parameter combination $C = 8, \gamma = 0.25$ in subsets of our dataset to find out whether increasing the number of shellcode engines for the classification makes the problem harder to solve. We performed the same testing procedure mentioned in the previous section to test the accuracy of our model for 2, ..., 13 shellcode classes. Our model can achieve 100% classification accuracy for up to 6 shellcode engines and 95.0% classification accuracy for 11 classes, which is higher than previous efforts Kong *et al.* (2011). Note that, compared with Kong *et al.* (2011) in which a specific model is built for each shellcode class, our approach only use one model to classify instances.
Figure 3.9: Accuracy with Increasing Number of Shellcode Classes

from all kinds of classes, hence does not need different parameter settings for each model.

3.8.3 The Strength of Encoding Engines

In Song et al. (2007), the authors introduced variation strength, propagation strength and overall strength on the byte sequence of shellcode to measure polymorphic engines’ strength. We redefine these measures to accommodate our binary representation form.

Variation strength: The variation strength of an encoding shellcode engine measures the engine’s ability to generate shellcodes that span a sufficiently large portion of 32-dimensional BFR space. We make use of covariance matrix to recover the hyper-ellipsoidal bound on the dataset of each engine. The matrix is defined as

\[ \Sigma(e) = \frac{1}{N} \sum_{i=1}^{N} (f^{(i)} - \mu)(f^{(i)} - \mu)^T, \]

where \( N \) is the number of samples generated by an engine \( e \) in our dataset. \( \Sigma \in \mathbb{R}^{32 \times 32} \) describes the shape of a 32-dimensional ellipsoid. Then, the problem of calculating the spanned set is transformed to an eigenvector decomposition problem. Thus, \( \mathbf{v} \) and \( \lambda \), such as \( \Sigma \mathbf{v} = \mathbf{v} \lambda \), are recovered where \( \lambda \) is a 32-dimensional vector. We define \( \Psi(e) = \frac{1}{32} \sum_{i=1}^{32} \sqrt{|\lambda_i|} \) as the variation strength of an encoder \( e \);

Euclidean distance: Given two BFRs \( f^{(i)} \) and \( f^{(j)} \) which represent two samples in our dataset, the Euclidean distance between these
Table 3.3: The Strength of Encoders

<table>
<thead>
<tr>
<th>Engine</th>
<th>Variation Strength</th>
<th>Propagation Strength</th>
<th>Overall Strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha_mixed</td>
<td>1.91</td>
<td>26.07</td>
<td>49.91</td>
</tr>
<tr>
<td>alpha_upper</td>
<td>1.42</td>
<td>22.50</td>
<td>31.86</td>
</tr>
<tr>
<td>avoid_utf8_tolower</td>
<td>1.29</td>
<td>14.53</td>
<td>18.70</td>
</tr>
<tr>
<td>call4_dword_xor</td>
<td>2.17</td>
<td>25.82</td>
<td>55.96</td>
</tr>
<tr>
<td>context_cpuid</td>
<td>0.27</td>
<td>4.66</td>
<td>1.25</td>
</tr>
<tr>
<td>context_stat</td>
<td>0.93</td>
<td>12.53</td>
<td>11.65</td>
</tr>
<tr>
<td>context_time</td>
<td>0.94</td>
<td>12.51</td>
<td>11.73</td>
</tr>
<tr>
<td>countdown</td>
<td>0.80</td>
<td>10.42</td>
<td>8.33</td>
</tr>
<tr>
<td>fnstenv_mov</td>
<td>2.29</td>
<td>26.11</td>
<td>59.71</td>
</tr>
<tr>
<td>jmp_call_additive</td>
<td>1.46</td>
<td>15.74</td>
<td>22.91</td>
</tr>
<tr>
<td>nonalpha</td>
<td>0.74</td>
<td>9.91</td>
<td>7.36</td>
</tr>
<tr>
<td>nonupper</td>
<td>0.98</td>
<td>11.93</td>
<td>11.71</td>
</tr>
<tr>
<td>shikata_ga_nai</td>
<td>1.58</td>
<td>16.78</td>
<td>26.46</td>
</tr>
<tr>
<td>single_static_bit</td>
<td>1.22</td>
<td>15.87</td>
<td>19.34</td>
</tr>
<tr>
<td>random_data_generator</td>
<td>3.31</td>
<td>49.07</td>
<td>162.59</td>
</tr>
</tbody>
</table>

BFRs is defined as $\delta(f^{(i)}, f^{(j)}) = \sqrt{\sum_{k=1}^{32} (f^{(i)}_k - f^{(j)}_k)^2}$; *Propagation strength*: Given $N$ samples labeled as outputs of an engine $e$, the propagation strength of this engine describes the average Euclidean distance between all sample pairs defined as $\Phi(e) = \frac{2}{N(N-1)} \sum_{i\neq j} \delta(f^{(i)}, f^{(j)})$; *Overall strength*: The overall strength of an encoder $e$ is defined as the multiplication of its variation strength and propagation strength $\Pi(e) = \Phi(e) \times \Psi(e)$. The higher overall strength of an engine indicates that its shellcode instances are more obscured and harder to be correctly attributed.

In order to remove the differences introduced by different payloads, we only took
Table 3.4: Shellcode Attribution Time Cost (millisecond)

<table>
<thead>
<tr>
<th>Kernel Function</th>
<th>Parameter</th>
<th>Training(^1)</th>
<th>Classification(^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Combination</td>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td>Radial Basis function</td>
<td>(C = 8, \gamma = 0.25)</td>
<td>2,760</td>
<td>1,720</td>
</tr>
<tr>
<td>Linear function</td>
<td>(C = 8)</td>
<td>1,840</td>
<td>1,320</td>
</tr>
<tr>
<td>Sigmoid function</td>
<td>(C = 2^{-4}, \gamma = 2^{-5})</td>
<td>15,640</td>
<td>7,240</td>
</tr>
<tr>
<td></td>
<td>(r = -2^{-3})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Polynomial function</td>
<td>(C = 8, \gamma = 0.25)</td>
<td>3,120</td>
<td>1,230</td>
</tr>
<tr>
<td></td>
<td>(r = 0.125, d = 3)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^1\) Training set includes 7,906 shellcode samples

\(^2\) Testing set includes 5,270 shellcode samples

the shellcode instances generated by different engines from the same payload into account. Table 3.3 shows the strength of these engines based on our metrics. *random_data_generator* refers to a generator that outputs a group of randomly generated strings with the value of each byte in \(\{0, ..., 255\}\). The lengths of these strings are also randomly generated with the value from 160 to 400 bytes, which is the length range of shellcode we mostly observed. It is not surprising to discover that *random_data_generator* is the strongest ‘encoder’ with the overall strength of 162.59.

Among all the encoders, we also noticed that *fnstenv_mov* and *call4_dword_xor* are two of strongest engines based on our metrics, while *context_crypt* is the weakest one. *alpha_mixed* (49.91) is stronger than *alpha_upper* (31.86), because it could output both upper and lower case alphabets. However, the strength is not doubled because the size of output character set is doubled. Similar observation can be found between *nonalpha* and *nonupper* where *nonupper* shows a little bit stronger obfuscation.
3.8.4 Performance Evaluation

We conducted experiments on a machine with Intel Core2 Duo CPU 3.16 GHz 3.25 GB RAM running Windows 7, IDA Pro 5.6 and Matlab R2010a. We used Windows API GetTickCount to measure the performance of our program in C language and cputime to measure the elapsed time in Matlab program. The training phase of Markov-chain-based model only took less than 15 milliseconds to learn from 140 shellcode samples. The detection phase with the calculation window length of 20 took less than a second to calculate the $S$-score of 1,200 data streams with variable-length from 160 bytes to 400 bytes.

Table 3.4 shows the time cost for shellcode attribution in training and testing with different kernel functions. We evaluated the performance of the parameter combination with the accuracy of each kind of kernel function. Linear function is the most efficient kernel with 1,840 milliseconds in training for 7,906 samples and 1,320 milliseconds in classifying 5,270 samples. Radial basis function that has 85.02% accuracy in classifying 14 shellcode classes is also efficient, taking 2,760 milliseconds in training and 1,720 milliseconds in classification.

3.9 Discussion

In this section, we discuss other possible ways to model, detect, and attribute shellcode samples.

3.9.1 The Feasibility of Using One Model

It is possible to use one unified model to detect and attribute encoded shellcode in a single step. However, we choose not to adopt such an approach due to the following issues: i) the problem of differentiating code from data and the problem of
attributing detected attack are two separate issues. By separating these two research
issues, we could achieve the most accurate results for each group. However, we have
to balance the detection rate and attribution rate if these two problems are mixed
together; ii) we only consider sequential information to detect and attribute shellcode
and we use Markov-chain-based model to fulfill this requirement. In the training
phase, we need to train a specific sequential model for each shellcode class instead
of modeling all shellcode samples together. Correspondingly, in the detection phase,
the given suspicious data stream has to be evaluated by all trained models. With the
increased number of shellcode class, the evaluation process will be slow and infeasible
for on-line detection; and iii) we also consider a standard classifier, such as SVM and
neural networks, to perform detection and attribution. Most standard classifiers do
not support modeling of sequential knowledge that may render valuable ‘ordering’
information useless.

3.9.2 Using Opcode Functionality Sequence to Detect Shellcode

In our experimental evaluation, we also considered to use opcode functionality
sequence to detect shellcode instead of opcode mnemonic sequence. For a given
instruction sequence \( i = (i_1, ..., i_m) \), its opcode functionality sequence is represented
as \( s^T = (s_1, ..., s_m) \), where \( s_k \in \{1, ..., 10\} \) that is the set of the opcode functionality
category. While opcode mnemonic sequence maps encoded shellcode instances into a
\( a^m \) dimensional space, opcode functionality sequence maps them into an even lower
dimensional space, \( 10^m \). However, the evaluation results show that both false negative
and false positive rate are high with this representation. We believe the reason is that
the \( 10^m \) dimensional space is not sufficient to capture the difference between code and
data.
3.9.3 The Arms Race and Future Work

It is possible for malicious code distributors to disturb our shellcode detection method by permutating the locations of instructions in a sequence. For instance, the sequence 1: \texttt{mov eax, 1}; 2: \texttt{add eax, 1}; 3: \texttt{mov ebx, 1}; 4: \texttt{add ebx, 1} has different transition probabilities from the sequence 3; 1; 2; 4. However, their permutation choices are limited in which the semantics of the instruction sequence has to be maintained. For example, permutation to 2; 4; 3; 1 is not possible. To cope with this potential arms race, we could integrate machine code slicing Cifuentes and Fraboulet (1997) into our approach. The previous example could be sliced into two independent pieces 1; 2 and 3; 4, and only intra-piece transition probabilities are considered for learning and detection. In addition, higher order Markov chain may be utilized to enhance the accuracy of our approach but might need to minimize unexpected performance overhead. For the attribution part, attackers may deliberately cram garbage instructions to interfere the distribution patterns used in BFR. Fortunately, this issue is easier to solve than byte cramming attacks with the awareness of semantics in disassembly. We could perform data flow analysis Wang et al. (2006b) on machine code first to prune useless instructions in a sequence to handle this challenge.

3.10 Related Work

On the signature generation side, Newsome et al. Newsome and Song (2005) proposed TaintCheck that performs dynamic taint analysis by implementing binary rewrite at run time. Newsome et al. introduced Polygraph Newsome et al. (2005), a mechanism that is robust to generate signatures for polymorphic code. Li et al. Li et al. (2006) proposed Hamsa, a noise-tolerant and attack-resilient network-based automated signature generation system for polymorphic worms. Approaches to gen-
erate vulnerability-based signatures Brumley et al. (2006); Li et al. (2007) were also proposed on the network level without any host-level analysis of execution. However, Chung et al. Chung and Mok (2007) showed that all of these signature generation schemes are vulnerable to advanced allergy attacks.

Several emulation and execution-based approaches were proposed to detect exploit, shellcode, and worm. Polychronakis et al. Polychronakis et al. (2007) proposed a heuristic detection method to execute and monitor suspicious data stream captured in network traffic. However, setting up such a runtime environment that fits every exploit and shellcode is extremely difficult. To solve this problem, Gu et al. Gu et al. (2010) proposed to dump the process’s virtual memory before any input data are consumed and use the dumped image to instantiate a runtime environment that emulates the target process’s input data consumption to monitor shellcode behaviors. In addition, Spector Borders et al. (2007) used symbolic execution to reveal high-level application programming interface calls from shellcode for a better understanding of what it does.

APE Toth and Kruegel (2002a) calculated the maximum execution length of a byte sequence, and learned threshold for detecting possible malicious packages. Stride Akritidis et al. (2005) complemented the previous effort by adding new criteria including non-privileged instruction in a byte sequence to identify sled in shellcode. Chinchani et al. Chinchani and Van Den Berg (2006) and Kruegel et al. Kruegel et al. (2006) proposed to utilize the control and data flow information in binary to detect polymorphic code. Wang et al. Wang et al. (2006b) first used data flow anomaly to prune useless instructions then compared the number of useful instructions with a certain threshold to determine if it has any code.

Wang et al. Wang et al. (2006a) compared the byte frequency of normal network packages with malicious ones to figure out the byte patterns that could lead to
attack detection. However, their solutions were vulnerable to byte cramming Detristan (2003) and polymorphic blending attacks Fogla et al. (2006). Recently, Kong et al. Kong et al. (2011) proposed to take advantage of semantic analysis and sequential models on $n$-gram data bytes to analyze the attribution of exploits. Wartell et al. Wartell et al. (2011) developed machine learning-based algorithms to differentiate code from data. Rosenblum et al. Rosenblum et al. (2010) proposed to use conditional random field to extract compiler provenance from code. While most of these work focus on the byte patterns identified in binary code, Song et al. Song et al. (2007) presented quantitative analysis of the byte strength of polymorphic shellcode and claimed that modeling the byte patterns in shellcode is infeasible.

Besides the aforementioned research efforts, several new binary encoding schemes were proposed in recent years. Mason et al. Mason et al. (2009) proposed English shellcode engine that transforms arbitrary shellcode to a representation that is similar to English prose. Wu et al. Wu et al. (2010) proposed mimimorphism to transform binary into mimicry counterpart that exhibits high similarity to benign programs in terms of statistical properties and semantic characteristics.

3.11 Summary

In this chapter, we proposed a technique for modeling shellcode detection and attribution through a novel feature extraction method, called instruction sequence abstraction, that extracts distinguishable features from suspicious data stream by reducing the size of input data dimension and removing ambiguous byte patterns. We also presented a Markov-chain-based model for shellcode detection and adopted support vector machines for shellcode attribution. Our experiments showed that our approach does not require any signature and is only based on static analysis and supervised machine learning. The evaluation results also suggested that our solution
detect and attribute shellcode to its originating engines with high accuracy and lower false positive rate.
Chapter 4

MITIGATING MANET ROUTING ATTACKS

4.1 Introduction

Mobile Ad hoc Networks (MANET) are utilized to set up wireless communication in improvised environments without a predefined infrastructure or centralized administration. Therefore, MANET has been normally deployed in adverse and hostile environments where central authority point is not necessary. Another unique characteristic of MANET is the dynamic nature of its network topology which would be frequently changed due to the unpredictable mobility of nodes. Furthermore, each mobile node in MANET plays a router role while transmitting data over the network. Hence, any compromised nodes under an adversary’s control could cause significant damage to the functionality and security of its network since the impact would propagate in performing routing tasks.

Several work Sun et al. (2006b); Refaei et al. (2010) addressed the intrusion response actions in MANET by isolating uncooperative nodes based on the node reputation derived from their behaviors. Such a simple response against malicious nodes often neglects possible negative side effects involved with the response actions. In MANET scenario, improper countermeasures may cause the unexpected network partition, bringing additional damages to the network infrastructure. To address the above-mentioned critical issues, more flexible and adaptive response should be investigated.

The notion of risk can be adopted to support more adaptive responses to routing attacks in MANET Cheng et al. (2007). However, risk assessment is still a non-
trivial, challenging problem due to its involvements of subjective knowledge, objective evidence and logical reasoning. Subjective knowledge could be retrieved from previous experience and objective evidence could be obtained from observation while logical reasoning requires a formal foundation. Wang et al. Wang et al. (2007) proposed a naïve fuzzy cost-sensitive intrusion response solution for MANET. Their cost model took subjective knowledge and objective evidence into account but omitted a seamless combination of two properties with logical reasoning. In this chapter, we seek a way to bridge this gap by using Dempster-Shafer mathematical theory of evidence (D-S theory), which offers an alternative to traditional probability theory for representing uncertainty Shafer (1976).

D-S theory has been adopted as a valuable tool for evaluating reliability and security in information systems and by other engineering fields Sun et al. (2006a); Mu et al. (2008), where precise measurement is impossible to obtain or expert elicitation is required. D-S theory has several characteristics. First, it enables us to represent both subjective and objective evidence with basic probability assignment and belief function. Second, it supports Dempster’s rule of combination to combine several evidence together with probable reasoning. However, as identified in Sentz and Ferson (2002); Zadeh (1984); Yager (1987); Wu et al. (2002); Zhao et al. (2010, 2012c), Dempster’s rule of combination has several limitations, such as treating evidence equally without differentiating each evidence and considering priorities among them. To address these limitations in MANET intrusion response scenario, we introduce a new Dempster’s rule of combination with a notion of importance factors in D-S evidence model.

In this chapter, we propose a risk-aware response mechanism to systematically cope with routing attacks in MANET, and propose an adaptive time-wise isolation method. Our risk-aware approach is based on the extended D-S evidence model.
In order to evaluate our mechanism, we perform a series of simulated experiments with a proactive MANET routing protocol, Optimized Link State Routing Protocol (OLSR) Clausen and Jacquet (2003). In addition, we attempt to demonstrate the effectiveness of our solution.

The rest of this chapter is organized as follows. Section 4.2 overviews a MANET routing protocol OLSR and routing attacks against OLSR. Section 4.3 describes how our extended D-S evidence model can integrate importance factors. Section 4.4 overviews our risk-aware response mechanism to cope with MANET routing attacks. Section 4.5 illustrates how to assess the risk by leveraging routing table change evidence. Section 4.6 illustrates how to adaptively make risk mitigation decisions. The evaluations of our approach are discussed in Section 4.7. Section 4.8 provides the related work in MANET intrusion detection and response systems, also reviews risk-aware approaches in different fields. Section 4.9 concludes this paper.

4.2 Background

In this section, we overview the OLSR and routing attacks on OLSR.

4.2.1 OLSR Protocol

The major task of the routing protocol is to discover the topology to ensure that each node can acquire a recent map of the network to construct routes to its destinations. Several efficient routing protocols have been proposed for MANET. These protocols generally fall into one of two major categories: reactive routing protocols and proactive routing protocols. In reactive routing protocols, such as Ad hoc On Demand Distance Vector (AODV) protocol Perkins et al. (2003), nodes find routes only when they must send data to the destination node whose route is unknown. In contrast, in proactive routing protocols, such as OLSR, nodes obtain routes by periodic
exchange of topology information with other nodes and maintain route information all the time.

OLSR protocol is a variation of the pure Link-state Routing (LSR) protocol and is designed specifically for MANET. OLSR protocol achieves optimization over LSR through the use of multipoint relay (MPR) to provide an efficient flooding mechanism by reducing the number of transmissions required. Unlike LSR, where every node declares its links and forwards messages for their neighbors, only nodes selected as MPR nodes are responsible for advertising, as well as forwarding an MPR selector list advertised by other MPRs.

In OLSR, a node selects its MPR set that can reach all its two-hop neighbors. In case there are multiple choices, the minimum set is selected as an MPR set. A node learns about its one-hop and two-hop neighbors from its one-hop neighbors’ HELLO messages. HELLO message is used for neighbor discovery and MPR selection. In OLSR, each node generates a HELLO message periodically. A node’s HELLO message contains its own address and the list of its one-hop neighbors. By exchanging HELLO messages, each node can learn a complete topology up to two hops. HELLO messages are exchanged locally by neighbor nodes and are not forwarded further to other nodes. In addition to HELLO message, Topology Control (TC) message is used for route calculation. In OLSR, each MPR node advertises TC messages periodically. A TC message contains the list of the sender’s MPR selector. Only MPR nodes are responsible for forwarding TC messages. Upon receiving TC messages from all of the MPR nodes, each node can learn the network topology and then build a route to every node in the network.
4.2.2 Routing Attack on OLSR

Based on the behavior of attackers, attacks against MANET can be classified into passive or active attacks. Attacks can be further categorized as either outsider or insider attacks. With respect to the target, attacks could be also divided into data packet or routing packet attacks. In routing packet attacks, attackers could not only prevent existing paths from being used, but also spoof non-existing paths to lure data packets to them. Several studies Deng et al. (2002); Hu and Perrig (2004); Kannhavong et al. (2007); Karlof and Wagner (2003) have been carried out on modeling MANET routing attacks. Typical routing attacks include black-hole, fabrication, and modification of various fields in routing packets (route request message, route reply message, route error message, etc.). All these attacks could lead to serious network dysfunctions.

In terms of attack vectors, a malicious node can disrupt the routing mechanism in the following simple ways: first, it changes the contents of a discovered route, modifies a route reply message and causes the packet to be dropped as an invalid packet; then it validates the route cache in other nodes by advertising incorrect paths, and refuses to participate in the route discovery process; and finally, it modifies the contents of a data packet or the route via which the data packet is supposed to travel or behave normally during the route discovery process but is dropped.

In OLSR, any node can either modify the protocol messages before forwarding them, or create false messages or spoof an identity. Therefore, the attacker can abuse the properties of the selection algorithm to be selected as MPR. The worst case is the possible selection of the attacker as the only MPR of a node. Or, the attackers can give wrong information about the topology of a network (TC message) in order to disturb the routing operation.
Typical routing attacks methods against OLSR are:

- Link spoofing attack. A malicious node advertises it has a direct link, which does not really exist, with non-neighbors to disrupt routing and application data operations related to this node. In OLSR, an attacker can advertise a fake link with a victim’s two-hop neighbors. This results in the victim node selecting the attacker as its MPR. As an MPR node, the attacker can manipulate routing traffic and data later on by modifying, dropping, recording and delaying.

- Link withholding attack. A malicious node refuses to advertise the existing link to specific nodes or a group of nodes, which can make these nodes unreachable for others. Specific to OLSR, the malicious node may disclaim the existence of its one-hop neighbors by modifying its HELLO message.

- Replay attack. A malicious node could record other nodes’ valid routing control message and replay them later. Because of the unpredictable mobility of MANET nodes, replaying control massage could advertise some topology which does not exist anymore, and then cause network routing chaos.

- Colluding misrelay attack. Multiple attackers collude to modify or drop routing packets to disrupt MANET routing process. This kind of attack can disrupt up to 100% data packets in OLSR.

4.3 Extended Dempster-Shafer Theory of Evidence

The Dempster-Shafer mathematical theory of evidence is both a theory of evidence and a theory of probable reasoning. The degree of belief models the evidence, while Dempster’s rule of combination (DRC) is the procedure to aggregate and summarize a corpus of evidence. However, previous research efforts identify several limitations of the Dempster’s rule of combination:
1. **Associative.** For DRC, the order of the information in the aggregated evidence does not impact the result. As shown in Yager (1987), a non-associative combination rule is necessary for many cases.

2. **Non-weighted.** DRC implies that we trust all evidence equally Wu et al. (2002). However, in reality, our trust on different evidence may differ. In other words, it means we should consider various factors for each evidence.

Yager (1987) and Yamada et al. Yamada and Kudo (2004) proposed rules to combine several evidence presented sequentially for the first limitation. Wu et al. Wu et al. (2002) suggested a weighted combination rule to handle the second limitation. However, the weight for different evidence in their proposed rule is ineffective and insufficient to differentiate and prioritize different evidence in terms of security and criticality. Our extended Dempster-Shafer theory with importance factors can overcome both of the aforementioned limitations.

### 4.3.1 Importance Factors and Belief Function

In D-S theory, propositions are represented as subsets of a given set. Suppose $\Theta$ is a finite set of states, and let $2^{\Theta}$ denote the set of all subsets of $\Theta$. D-S theory calls $\Theta$, a frame of discernment. When a proposition corresponds to a subset of a frame of discernment, it implies that a particular frame discriminates the proposition. First, we introduce a notion of importance factors.

**Definition 4.1.** Importance factor (IF) is a positive real number associated with the importance of evidence. IFs are derived from historical observations or expert experiences.

**Definition 4.2.** An evidence $E$ is a 2-tuple $(m, IF)$, where $m$ describes the basic probability assignment Shafer (1976). Basic probability assignment function $m$ is
defined as follows:

\[ m(\emptyset) = 0 \]  \hspace{1cm} (4.1)

and

\[ \sum_{A \subseteq \Theta} m(A) = 1 \]  \hspace{1cm} (4.2)

According to Shafer (1976), a function \( Bel : 2^\Theta \rightarrow [0, 1] \) is a belief function over \( \Theta \) if it is given by (3) for some basic probability assignment \( m : 2^\Theta \rightarrow [0, 1] \).

\[ Bel(A) = \sum_{B \subseteq A} m(B) \]  \hspace{1cm} (4.3)

for all \( A \in 2^\Theta \), \( Bel(A) \) describes a measure of the total beliefs committed to the evidence \( A \).

Given several belief functions over the same frame of discernment and based on distinct bodies of evidence, Dempster’s rule of combination, which is given by (4.4), enables us to compute the orthogonal sum, which describes the combined evidence.

Suppose \( Bel_1 \) and \( Bel_2 \) are belief functions over the same frame \( \Theta \), with basic probability assignments \( m_1 \) and \( m_2 \). Then the function \( m : 2^\Theta \rightarrow [0, 1] \) defined by

\[ m(\emptyset) = 0 \text{ and } \]

\[ m(C) = \frac{\sum_{A_i \cap B_j = C} m_1(A_i)m_2(B_j)}{1 - \sum_{A_i \cap B_j = \emptyset} m_1(A_i)m_2(B_j)} \]  \hspace{1cm} (4.4)

for all non-empty \( C \subseteq \Theta \), \( m(C) \) is a basic probability assignment which describes the combined evidence.

Suppose \( IF_1 \) and \( IF_2 \) are importance factors of two pieces of independent evidence named \( E_1 \) and \( E_2 \) respectively. The combination of these two pieces of evidence implies that our total belief to these two pieces of evidence is 1, but in the same time, our belief to either of evidence is less than 1. This is straightforward since if our belief to one evidence is 1, it would mean our belief to the other is 0, which models
meaningless evidence. And we define the importance factors of the combination result equals to \((IF_1 + IF_2)/2\).

**Definition 4.3.** Extended D-S evidence model with importance factors: Suppose \(E_1 = \langle m_1, IF_1 \rangle\) and \(E_2 = \langle m_2, IF_2 \rangle\) are two pieces of independent evidence. Then, the combination of \(E_1\) and \(E_2\) is \(E = \langle m_1 \oplus m_2, (IF_1 + IF_2)/2 \rangle\), where \(\oplus\) is Dempster’s rule of combination with importance factors.

4.3.2 Expected Properties for Our Dempster’s Rule of Combination with Importance Factors

The proposed rule of combination with *importance factors* should be a superset of Dempster’s rule of combination. In this section, we describe four properties that a candidate Dempster’s rule of combination with *importance factors* should follow. Property 1 and Property 2 ensure that the combined result is valid evidence. Property 3 guarantees that the original Dempster’s Rule of Combination is a special case of Dempster’s Rule of Combination with *importance factors*, where the combined evidence have the same priority. Property 4 ensures that importance factors of the evidence are also independent from each other.

**Property 4.1.** No belief ought to be committed to \(\phi\) in the result of our combination rule.

\[
m'(\phi) = 0
\]  \hfill (4.5)

**Property 4.2.** The total belief ought to be equal to 1 in the result of our combination rule.

\[
\sum_{A \subseteq \Theta} m'(A) = 1
\]  \hfill (4.6)

**Property 4.3.** If the importance factors of each evidence are equal, our Dempster’s rule of combination should be equal to Dempster’s rule of combination without im-
importance factors.

\[ m'(A, IF_1, IF_2) = m(A), \text{ if } IF_1 = IF_2 \]  

for all \( A \in \Theta \), where \( m(A) \) is the original Dempster’s Combination Rule.

**Property 4.4.** Importance factors of each evidence must not be exchangeable.

\[ m'(A, IF_1, IF_2) \neq m'(A, IF_2, IF_1) \text{ if } (IF_1 \neq IF_2) \]  

### 4.3.3 Dempster’s Rule of Combination with Importance Factors

In this section, we propose a Dempster’s rule of combination with importance factors. We prove our combination rule follows the properties defined in the previous section.

**Theorem 4.1.** Dempster’s Rule of Combination with Importance Factors (DRCIF):

Suppose \( Bel_1 \) and \( Bel_2 \) are belief functions over the same frame of discernment \( \Theta \), with basic probability assignments \( m_1 \) and \( m_2 \). The importance factors of evidence are \( IF_1 \) and \( IF_2 \). Then the function \( m' : 2^\Theta \to [0, 1] \) defined by

\[ m'(\phi) = 0 \]

and

\[ m'(C, IF_1, IF_2) \]

\[ = \frac{\sum_{A_i \cap B_j = C} \left[ m_1(A_i)^{IF_2} \cdot m_2(B_j)^{IF_1} \right]}{\sum_{C \subseteq \Theta, C \neq \phi} \sum_{A_i \cap B_j = C} \left[ m_1(A_i)^{IF_2} \cdot m_2(B_j)^{IF_1} \right]} \]

for all non-empty \( C \subseteq \Theta \), \( m' \) is a basic probability assignment for the combined evidence.
It is obvious that our proposed DRCIF holds Property 1 and Property 4. We prove our proposed DRCIF also holds Property 2 and Property 3 here.

Property 2:

\[
\sum_{A \subseteq \Theta} m'(A, IF_1, IF_2) = \sum_{A_i \cap B_j = A} \frac{\sum_{A \subseteq \Theta, A \neq \phi} \sum_{A_i \cap B_j = A} \sum_{A \subseteq \Theta, A \neq \phi} [m_1(A_i)^{IF_1} \cdot m_2(B_j)^{IF_2}]}{\sum_{A \subseteq \Theta, A \neq \phi} \sum_{A_i \cap B_j = A} \sum_{A \subseteq \Theta, A \neq \phi} [m_1(A_i)^{IF_1} \cdot m_2(B_j)^{IF_2}]} = 1
\]

Property 3:

\[
m'(A, IF_1, IF_1) = \sum_{A_i \cap B_j = A} \frac{\sum_{A \subseteq \Theta, A \neq \phi} \sum_{A_i \cap B_j = A} [m_1(A_i)^{IF_1} \cdot m_2(B_j)^{IF_1}]}{\sum_{A \subseteq \Theta, A \neq \phi} \sum_{A_i \cap B_j = A} [m_1(A_i) \cdot m_2(B_j)]} = \frac{\sum_{A_i \cap B_j = A} m_1(A_i)m_2(B_j)}{1 - \sum_{A_i \cap B_j = \phi} m_1(A_i)m_2(B_j)} = m(A)
\]
Our proposed DRCIF is non-associative for multiple pieces of evidence. Therefore, for the case in which sequential information is not available for some instances, it is necessary to make the result of combination consistent with multiple pieces of evidence. Our combination algorithm supports this requirement and the complexity of our algorithm is $O(n)$, where $n$ is the number of pieces of evidence. It indicates our extended Dempster-Shafer theory demands no extra computational cost compared to a naïve fuzzy-based method. The algorithm for combination of multiple pieces of evidence is constructed as follows:

**Algorithm 2: MUL-EDS-CMB**

**Input:** Evidence pool $Ep$

**Output:** One evidence

1. $|Ep| = \text{sizeof}(Ep)$;
2. while $|Ep| > 1$ do
3. \hspace{1em} Pick two pieces of evidence with the least $IF$ in $Ep$, named $E_1$ and $E_2$;
4. \hspace{1em} Combine these two pieces of evidence, $E = \langle m_1 \oplus m_2, (IF_1 + IF_2)/2 \rangle$;
5. \hspace{1em} Remove $E_1$ and $E_2$ from $Ep$;
6. \hspace{1em} Add $E$ to $Ep$;
7. end
8. return the evidence in $Ep$

### 4.4 Overview of Risk-Aware Response Mitigation for MANET Routing Attacks

In this section, we articulate an adaptive risk-aware response mechanism based on quantitative risk estimation and risk tolerance. Instead of applying simple binary isolation of malicious nodes, our approach adopts an isolation mechanism in a tem-
poral manner based on the risk value. We perform risk assessment with the extended D-S evidence theory introduced in Section 4.5 for both attacks and corresponding countermeasures to make more accurate response decisions illustrated in Figure 4.1.

Because of the infrastructure-less architecture of MANET, our risk-aware response system is distributed, which means each node in this system makes its own response decisions based on the evidence and its own individual benefits. Therefore, some nodes in MANET may isolate the malicious node, but others may still keep in cooperation with due to high dependency relationships. Our risk-aware response mechanism is divided into the following four steps shown in Figure 4.1.

**Evidence Collection.** In this step, Intrusion Detection System (IDS) gives an attack alert with a confidence value, and then Routing Table Change Detector (RTCD) runs to figure out how many changes on routing table are caused by the attack.

**Risk Assessment.** Alert confidence from IDS and the routing table changing information would be further considered as independent evidence for risk calculation and combined with the extended D-S theory. Risk of countermeasures is calculated as well during a risk assessment phase. Based on the risk of attacks and the risk of countermeasures, the entire risk of an attack could be figured out.

**Decision Making.** The adaptive decision module provides a flexible response decision making mechanism, which takes risk estimation and risk tolerance into account. To adjust temporary isolation level, a user can set different thresholds to fulfill her goal.

**Intrusion Response.** With the output from risk assessment and decision making module, the corresponding response actions, including routing table recovery and node isolation, are carried out to mitigate attack damages in a distributed manner.
In our approach, we use two different responses to deal with different attack methods: **routing table recovery** and **node isolation**.

**Routing table recovery** includes local routing table recovery and global routing recovery. Local routing recovery is performed by victim nodes that detect the attack and automatically recover its own routing table. Global routing recovery involves with sending recovered routing messages by victim nodes and updating their routing table based on corrected routing information in real time by other nodes in MANET.

Routing table recovery is an indispensable response and should serve as the first response method after successful detection of attacks. In proactive routing protocols like OLSR, routing table recovery does not bring any additional overhead since it periodically goes with routing control messages. Also, as long as the detection of attack is positive, this response causes no negative impacts on existing routing operations.

**Node isolation** may be the most intuitive way to prevent further attacks from being launched by malicious nodes in MANET. To perform a node isolation response, the neighbors of the malicious node ignore the malicious node by neither forwarding packets through it nor accepting any packets from it. On the other hand, a binary
node isolation response may result in negative impacts to the routing operations, even bringing more routing damages than the attack itself.

**Figure 4.2: An Example of Link Spoofing Attack**

For example, in Figure 4.2, Node 1 behaves like a malicious node. However, if every other node simply isolates Node 1, Node 6 will be disconnected from the network. Therefore, more flexible and fine-grained node isolation mechanism are required. In our risk-aware response mechanism, we adopt two types of time-wise isolation responses: *temporary isolation* and *permanent isolation*, which are discussed in Section 4.6.

### 4.5 Risk Assessment Based on Routing Table Change Patterns

Since the attack response actions may cause more damages than attacks, the risks of both attack and response should be estimated. We classify the security states of MANET into two categories: {Secure, Insecure}. In other words, the frame of discernment would be \( \{ \phi, \{ \text{Secure} \}, \{ \text{Insecure} \}, \{ \text{Secure, Insecure} \} \} \). Note that \{Secure, Insecure\} means the security state of MANET could be either secure or insecure, which describes the uncertainty of the security state. \( \text{Bel}\{\text{Insecure}\} \) is used to represent the risk of MANET.
Our evidence selection approach considers subjective evidence from experts’ knowledge and objective evidence from routing table modification. We propose a unified analysis approach for evaluating the risks of both attack ($Risk_A$) and countermeasure ($Risk_C$).

We take the confidence level of alerts from IDS as the subjective knowledge in Evidence 1. In terms of objective evidence, we analyze different routing table modification cases. There are three basic items in OLSR routing table ($destination$, $next hop$, $distance$). Thus, routing attack can cause existing routing table entries to be missed, or any item of a routing table entry to be changed. We illustrate the possible cases of routing table change and analyze the degrees of damage in Evidence 2 through 5.

**Evidence 1: Alert Confidence.** The confidence of attack detection by the IDS is provided to address the possibility of the attack occurrence. Since the false alarm is a serious problem for most IDSs, the confidence factor must be considered for the risk assessment of the attack. The basic probability assignments of Evidence 1 are based on three equations 4.9–4.11:

\[
m(Insecure) = c, \text{ } c \text{ is confidence given by IDS} \tag{4.9}
\]

\[
m(Secure) = 1 - c \tag{4.10}
\]

\[
m(Secure, Insecure) = 0 \tag{4.11}
\]

**Evidence 2: Missing Entry.** This evidence indicates the proportion of missing entries in routing table. Link withholding attack or node isolation countermeasure can cause possible deletion of entries from routing table of the node.
Evidence 3: Changing Entry Type I. This evidence represents the proportion of changing entries in the case of next hop being the malicious node. In this case, the malicious node builds a direct link to this node. So it is highly possible for this node to be the attacker’s target. Malicious node could drop all the packages to or from the target node, or it can behave as a normal node and wait for future attack actions. Note that isolating a malicious node cannot trigger this case.

Evidence 4: Changing Entry Type II. This evidence shows the proportion of changed entries in the case of different next hop (not the malicious node) and the same distance. We believe the impacts on the node communication should be very minimal in this case. Both attacks and countermeasures could cause this case.

Evidence 5: Changing Entry Type III. This evidence points out the proportion of changing entries in the case of different next hop (not the malicious node) and the different distance. Similar to Evidence 4, both attacks and countermeasures could result in this evidence. The path change may also affect routing cost and transmission delay of the network.

Basic probability assignments of Evidence 2 to 5 are based on Equations 4.12, 4.13 and 4.14. Equations 4.12, 4.13 and 4.14 are piecewise linear functions, where a, b, c, and d are constants and determined by experts. d is the minimum value of the belief that implies the status of MANET is insecure. On the other hand, 1-d is the maximum value of the belief that means the status of MANET is secure. a, b, and c are the thresholds for minimum belief or maximum belief for each respective mass function.

\[
m(\text{Insecure}) = \begin{cases} 
  d & x \in [0,a] \\
  (1 - \frac{2d}{c-a})(x - a) & x \in (a,c) \\
  1 - d & x \in (c,1] 
\end{cases} \tag{4.12}
\]
\[ m(\text{Secure}) = \begin{cases} 
1 - d + \left(\frac{2d - 1}{b}\right)x & x \in [0, b] \\
d & x \in (b, 1] 
\end{cases} \quad (4.13) \]

\[ m(\text{Secure, Insecure}) = \begin{cases} 
\frac{1 - 2d}{b}x & x \in [0, a] \\
d - \frac{2d - 1}{b}x - \left(\frac{1 - 2d}{c - a}\right)(x - a) & x \in (a, b] \\
1 - b - \left(\frac{1 - 2d}{c - a}\right)(x - a) & x \in (b, c] \\
0 & x \in (c, 1] 
\end{cases} \quad (4.14) \]

\subsection*{4.5.2 Combination of Evidence}

For simplicity, we call the combined evidence for an attack, \( E_A \) and the combined evidence for a countermeasure, \( E_C \). Thus, \( \text{Bel}_A(\text{Insecure}) \) and \( \text{Bel}_C(\text{Insecure}) \) represent risks of attack \( (\text{Risk}_A) \) and countermeasure \( (\text{Risk}_C) \), respectively. The combined evidence, \( E_A \) and \( E_C \) are defined in Equations 4.15 and 4.16. The entire risk value derived from \( \text{Risk}_A \) and \( \text{Risk}_C \) is given in Equation 4.17.

\[ E_A = E_1 \oplus E_2 \oplus E_3 \oplus E_4 \oplus E_5 \quad (4.15) \]

\[ E_C = E_2 \oplus E_4 \oplus E_5 \quad (4.16) \]

where \( \oplus \) is \textit{Dempster’s rule of combination with important factors} defined in Theorem 1.

\[ \text{Risk} = \text{Risk}_A - \text{Risk}_C = \text{Bel}_A(\text{Insecure}) - \text{Bel}_C(\text{Insecure}) \quad (4.17) \]
4.6 Adaptive Decision Making

Our adaptive decision making module is based on quantitative risk estimation and risk tolerance, which is shown in Figure 4.3. The response level is additionally divided into multiple bands. Each band is associated with an isolation degree, which presents a different time period of the isolation action. The response action and band boundaries are all determined in accordance with risk tolerance and can be changed when risk tolerance threshold changes. The upper risk tolerance threshold ($UT$) would be associated with permanent isolation response. The lower risk tolerance threshold ($LT$) would remain each node intact. The band between the upper tolerance threshold and lower tolerance threshold is associated with the temporary isolation response, in which the isolation time ($T$) changes dynamically based on the different response level given by Equations 4.18 and 4.19, where $n$ is the number of bands and $i$ is the corresponding isolation band.

$$i = \left\lceil \frac{Risk - LT}{UT - LT} \times n \right\rceil, \quad Risk \in (LT, UT)$$ \hspace{1cm} (4.18)

$$T = 100 \times i \text{ (milliseconds)}$$ \hspace{1cm} (4.19)

We recommend the value of lower risk tolerance threshold be 0 initially if no additional information is available. It implies when the risk of attack is greater than the risk of isolation response, the isolation is needed. If other information is available, it could be used to adjust thresholds. For example, node reputation is one of important factors in MANET security, our adaptive decision making module could take this factor into account as well. That is, if the compromised node has a high or low reputation level, the response module can intuitively adjust the risk tolerance thresholds accordingly. In the case that $LT$ is less than 0, even if the risk of attack
is not greater than the risk of isolation, the response could also perform an isolation task to the malicious nodes.

The risk tolerance thresholds could also be dynamically adjusted by other factors, such as attack frequency. If the attack frequency is high, more severe response action should be taken to counter this attack. Our risk-aware response module could achieve this objective by reducing the values of risk tolerance threshold and narrowing the range between two risk tolerance thresholds.

### 4.7 Case Study and Evaluation

In this section, we first explain the methodology of our experiments and the metrics considered to evaluate the effectiveness of our approach. Then, we demonstrate the detailed process of our solution with a case study and also compare our risk-aware approach with binary isolation. In addition, we evaluate our solution with five random network topologies considering different size of nodes. The results show the effectiveness and scalability of our approach.
Figure 4.4: Routing Tables

4.7.1 Methodology and Metrics

The experiments were carried out using NS-2 as the simulation tool from VINT Project Fall and Varadhan (2010) with UM-OLSR Ros (2007). NS-2 is a discrete event network simulator which provides a detailed model of the physical and link layer behavior of a wireless network and allows arbitrary movement of nodes within the network. UM-OLSR is an implementation of Optimized Link State Routing protocol for the NS-2, which complies with Clausen and Jacquet (2003) and supports all core functionalities of OLSR plus the link-layer feedback option. In our experiments,
we constructed MANET scenarios in a topology of 1000m×1000m area. The total simulation time was set to 1200 seconds, and the bandwidth was set to 2 Mbps. Constant Bit Rate (CBR) traffic was used to send 512 byte-UDP packets between nodes. The queuing capacity of every node was set to 15. We adopted a random traffic generator in the simulation that chose random pairs of nodes and sent packets between them. Every node kept track of all packets sent by itself and the entire packet received from other nodes in the network.

In order to evaluate the effectiveness of our adaptive risk-aware response solution, we divided the simulation process into three stages and compared the network performance in terms of six metrics. The following describes the activities associated with each stage:

**Stage 1 - Before attack:** random packets were generated and transmitted among nodes without activating any of them as attackers. This simulation can present the traffic patterns under the normal circumstance.

**Stage 2 - After attack:** specific nodes were set as attackers which conducted malicious activities for their own profits. However, any detection or response is not available in this stage. This simulation process can present the traffic patterns under the circumstance with malicious activities.

**Stage 3 - After response:** response decisions for each node were made and carried out based on three different mechanisms.

We computed six metrics Hu et al. (2005) for each simulation run:

- **Packet Delivery Ratio:** The ratio between the number of packets originated by the application layer CBR sources and the number of packets received by the CBR sink at the final destination.

- **Routing Cost:** The ratio between the total bytes of routing packets transmitted
during the simulation and the total bytes of packets received by the CBR sink at the final destination.

- **Packet Overhead**: The number of transmitted routing packets; for example, a HELLO or TC message sent over four hops would be counted as four packets in this metric.

- **Byte Overhead**: The number of transmitted bytes by routing packets, counting each hop similar to *Packet Overhead*.

- **Mean Latency**: The average time elapsed from “when a data packet is first sent” to “when it is first received at its destination.”

- **Average Path Length**: This is the average length of the paths discovered by OLSR. It was calculated by averaging the number of hops taken by each data packet to reach the destination.

### 4.7.2 Case Study

Figure 4.2 shows our case study scenario, where packets from Node 5 to Node 0 are supposed to go through Node 2 and Node 4. Suppose a malicious Node 1 advertises it has a direct link (fake link) to Node 0 and it would cause every node to update its own routing table accordingly. As a result, the packets from Node 5 to Node 0 traverse Node 1 rather than Node 2 and Node 4. Hence, Node 1 can drop and manipulate the traffic between Node 5 and Node 0. We assume, as Node 1’s one-hop neighbors, Node 0, Node 4 and Node 6 get the intrusion alerts with 80% confidence from their respective IDS modules. Figures 4.4(a)-(c) show the routing tables of Node 0, Node 4 and Node 6 before the attack, after the attack and after the isolation, respectively.
We set \( a = 0.2, \) \( b = 0.7, \) \( c = 0.8, \) \( d = 0.05, \) \( IF_1 = 5, \) \( IF_2 = 7, \) \( IF_3 = 10, \) \( IF_4 = 3, \) \( IF_5 = 3, \) \( LT = -0.0017, \) \( UT = 1, \) and \( n = 5 \) in our experiments.

We examine binary isolation approach, risk-aware approach with DRC, and risk-aware approach with DRCIF to calculate the response decisions for Node 0, Node...
Table 4.1: Risk Assessment and Decision Making

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Index</th>
<th>Node</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td><strong>BINNARY</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decision</td>
<td>isolation</td>
<td>isolation</td>
</tr>
<tr>
<td>Risk (A)</td>
<td>0.00011</td>
<td>0.0000057</td>
</tr>
<tr>
<td>Risk (C)</td>
<td>0.00164</td>
<td>0.00164</td>
</tr>
<tr>
<td>Risk</td>
<td>-0.00153</td>
<td>-0.00163</td>
</tr>
<tr>
<td><strong>DRC</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decision</td>
<td>isolation</td>
<td>isolation</td>
</tr>
<tr>
<td>Risk (A)</td>
<td>0.467</td>
<td>0.00355</td>
</tr>
<tr>
<td>Risk (C)</td>
<td>0.0136</td>
<td>0.0136</td>
</tr>
<tr>
<td>Risk</td>
<td>0.4534</td>
<td>-0.01005</td>
</tr>
<tr>
<td><strong>DRCIF</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decision</td>
<td>isolation</td>
<td>no isolation</td>
</tr>
<tr>
<td>Time</td>
<td>300 ms</td>
<td>0</td>
</tr>
</tbody>
</table>

4 and Node 6. As shown in Table 4.1, binary isolation suggests all nodes to isolate the malicious one since it does not take countermeasure risk into account. With our risk-aware response mechanism based on our extended D-S theory, Node 1 should be isolated only by Node 0 while the original D-S theory would suggest that both Node 0 and Node 4 isolate Node 1.

In Figure 4.5(a), due to routing attacks, the packet delivery ratio decreases in Stage 2. After performing binary isolation and DRC risk-aware response in Stage 3, the packet delivery ratio even decreases more. This is because these two response mechanisms largely destroy the topology of network. However, the packet delivery ratio using our DRCIF risk-aware response in Stage 3 is higher than those of the former two response mechanisms.
In Figure 4.5(b), the routing attacks increase the routing cost in Stage 2. Rather than recovering the routing cost in Stage 3, binary isolation and DRC risk-aware responses increase the routing cost. DRCIF risk-aware response, however, decreases
the routing cost. Compared with other two response mechanisms, it indicates our DRCIF risk-aware response effectively handles the attack.

Figures 4.5(c) and 4.5(d) show the packet and byte overhead, respectively. Since the routing attacks do not change the network topology further in the given case, the packet overhead and byte overhead remain almost the same in Stage 2. In Stage 3, however, they are higher when our DRCIF risk-aware response mechanism is applied. This result meet our expectation, because the number of nodes which isolate malicious node using binary isolation and DRC risk-aware response are greater than those of our DRCIF risk-aware response mechanism. As shown in Table 4.1, the number of isolated nodes for each mechanism varies.

In Figure 4.5(e), as a consequence of the routing attacks, the mean latency increases in Stage 2. After response, we notice the mean latencies in Stage 3 for three different response mechanisms have approximately the same results.

In Figure 4.5(f), the average path length decreases in Stage 2 due to the malicious action claiming a shorter path performed by Node 1. After response, the average path length using binary isolation is higher than those of the other two response mechanisms because more nodes isolated the malicious node based on the nature of binary isolation. Hence, some packets may be retransmitted by more hops than before.

4.7.3 Evaluation with Random Network Topologies

In order to test the effectiveness and scalability of our solution, we evaluated our risk-aware approach with DRCIF on five random network topologies. These five topologies have 10, 20, 30, 40 and 50 nodes respectively.

Figure 4.6 shows the performance results in these random network topologies of our risk-aware approach with DRCIF, risk-aware approach with DRC and binary
isolation approach. In Figure 4.6(a), as the number of nodes increases, the packet delivery ratio also increases because there are more route choices for the packet transmission. Among these three response mechanisms, we also notice the packets delivery ratio of our DRCIF risk-aware response is higher than those of the other two approaches.

In Figure 4.6(b), we can observe that the routing cost of our DRCIF risk-aware response is lower than those of the other two approaches. Note that the fluctuations of routing cost shown in Figure 4.6(b) are caused by the random traffic generation and random placement of nodes in our realistic simulation.

In our DRCIF risk-aware response, the number of nodes which isolate the malicious node is less than the other two response mechanisms. As shown in Figures 4.6(c) and 4.6(d), that is the reason why we can also notice that as the number of nodes increases, the packet overhead and the byte overhead using our DRCIF risk-aware response are slightly higher than those of the other two response mechanisms.

In Figure 4.6(e), the mean latency using our DRCIF risk-aware response is higher than those of the other two response mechanisms, when the number of nodes is smaller than 20. However, when the number of nodes is greater than 20, the mean latency using our approach is less than those of the other two response mechanisms.

4.8 Related Work

Intrusion Detection and Response in MANET. Some research efforts have been made to seek preventive solutions Hu et al. (2005); Levine et al. (2002); Hu et al. (2003); Awerbuch et al. (2008) for protecting the routing protocols in MANET. Although these approaches can prevent unauthorized nodes from joining the network, they introduce a significant overhead for key exchange and verification with the limited intrusion elimination. Besides, prevention-based techniques are less helpful to
cope with malicious insiders who possess the legitimate credentials to communicate in the network.

Numerous IDSs for MANET have been recently introduced. Due to the nature of MANET, most IDS are structured to be distributed and have a cooperative architecture. Similar to signature-based and anomaly-based IDS models for the wired network, IDSs for MANET use specification-based or statistics-based approaches. Specification-based approaches, such as DEMEM Tseng et al. (2006b) and Tseng et al. (2006a); Mohammed et al. (2011); Felix et al. (2011), monitor network activities and compare them with known attack features, which are impractical to cope with new attacks. On the other hand, statistics-based approaches, such as Watchdog Marti et al. (2000), and Kurosawa et al. (2006), compare network activities with normal behavior patterns, which result in higher false positives rate than specification-based ones. Because of the existence of false positives in both MANET IDS models, intrusion alerts from these systems always accompany with alert confidence, which indicates the possibility of attack occurrence.

Intrusion response system (IRS) Hu et al. (2004) for MANET is inspired by MANET IDS. In Sun et al. (2006b) and Refaei et al. (2010), malicious nodes are isolated based on their reputations. Their work fails to take advantage of IDS alerts and simple isolation may cause unexpected network partition. Wang et al. Wang et al. (2007) brought the concept of cost-sensitive intrusion response which considers topology dependency and attack damage. The advantage of our solution is to integrate evidence from IDS, local routing table with expert knowledge, and countermeasures with a mathematical reasoning approach.

Risk-aware Approaches. When it comes to make response decisions Toth and Kruegel (2002b); Strasburg et al. (2009), there always exists inherent uncertainty which leads to unpredictable risk, especially in security and intelligence arena. Risk-
aware approaches are introduced to tackle this problem by balancing action benefits and damage tradeoffs in a quantified way. Cheng et al. Cheng et al. (2007) presented a fuzzy logic control model for adaptive risk-based access control. Teo et al. Teo et al. (2003) applied dynamic risk-aware mechanism to determine whether an access to the network should be denied or permitted. Jing et al. Jing et al. (2014) presented continuous and automated risk assessment of mobile applications.

However, risk assessment is still a non-trivial challenging problem due to its involvements of subjective knowledge, objective evidence and logical reasoning. Wang et al. Wang et al. (2007) proposed a naïve fuzzy cost-sensitive intrusion response solution for MANET. Their cost model took subjective knowledge and objective evidence into account but omitted a seamless combination of two properties with logical reasoning. Mu et al. Mu et al. (2008) adopted Dempster-Shafer theory to measure the risk of attacks and responses. However, as identified in Sentz and Ferson (2002), their model with Dempster’s rule treats evidence equally without differentiating them from each other. To address this limitation, we propose a new Dempster’s rule of combination with a notion of importance factors in D-S evidence model.

4.9 Summary

We have proposed a risk-aware response solution for mitigating MANET routing attacks. Especially our approach considered the potential damages of attacks and countermeasures. In order to measure the risks of both attacks and countermeasures, we extended Dempster-Shafer theory of evidence with a notion of importance factors. Based on several metrics, we also investigated the performance and practicality of our approach and the experiment results clearly demonstrated the effectiveness and scalability of our risk-aware approach.
Chapter 5

GUESSING PICTURE PASSWORDS

5.1 Introduction

Using text-based passwords that include alphanumerics and symbols on touch-screen devices is unwieldy and time-consuming due to small-sized screens and the absence of physical keyboards. Consequently, mobile operating systems, such as iOS and Android, integrate a numeric PIN and a draw pattern as alternative authentication schemes to provide user-friendly login services. However, the password spaces of these schemes are significantly smaller than text-based passwords, rendering them less secure and easy to break with some knowledge of device owners Bonneau et al. (2012).

To bring a fast and fluid login experience on touch-screen devices, the Windows 8™ operating system comes with a picture password authentication system, namely picture gesture authentication (PGA) Johnson (2012), which is also an instance of background draw-a-secret (BDAS) schemes Dunphy and Yan (2007). This new authentication mechanism hit the market with miscellaneous computing devices including personal computers and tablets. At the time of writing, over 60 million Windows 8™ licenses have been sold Foley (2013) and it is estimated that 400 million computers and tablets will run Windows 8™ with this newly introduced authentication scheme in one year Ovide (2012). Consequently, it is imperative to examine and explore potential attacks on picture gesture authentication in such a prevalent operating system for further understanding user experiences and enhancing this commercially popular picture password system.
Many graphical and gesture-based password schemes—including DAS Jermyn et al. (1999), Face Brostoff and Sasse (2000), Story Davis et al. (2004), PassPoints Wiedenbeck et al. (2005) and BDAS Dunphy and Yan (2007)—have been proposed in the past decade (for more, please refer to Dhamija and Perrig (2000); Thorpe and Van Oorschot (2004b); Suo et al. (2005); Chiasson et al. (2007); Gao et al. (2008); Bicakci et al. (2009); Biddle et al. (2011); Chiasson et al. (2012); Li et al. (2013c,b,a)). Amongst these schemes, click-based schemes, such as PassPoints, have attracted considerable attention and some research has analyzed the patterns and predictable characteristics shown in their passwords Chiasson et al. (2009); van Oorschot and Thorpe (2011). Furthermore, harvesting characteristics from passwords of a target picture and exploiting hot-spots and geometric patterns on the target picture have been proven effective for attacking click-based schemes Dirik et al. (2007); Thorpe and van Oorschot (2007); Salehi-Abari et al. (2008). However, PGA allows complex gestures other than a simple click. Moreover, a new feature in PGA, autonomous picture selection by users, makes it unrealistic to harvest passwords from the target pictures for learning. In other words, the target picture is previously unseen to any attack models. All existing attack approaches lack a generic knowledge representation of user choice in password selection that should be abstracted from specific pictures. The absence of this abstraction makes existing attack approaches impossible or abysmal (if possible) to work on previously unseen target pictures.

In this chapter, we provide an empirical analysis of user choice in PGA based on real-world usage data, showing interesting findings on user choice in selecting background picture, gesture location, gesture order, and gesture type. In addition, we propose a new attack framework that represents and learns users’ password selection patterns from training datasets and generates ranked password dictionaries for previously unseen target pictures. To achieve this, it is imperative to build generic
knowledge of user choice from the abstraction of hot-spots in pictures. The core of our framework is the concept of a selection function that simulates users’ selection processes in choosing their picture passwords. Our approach is not coupled with any specific pictures. Hence, the generation of a ranked password list is then transformed into the generation of a ranked selection function list which is then executed on the target pictures. We present two algorithms for generating the selection function list: one algorithm is to appropriately develop an optimal guessing strategy for a large-scale training dataset and the other deals with the construction of high-quality dictionaries even when the size of the training dataset is small. We also discuss the implementation of our attack framework over PGA, and evaluate the efficacy of our proposed approach with the collected datasets.

The rest of this chapter is organized as follows. Section 5.2 gives an overview of picture gesture authentication. Section 5.3 discusses our empirical analysis on picture gesture authentication. In Section 5.4, we illustrate our attack framework. Section 5.5 presents the implementation details and automated identified PoIs and gestures. Section 5.6 presents the evaluation results of the proposed attack framework. We discuss several research issues in Section 5.7 followed by the related work in Section 5.8. Section 5.9 concludes the chapter.

5.2 Background: Picture Gesture Authentication

Like other login systems, Windows 8® Picture Gesture Authentication has two independent phases, namely registration and authentication. In the registration stage, a user chooses a picture from his or her local storage as the background. PGA does not force users to choose pictures from a predefined repository. Even though users may choose pictures from common folders, such as the Picture Library folder in Windows 8®, the probability for different users to choose an identical picture as the
background for their passwords is low. This phenomenon requires potential attack approaches to have the ability to perform attacks on previously unseen pictures. PGA then asks the user to draw exactly three gestures on the picture with his or her finger, mouse, stylus, or other input devices depending on the equipment he or she is using. A gesture could be viewed as the cursor movements between a pair of ‘finger-down’ and ‘finger-up’ events. PGA does not allow free-style gestures, but only accepts tap (indicating a location), line (connecting areas or highlighting paths), and circle (enclosing areas) Pace (2011a). If the user draws a free-style gesture, PGA will convert it to one of the three recognized gestures. For instance, a curve would be converted to a line and a triangle or oval will be stored as a circle. To record these gestures, PGA divides the longest dimension of the background image into 100 segments and the short dimension on the same scale to create a grid, then stores the coordinates of the gestures. The line and circle gestures are also associated with additional information such as directions of the finger movements.

Once a picture password is successfully registered, the user may login the system by drawing corresponding gestures instead of typing his or her text-based password. In other words, PGA first brings the background image on the screen that the user chose in the registration stage. Then, the user should reproduce the drawings he or she set up as his or her password. PGA compares the input gestures with the previously stored ones from the registration stage. The comparison is not strictly rigid but shows tolerance to some extent. If any of gesture type, ordering, or directionality is wrong, the authentication fails. When they are all correct, an operation is further taken to measure the distance between the input password and the stored one. For tapping, the gesture passes authentication if the predicate $12 - d^2 \geq 0$ satisfies, where $d$ denotes the distance between the tap coordinates and the stored coordinates. The starting
and ending points of line gestures and the center of circle gestures are measured with the same predicate Pace (2011a).

The differences between PGA and the first BDAS scheme proposed in Dunphy and Yan (2007) include: i) in PGA, a user uploads his or her picture as the background instead of choosing one from a predefined picture repository; ii) a user is only allowed to draw three specific types of gestures in PGA, while BDAS takes any form of strokes. The first difference makes PGA more secure than the previous scheme, because a password dictionary could only be generated after the background picture is acquired. However, the second characteristic reduces the theoretical password space from its counterpart. Pace et al. Pace (2011a) quantified the size of the theoretical password space of PGA which is \(2^{30.1}\) with current length-three configuration in Windows 8™. For more details, please refer to Pace (2011a).

5.3 User Choice Patterns in Picture Gesture Authentication Passwords

In this section, we present an empirical analysis on user choice in PGA by analyzing data collected from our user studies. Our empirical study is based on human cognitive capabilities. Since human cognition of pictures is limited in a similar way to their cognition of texts, the picture passwords selected by users are probably constrained by human cognitive limits which would be similar to the ones in text-based passwords Yuille (1983).

5.3.1 Experiment Design

For the empirical study, we developed a web-based PGA system for conducting user studies. The developed system resembles Windows 8™ PGA in terms of its workflow and appearance. The differences between our implementation and Windows 8™ PGA include: i) our system works with major browsers in desktop PCs and
tablets whereas Windows 8™ PGA is a stand-alone program; ii) some information, such as the criterion for circle radius comparison, is not disclosed. In other words, our implementation and Windows 8™ PGA differ in some criteria (we regard radiiuses the same if their difference is smaller than 6 segments in grid). In addition, our developed system has a tutorial page that includes a video clip educating how to use the system and a test page on which users can practice gesture drawings.

Our study protocol, including the type of data we plan to collect and the questionnaire we plan to use, was reviewed by our institution’s IRB. The questionnaire consisted of four sections: i) general information of the subject (gender, age, level of education received, and race); ii) general feeling toward PGA (is it easier to remember, faster to input, harder to guess, and easier to observe than text-based password); iii) selection of background picture (preferred picture type); and iv) selection of password (preferred gesture location and type).

We started user studies after receiving the IRB approval letter in August 2012 and compiled two datasets from August 2012 to January 2013 using this system. Dataset-1 was acquired from a testbed of picture password used by an undergraduate computer science class. Dataset-2 was produced by advertising our studies in schools of engineering and business in two universities and Amazon’s Mechanical Turk crowdsourcing service that has been used in security-related research work Kel- ley et al. (2012). Turkers who had finished more than 50 tasks and had an approval rate greater than 60% were qualified for our user study.

For registration, subjects in Dataset-1 were asked to provide their student IDs for a simple verification after which they were guided to upload a picture, register a password and then use the password to access class materials including slides, homework, assignments, and projects. Subjects used this system for the Fall 2012 semester which lasted three and a half months at our university. If subjects forgot
their passwords during the semester, they would inform the teaching assistant who reset their passwords. Subjects were allowed to change their passwords by clicking a change password link after login. There were 56 subjects involved in Dataset-1 resulting in 58 unique pictures, 86 registered passwords, and 2,536 login attempts.

Instead of asking subjects to upload pictures for Dataset-2, we chose 15 pictures in advance from the PASCAL Visual Object Classes Challenge 2007 dataset Everingham et al. (2007). Figure 5.1 shows the 15 images that are used in Dataset-2 as the background pictures for password selection. We chose these pictures because they represent a diverse range of pictures in terms of category (portrait, wedding, party, bicycle, train, airplane and car) and complexity (pictures with few and plentiful stand-out regions). Subjects were asked to choose one password for each picture by pretending that it was protecting their bank information. The 15 pictures were presented to subjects in a random order to reduce the dependency of password selection upon the picture presentation order. 762 subjects participated in the Dataset-2 collection resulting in 10,039 passwords. The number of passwords for each picture in the Dataset-2 varies slightly, with an average of 669, because some subjects quit the study without setting up passwords for all pictures.
**Table 5.1:** Survey Question: Which of the following best describes what you are considering when you choose locations to perform gestures?

<table>
<thead>
<tr>
<th>Multi-choice Answers</th>
<th>Dataset</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>I try to find locations where special objects are, such as face, eye, clock, car,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>badge, etc.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24 (72.7%)</td>
<td>389 (59.6%)</td>
<td>413 (60.3%)</td>
</tr>
<tr>
<td>I try to find locations where some special shapes are, such as circle and line, etc.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 (24.2%)</td>
<td>143 (21.9%)</td>
<td>151 (22.1%)</td>
</tr>
<tr>
<td>I try to find locations where colors are different from their surroundings, such as</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a red apple in a green lemon pile, etc.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 (0%)</td>
<td>57 (8.7%)</td>
<td>57 (8.3%)</td>
</tr>
<tr>
<td>I randomly choose a location to draw without thinking about the background picture.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 (3.0%)</td>
<td>66 (10.1%)</td>
<td>67 (9.8%)</td>
</tr>
</tbody>
</table>

For both datasets, subjects were asked to finish the aforementioned questionnaire to help us understand their experiences. We collected 685 (33 for Dataset-1, 652 for Dataset-2) copies of survey answers in total. According to the demographic-related inquiries in the exit survey, 81.8% subjects in Dataset-1 are self-reported male and 63.6% are between 18 and 24 years old. While participants in Dataset-2 are more diverse with 64.4% male, 37.2% among 18 to 24 years old, 45.4% among 25 - 34, and 15.0% among 35 - 50. Even though the subjects in our studies do not represent all possible demographics, the data collected from them represents the most comprehensive PGA usage so far. Their tendencies could provide us with significant insights into the user choice in PGA.
5.3.2 Findings

This section summarizes our empirical analysis on the above-mentioned datasets by presenting five findings.

Finding 1: Relationship Between Background Picture and User’s Identity, Personality, or Interests

We analyzed all unique pictures in Dataset-1, and the background pictures chosen by subjects range from celebrity to system screenshot. We categorize them into six classes: i) people (27/58), ii) civilization (7/58), iii) landscape (3/58), iv) computer-generated picture (14/58), v) animals (6/58), and vi) others (1/58).

For the category of ‘people’, 6 pictures were categorized as ‘me’; 12 pictures were subjects’ families; 4 were pictures of subjects’ friends; and 5 were celebrities. The analysis of answers to the survey question “Could you explain why you choose such types of pictures?” revealed two opposite attitudes towards using picture of people. The advocates for such pictures considered: i) it is more friendly. e.g. “The image was special to me so I enjoy seeing it when I log in”; ii) it is easier for remembering passwords. e.g. “Marking points on a person is easier to remember”; and iii) it makes password more secure. e.g. “The picture is personal so it should be much harder for someone to guess the password”. However, other participants believed it may leak his or her identity or privacy. e.g. “revealing myself or my family to anyone who picks up the device”. They preferred other types of pictures because “less personal if someone gets my picture” and “landscape usually doesn’t have any information about who you are”.

14 pictures in Dataset-1 could be categorized as computer-generated pictures including computer game posters, cartoons, and some geometrical graphs. 24.1%
(14/58) of such pictures were observed in Dataset-1 but the survey results indicated 6.4% (42/652) of participants were in such a usage pattern in Dataset-2 based on the following survey question: “Please indicate the type of pictures you prefer to use as the background”. We concluded the population characteristics (male, age 18-24, college students) in Dataset-1 were the major reason behind this phenomenon. The answers to “Could you explain why you choose such types of pictures?” in Dataset-1 supported this conjecture: “computer game is something I am interested in” and “computer games picture is personalized to my interests and enjoyable to look at”.

It is obvious that pictures with personally identifiable information may leak personal information. However, it is less obvious that even pictures with no personally identifiable information may provide some clues which may reveal the identity or persona of a device owner. Traditional text-based password does not have this concern as long as the password is kept secure. Previous graphical password schemes, such as Face and PassPoints, do not have this concern either because pictures are selected from a predefined repository.

Finding 2: Gestures on Points of Interest

The security of background draw-a-secret schemes mostly relies on the location distribution of users’ gestures. It is the most secure if the locations of users’ gestures follow a uniform distribution on any picture. However, such passwords would be difficult to remember and may not be preferable by users. By analyzing the collected passwords, we notice that subjects frequently chose standout regions (points of interest, PoIs) on which to draw. As shown in Table 5.1, only 9.8% subjects claimed to choose locations randomly without caring about the background picture. The observation is supported by survey answers to “Could you explain the way you choose locations
to perform gestures?"; “If I have to remember it; it [would] better stand out.” and “Something that would make it easier to remember”.

Even though the theoretical password space of PGA is larger than text-based passwords with the same length, a background picture affects user choice in gesture location, reducing the feasible password space tremendously. We summarize three popular ways that subjects used to identify standout regions: i) finding regions with objects. e.g. “I chose eyes and other notable features” and “I chose locations such as nose, mouth or whole face”; ii) finding regions with remarkable shapes. e.g. “if there is a circle there I would draw a circle around that”; and iii) finding regions with outstanding colors. The detailed distribution of these selection processes is shown in Table 5.1. 60.3% of subjects prefer to find locations where special objects catch their eyes while 22.1% of subjects would rather draw on some special shapes.

**Finding 3: Similarities Across Points of Interest**

We analyzed the attributes of PoIs that users preferred to draw on. We paid more attention to the pictures of people because it was the most popular category. In the 31 registered passwords for the 27 pictures of people uploaded by 22 subjects in Dataset-1, we analyzed the patterns of PoI choice. As shown in Table 5.2, 36 gestures were drawn on eyes and 21 gestures were drawn on noses. Other locations that attracted

### Table 5.2: Attributes of Most Frequently Used PoIs

<table>
<thead>
<tr>
<th>Attributes</th>
<th># Gesture</th>
<th># Password</th>
<th># Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eye</td>
<td>36</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>Nose</td>
<td>21</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>Hand/Finger</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Jaw</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Face (Head)</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
subjects to draw included hand/finger, jaw, face (head), and ear. Interestingly, 19 subjects out of 22 (86.3%) drew on eyes at least once, while 10 subjects (45.4%) performed gestures on noses. The tendencies to choose similar PoIs by different subjects are common in other picture categories as well. Figure 5.2 shows another example where two subjects uploaded two versions of *Starry Night* in Dataset-1. The passwords they chose show strikingly similar patterns with three taps on stars, even if there is no single gesture location overlap.

**Finding 4: Directional Patterns in PGA Password**

Salehi-Abari et al. (2008) suggest many passwords in click-based systems follow some directional patterns. We are interested in whether PGA passwords show similar characteristics. For simplicity, we consider the coordinates of tap and circle gestures as their locations and the middle point of the starting and ending points of line as its location. If the x or y coordinate of a gesture sequence follows a consistent direction regardless of the other coordinate, we say the sequence follows a LINE pattern. We divide LINE patterns into four categories: i) H+, denoting left-to-right \( (x_i \leq x_{i+1}) \); ii) H-, denoting right-to-left \( (x_i \geq x_{i+1}) \); iii) V+, denoting top-to-bottom \( (y_i \leq y_{i+1}) \); and iv) V-, denoting bottom-to-top \( (y_i \geq y_{i+1}) \). If a se-
Table 5.3: Numbers of Gesture-order Patterns

<table>
<thead>
<tr>
<th>Pattern</th>
<th>H+</th>
<th>H-</th>
<th>V+</th>
<th>V-</th>
<th>DIAG</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dataset-1</strong></td>
<td>43</td>
<td>5</td>
<td>16</td>
<td>4</td>
<td>22</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>50.0%</td>
<td>5.8%</td>
<td>18.6%</td>
<td>4.6%</td>
<td>25.5%</td>
<td>20.9%</td>
</tr>
<tr>
<td><strong>Dataset-2</strong></td>
<td>3144</td>
<td>1303</td>
<td>1479</td>
<td>887</td>
<td>2621</td>
<td>3326</td>
</tr>
<tr>
<td></td>
<td>31.3%</td>
<td>12.9%</td>
<td>14.7%</td>
<td>8.8%</td>
<td>26.1%</td>
<td>33.1%</td>
</tr>
</tbody>
</table>

sequence of gestures follows a horizontal pattern and a vertical pattern at the same time, we say it follows a DIAG pattern.

We examined the occurrence of each LINE and DIAG pattern in the collected data. As shown in Table 5.3, more than half passwords in both datasets exhibited some LINE patterns, and a quarter of them exhibited some DIAG patterns. Among four LINE patterns, H+ (drawing from left to right) was the most popular one with 50.0% and 31.3% occurrences in Dataset-1 and Dataset-2, respectively. And, V+ (drawing from top to bottom) was the second most popular with 18.6% and 14.7% occurrences in two datasets, respectively. This finding shows it is reasonable to use gesture-order patterns as one heuristic factor to prioritize generated passwords.

Finding 5: Time Disparity among Different Combinations of Gesture Types

We analyzed all registered passwords to understand the gesture patterns and the relationship between gesture type and input time. For 86 registered passwords (258 gestures) in Dataset-1, 212 (82.1%) gesture types were taps, 39 (15.1%) were lines, and only 7 (2.7%) were circles. However, the corresponding occurrences for 10,039 registered passwords (30,117 gestures) in Dataset-2 were 15,742 (52.2%), 10,292 (34.2%), and 4,083 (13.5%), respectively. Obviously, subjects in Dataset-2 chose more diverse gesture types than subjects in Dataset-1. As shown in Table 5.4, there was
a strong connection between the time subjects spent on reproducing passwords and the gesture types they chose. Three taps, the most common gesture combination, appeared in both datasets with the lowest average time (5.74 seconds and 4.33 seconds in corresponding dataset). On the other hand, the passwords with two circles and one line took the longest average input time (10.19 seconds in Dataset-2). In the user studies, subjects in Dataset-2 were asked to set up the passwords by pretending they were protecting their bank information. However, subjects in Dataset-1 actually used these passwords to access the class materials which they accessed more than four times a week on average. This may be a reason why subjects in Dataset-1 prefer passwords with simpler gesture type combinations that are easier to reproduce in a timely manner.
5.3.3 Memorability and Usability Analysis

The tolerance introduced in PGA is a trade-off between security and usability. In order to quantify this tradeoff, we calculate the distance between input PGA passwords with the registered ones. When the types or directions of gestures do not match, we regard input passwords incomparable with the registered ones. Otherwise, the distance is defined as the average distance of all gestures. We denote the password presented for the \( i \)-th attempt \( \vec{\pi}(i) \) and \( \vec{\pi}(0) \) as the password registered for the same picture.

In the 2,536 login attempts collected in Dataset-1, 422 are unsuccessful in which 146 are type or direction errors and 276 are distance errors. Figure 5.3(a) shows the distance distribution for the password whose distance is less than 10 and the red line denotes the threshold for being classified as successful. The result shows the current setup in our system is quite reasonable to capture most closely presented passwords.

Figure 5.3(b) shows the average time in seconds that subjects spent on registering, confirming, and reproducing passwords. \( x = 1 \) denotes the registration, \( x = 2 \) denotes the confirmation, and all others denote the later login attempts. As we can notice, the average time for the registration is 7.43 seconds while 4.53 seconds are taken for the confirmation. With subjects getting used to the picture password system, the
average time spent for successful logins is reduced to as low as 2.51 seconds. On the other hand, the average time spent on all unsuccessful login attempts is 5.86 seconds.

5.4 Using User Choice Patterns to Attack PGA Passwords

In this section, we present an attack framework on Windows 8™ picture gesture authentication, leveraging the findings addressed in Section 5.3. Our attack framework takes the target picture’s PoIs, a set of learning pictures’ PoIs and corresponding password pairs as input, and produces a list of possible passwords, which is ranked in the descending order of the password probabilities.

Next, we first discuss the attack models followed by the representations of picture password and PoI. We then illustrate the idea of a selection function and its automatic identification. We also present two algorithms for generating a selection function sequence list and describe how it can generate picture password dictionaries for previously unseen target pictures.

5.4.1 Attack Models

Depending on the resources an attacker possesses, we articulate three different attack models: i) *Pure Brute-force Attack*: an attacker blindly guesses the picture password without knowing any information of the background picture and the users’ tendencies. The password space in this model is $2^{30.1}$ in PGA Pace (2011a). ii) *PoI-assisted Brute-force Attack*: an attacker assumes the user only performs drawings on PoIs of the background picture and this model randomly guesses passwords on identified PoIs. The password space for a picture with 20 PoIs in this model is $2^{27.7}$ Pace (2011a). Salehi-Abari et al. Salehi-Abari et al. (2008) designed an approach to automatically identify hot-spots in a picture and generate passwords on them. iii) *Knowledge-based PoI-assisted Attack*: in addition to the assumption for PoI-assisted
brute-force attack, an attacker ought to have some knowledge about the password patterns learned from collected picture and password pairs (not necessarily from the target user or picture). The guessing space in this model is the same as the one in PoI-assisted brute-force attack. However, the generated dictionaries in this model are ranked with the higher possibility passwords on the top of the list.

Attack schemes could also be divided into two categories based on whether or not an attacker has the ability to attack previously unseen pictures. The method presented in Salehi-Abari et al. (2008) is able to attack previously unseen pictures for click-based graphical password. It uses click-order heuristics to generate partially ranked dictionaries. However, this approach cannot be applied directly to background draw-a-secret schemes because the gestures allowed in such schemes are much more complex and the order-based heuristics could not capture users’ selection processes accurately. In contrast, our attack framework could abstract generic knowledge of user choice in picture password schemes. In addition, as a working knowledge-based PoI-assisted model, it is able to generate ranked dictionaries for previously unseen pictures.

5.4.2 Password and PoI Representations

We first formalize the representation of a password in PGA with the definition of a location-dependent gesture which represents a single gesture on some locations in a picture.

**Definition 5.1.** A location-dependent gesture (LdG) denoted as $\pi$ is a 7-tuple $⟨g, x_1, y_1, x_2, y_2, r, d⟩$ that consists of gesture’s type, location, and other attributes.

In this definition, $g$ denotes the type of LdG that must be one of tap, line, and circle. A tap LdG is further represented by the coordinates of a gesture $⟨x_1, y_1⟩$. A
line LdG is denoted by the coordinates of the starting and ending points of a gesture \( \langle x_1, y_1 \rangle \) and \( \langle x_2, y_2 \rangle \). A circle LdG is denoted by the coordinates of its center \( \langle x_1, y_1 \rangle \), radius \( r \), and direction \( d \in \{+, -\} \) (clockwise or not). We define the password space of location-dependent gesture as \( \Pi = \Pi_{\text{tap}} \cup \Pi_{\text{line}} \cup \Pi_{\text{circle}} \). A valid PGA password is a length-three sequence of LdGs denoted as \( \overrightarrow{\pi} \), and the PGA password space could be denoted as \( \overrightarrow{\Pi} \).

A point of interest is a standout region in a picture. PoIs could be regions with semantic-rich meanings, such as face (head), eye, car, clock, etc. Also, they could stand out in terms of their shapes (line, rectangle, circle, etc.) or colors (red, green, blue, etc.). We denote a PoI by the coordinates of its circumscribed rectangle and some describing attributes. A PoI is a 5-tuple \( \langle x_1, y_1, x_2, y_2, D \rangle \), where \( \langle x_1, y_1 \rangle \) and \( \langle x_2, y_2 \rangle \) are the coordinates of the top-left and bottom-right points of the circumscribed rectangle, and \( D \subseteq 2^D \) is a set of attributes that describe this PoI. \( D \) has three sub-categories \( D_o, D_s \) and \( D_c \) and four wildcards \( *_o, *_s, *_c, \) and \( * \), where \( D_o = \{ \text{head, eye, nose, } \ldots \} \), \( D_s = \{ \text{line, rectangle, circle, } \ldots \} \), and \( D_c = \{ \text{red, blue, yellow, } \ldots \} \). Wildcards are used when no specific information is available. For example, if a PoI is identified with objectness measure Alexe et al. (2012) that gives no semantics about the identified region, we mark the PoI’s describing attribute as \( * \).

### 5.4.3 Location-dependent Gesture Selection Functions

A key concept in our framework is the location-dependent gesture selection function (LdGSF) which models and simulates the ways of thinking that users go through when they select a gesture on a picture. The motivation behind this abstraction is that the set of PoIs and their locations differ from picture to picture, but the ways that users think to choose locations for drawing a gesture exhibit certain patterns.
A location-dependent gesture selection function (LdGSF) is a mapping \( s : G \times 2^P \times 2^D \times \Theta \rightarrow 2^\Pi \) which takes a gesture, two sets of PoI attributes, and a set of PoIs in the learning picture as input to produce a set of location-dependent gestures.

The universal set of LdGSF is defined as \( S \). A length-three sequence of LdGSF is denoted as \( \vec{s} \), and a set of length-three LdGSF sequences is denoted as \( \vec{S} \). \( s(\text{tap}, \{\text{red}, \text{apple}\}, \emptyset, \theta_k) \) is interpreted as ‘tap a red apple in the picture \( p_k \)’ and \( s(\text{circle}, \{\text{head}\}, \emptyset, \theta_k) \) as ‘circle a head in \( p_k \)’. Note that, no specific information of the locations of ‘red apple’ and ‘head’ is provided here which makes the representations independent from actual locations of objects in the picture.

One challenge we face is some PoIs may be big enough to take several unique gestures. Let us consider a picture with a big car image in it. Simply saying ‘tap a car’ could result in lots of distinct tap gestures in the circumscribed rectangle of the car. One solution to this problem is to divide the circumscribed rectangle into a grid with the scale of toleration threshold. However, this solution would result

This conjecture is supported by our observations from collected data and surveys discussed in Section 5.3. With the help of LdGSF, the PoIs and corresponding passwords in training pictures are used to generalize picture-independent knowledge that describes how users choose passwords.

Definition 5.2. A location-dependent gesture selection function (LdGSF) is a mapping \( s : G \times 2^P \times 2^D \times \Theta \rightarrow 2^\Pi \) which takes a gesture, two sets of PoI attributes, and a set of PoIs in the learning picture as input to produce a set of location-dependent gestures.
in too many password entries in the generated dictionary. For simplicity, we introduce five inner points for one PoI, namely center, top, bottom, left, and right that denote the center of the PoI and four points of the center of two consecutive corners. Any gesture that falls into the proximities of these five points of a PoI would be considered as an action on this PoI. For some PoIs that are big enough to take an inner line gesture, we put ∅ as the input of the second set of PoI attributes. $s(line, \{mouth\}, \emptyset, \theta_k)$ denotes ‘line from the left(right) to the right(left) on the same mouth’. While, $s(line, \{mouth\}, \{mouth\}, \theta_k)$ means ‘connect two different mouths’.

Figure 5.4 shows an example demonstrating how LdGSF simulates a user’s selection processes that were taken from Pace (2011b). In reality, a user’s selection process on a PoI and gesture selection may be determined by some subjective knowledge and cognition. For example, ‘circle my father’s head’ and ‘tap my mother’s nose’ may involve some undecidable computing problems. One solution to handle this issue is to approximate subjective selection processes in objective ways by including some modifiers. ‘circle my father’s head’ may be transformed into ‘circle the uppermost head’ or ‘circle the biggest head’. However, it is extremely difficult, if not impossible, to accurately approximate subjective selection processes in this way, and it may bring serious over-fitting problems in the learning stage. Instead, we choose to ignore subjective information by abstracting ‘circle my father’s head’ to ‘circle a head’. A
drawback of this abstraction is that an LdGSF may return more than one LdG and we have no knowledge to rank them directly, as they come from the same LdGSF. Using Figure 5.4(a) as an example, ‘circle a head’ outputs four different LdGs on each head in the picture. The LdGSF sequence shown in Figure 5.4(c) generates $4 \times (4 \times 3) \times 4 = 192$ passwords. To cope with this issue, we use gesture-order to rank the passwords generated by the same LdGSF sequence that will be detailed in Section 5.4.5. Next, we present an automated approach to extract users’ selection processes from the collected data and represent them with LdGSFs.

Figure 5.5 shows an example demonstrating that how to extract users’ selection processes from PoIs automatically. First, PoIs in the background picture are identified using mature computer vision techniques such as face detection and recognition Zhang and Li (2010, 2012), object detection Canny (1986), and objectness measure Alexe et al. (2012). Then, each LdG in a password is compared with PoIs based on their coordinates and sizes. If a match between PoIs and LdGs is found, a new LdGSF is created as the combination of the LdG’s gesture type and PoI’s attributes. For instance, the location and size of LdG 1 in Figure 5.5(c) matches PoI 2 in Figure 5.5(b) (the locations of the circle gesture and PoI center are compared first; then, the radius of the circle is compared with 1/2 of PoI’s height and width). Then, an LdGSF $s(\text{circle}, \{\text{head}\}, \emptyset)$ is created which is equivalent to the LdG shown in Figure 5.4(c).

To choose a password in PGA, the user selects a length-three LdGSF sequence. With the definition of LdGSF, the generation of ranked password list is simplified into the generation of the ranked LdGSF sequence list. Let $\text{order}: \vec{S} \rightarrow \{1..|\vec{S}|\}$ be a bijection which indicates the order LdGSF sequences should be performed. The objective of generating ranked LdGSF sequence list is to find such a bijection.
5.4.4 LdGSF Sequence List Generation and Ordering

Now we present our approach to find the aforementioned bijection that indicates the order that the LdGSF sequences should be performed on a target picture for generating the password dictionary. Our framework is not dependent on certain rules, but is adaptive to the tendencies shown by users who participate in the training set. The characteristic of adaptiveness helps our framework generate dedicated guessing paths for different training data. Next, we present two algorithms for obtaining such a feature.

**BestCover LdGSF Sequence List Generation**

We first propose an LdGSF sequence list generation algorithm named **BestCover** that is derived from $B_{\text{mssc}}$ Feige et al. (2004) and $B_{\text{ems}}$ Zhang et al. (2010). The objective of **BestCover** LdGSF sequence list generation is to optimize the guessing order for the sequences in the list by minimizing the expected number of sequences that need to be tested on a random choice of picture in the training dataset.

The problem is formalized as follows: **Instance**: The collection of LdGSF sequences $\vec{s}_1, ..., \vec{s}_n$ and corresponding picture password $\vec{\pi}_1, ..., \vec{\pi}_n$, for which $\vec{s}_i(\vec{\theta}_i) \ni \vec{\pi}_i$, $i \in \{1..n\}$ and $\vec{\theta}_1, .., \vec{\theta}_n$ are the sets of PoIs in pictures $p_1, .., p_n$. **Question**: Expected Min Selection Search (**emss**): The objective is to find order so as to minimize $\mathbb{E}(\min\{i : \vec{s}_i(\vec{\theta}_r) \ni \vec{\pi}_r\})$, where $\vec{s}_i = \text{order}^{-1}(i)$ and the expectation is taken with respect to a random choice of $r \leftarrow \{1..n\}$. We use $\text{cover}_{\text{emss}}(k) = \min_{\vec{s}, \vec{s}(\vec{\theta}_k) \ni \vec{\pi}_k}(\text{order}_{\text{emss}}(\vec{s}))$ to compute the number of required guesses to break $\vec{\pi}_k$. Therefore, $\mathbb{E}(\min\{i : \vec{s}_i(\vec{\theta}_r) \ni \vec{\pi}_r\}$ is equivalent to $\mathbb{E}(\text{cover}_{\text{emss}}(r))$.

The hardness of this problem is that different LdGSFs and LdGSF sequences may generate the same list of LdGs and passwords. For instance, ‘tap a red object’ and
An overlap in different LdGSF results is similar to the coverage characteristics in the set cover problem. We can prove the NP-hardness of \texttt{emss} by reducing from \texttt{mssc} \citet{Feige2004, Zhang2010}. Min Sum Set Cover (\texttt{mssc}) is formalized as follows: Given is a set \( U \) and a collection \( \mathcal{C} \) of subsets of \( U \) where \( \bigcup_{C \in \mathcal{C}} C = U \). Let \( \text{\texttt{order}}_{\text{mssc}} : \mathcal{C} \to \{1..|\mathcal{C}|\} \) be a bijection, and let \( \text{\texttt{cover}}_{\text{mssc}} : U \to \{1..|\mathcal{C}|\} \) be defined by \( \text{\texttt{cover}}_{\text{mssc}}(j) = \min_{C \ni j} (\text{\texttt{order}}_{\text{mssc}}(C)) \). The problem is called min sum, because the object is to minimize \( \sum_{j \in U} \text{\texttt{cover}}_{\text{mssc}}(j) \).

Given any instance \((U, \mathcal{C})\) of \texttt{mssc}, denote \( U = \{1..n\} \). We create a set of PoIs \( \theta_j \) and a picture password \( \pi_j \) for each \( j \in U \). \( \theta_j \) and \( \pi_j \) must be different from \( \theta_k \) and \( \pi_k \) respectively for any \( k \neq j \). For each \( C \in \mathcal{C} \), we create an LdGSF sequence \( s_C \) such that \( s_C(\theta_j) \ni \pi_j \) if \( j \in C \) and such that \( s_C(\theta_j) = \phi \) if \( j \not\in C \). We can always construct such an LdGSF sequence for each \( C \) by combining all \( \theta_j, j \in C \) as a new PoI type in a wildcard representation. The set \( \tilde{S} \) consists of the set of \( s_C \) for different \( C \). Set \( \text{\texttt{order}}_{\text{mssc}}(C) \leftarrow \text{\texttt{order}}_{\text{emss}}(s_C) \), then

\[
\mathbb{E}(\text{\texttt{cover}}_{\text{emss}}(r)) = \sum_{i=1}^{n} i \times Pr(\text{\texttt{cover}}_{\text{emss}}(r) = i)
\]

The number of picture passwords that are cracked for the first time at the \( i \)th guess divided by the total number of picture passwords

\[
= \sum_{i=1}^{n} i \times \frac{|k \in \{1..n\} : \text{\texttt{cover}}_{\text{emss}}(k) = i|}{n}
\]

\[
= \sum_{i=1}^{n} i \times \frac{|j \in U : \text{\texttt{cover}}_{\text{mssc}}(j) = i|}{n}
\]

\[
= \sum_{j \in U} \frac{\text{\texttt{cover}}_{\text{mssc}}(j)}{n}
\]

Therefore, \( \text{\texttt{order}}_{\text{emss}} \) minimizes \( \mathbb{E}(\text{\texttt{cover}}_{\text{emss}}(r)) \) if and only if \( \text{\texttt{order}}_{\text{mssc}} \) minimizes
\[ \sum_{j \in U} \text{cover}_{\text{mssc}}(j). \] We give an approximation algorithm for \( \text{emss} \) in Algorithm 8 that is a modification from \( \mathcal{B}_{\text{mssc}} \) Feige et al. (2004) and \( \mathcal{B}_{\text{emts}} \) Zhang et al. (2010). The time complexity of \textbf{BestCover} is \( O(n^2 + |\vec{S}'| \log(|\vec{S}'|)) \).

Algorithm 3: \textbf{BestCover}(\((\vec{s}_1, \ldots, \vec{s}_n), (\vec{\pi}_1, \ldots, \vec{\pi}_n)\))

1 for \( i = 1..n \) do
2 \( T_{\vec{s}_i} \leftarrow \{ k : \vec{s}_i(\theta_k) \ni \vec{\pi}_k \} \);  
3 end

4 \( \vec{S}' \leftarrow \{ \vec{s} : |T_{\vec{s}}| > 0 \} \);  
5 for \( i = 1..|\vec{S}'| \) do
6 \( \text{order}^{-1}(i) \leftarrow \vec{s}_k \), that \( T_{\vec{s}_k} \) has most elements that are not included in \( \bigcup_{i' < i} \text{order}^{-1}(i') \);  
7 end
8 return \( \text{order} \)

\textbf{BestCover} is good for a training dataset that consists of comprehensive and large scale password samples, because it assumes the target passwords exhibit same or at least very similar distributions to the training data. However, if the training dataset is small and biased, the results from \textbf{BestCover} may over-fit the training data and fail in testing data.

\textbf{Unbiased LdGSF Sequence List Generation}

The over-fitting problem in \textbf{BestCover} is brought about by the biased PoI attribute distributions in training data. For example, we have a training set with 9 pictures of apples and 1 picture of a car, and 5 corresponding passwords have circles on apples and 1 has a circle on car. In the generated LdGSF sequence list, \textbf{BestCover} will put sequences with ‘circle an apple’ prior to the ones with ‘circle a car’, because the
former ones have an LdGSF that was used in more passwords. However, we can see the probability for users to circle car (1/1) is higher than apples (5/9) if we consider the occurrences of apple and car in pictures.

**Unbiased** LdGSF sequence list generation copes with this issue by considering the PoI attribute distributions. It removes the biases from the training dataset by normalizing the occurrences of LdGSFs with the occurrences of their corresponding PoIs. Let $D_{s_k} \subseteq \theta$ denote the event that $\theta$ contains enough PoIs that have attributes specified in $s_k$. If a PoI with a specific type of attributes does not exist in a picture, the probability that a user select the PoI with such an attribute on this picture to draw a password is 0, denoted as $Pr(s_k|D_{s_k} \subseteq \theta) = 0$, e.g. a user would not think and perform ‘tap a red apple’ on a picture without the existence of the red apple. We assume each LdGSF in a sequence is independent of each other and approximately compute $Pr(s_k|D_{s_k} \subseteq \theta)$ with Equation 5.1.

$$Pr(s_k|D_{s_k} \subseteq \theta) = Pr(s_1s_2s_3|D_{s_1} \subseteq \theta \land D_{s_2} \subseteq \theta \land D_{s_3} \subseteq \theta)$$

$$= Pr(s_1|D_{s_1} \subseteq \theta) \times Pr(s_2|D_{s_2} \subseteq \theta) \times Pr(s_3|D_{s_3} \subseteq \theta)$$

(5.1)

For each $s_i \in S$, we compute $Pr(s_i|D_{s_i} \subseteq \theta)$ with Equation 5.2:

$$Pr(s_i|D_{s_i} \subseteq \theta) = \frac{\sum_{j=1}^{n} count(D_{s_i}, \pi_j)}{\sum_{j=1}^{n} count(D_{s_i}, \theta_j)}$$

(5.2)

where $\sum_{j=1}^{n} count(D_{s_i}, \pi_j)$ denotes the number of LdGs in passwords of the training set that share the same attributes with $s_i$, and $\sum_{j=1}^{n} count(D_{s_i}, \theta_j)$ denotes the number of PoIs in the training set that share the same attributes with $s_i$. $Pr(s_i|D_{s_i} \subseteq \theta)$ describes the probability of using a certain LdGSF when there are enough PoIs with the required attributes.
The Unbiased algorithm generates an LdGSF sequence list by ranking \( Pr(\bar{s}_k|D_{\bar{s}_k} \subseteq \theta) \) instead of \( Pr(s_k) \) in descending order as shown in Algorithm 10. The time complexity of Unbiased is \( O(n|S| + |\bar{S}| \log(|\bar{S}|)) \). The Unbiased algorithm would be better for the scenarios where fewer samples are available or samples are highly biased.

\[ \text{Algorithm 4: Unbiased}(S) \]

1. for \( s \in S \) do
   2. Compute \( Pr(s|D_s \subseteq \theta) \) with Equation 5.2;
3. end

4. for \( \bar{s} \in \bar{S} \) do
   5. Compute \( Pr(\bar{s}|D_{\bar{s}} \subseteq \theta) \) with Equation 5.1;
6. end

7. for \( i = 1..|\bar{S}| \) do
   8. \( \text{order}^{-1}(i) \leftarrow \bar{s}_k \), that \( Pr(\bar{s}_k|D_{\bar{s}_k} \subseteq \theta) \) holds the \( i \)-th position in the descending ordered \( Pr(\bar{s}|D_{\bar{s}} \subseteq \theta) \) list;
9. end
10. return order

5.4.5 Password Dictionary Generation

The last step in our attack framework is to generate the password dictionary for a previously unseen target picture. First, the PoIs in the previously unseen picture are identified. Then, a dictionary is acquired by applying the LdGSF sequences on the PoIs, following the order created by the BestCover or Unbiased algorithm. Obviously, the passwords generated by an LdGSF sequence that holds a higher position in the LdGSF sequence list will also be in higher positions in the dictionary. However, as addressed earlier, BestCover and Unbiased algorithms do not provide extra information
to rank the passwords generated by the same LdGSF sequence. Inspired by using the click-order patterns as the heuristics for dictionary generation Salehi-Abari et al. (2008), we propose to rank such passwords generated by the same LdGSF sequence with gesture-orders. In the training stage, we record the gesture-order occurrence of each LINE and DIAG pattern and rank the patterns in descending order. In the attack stage, for the passwords generated by the same LdGSF sequence, we reorder them with their gesture-orders in the order of LINE and DIAG patterns. Passwords that do not belong to any LINE or DIAG pattern hold lower positions.

5.5 Implementation

In this section, we discuss the implementation details of our proof-of-concept system.

5.5.1 PoI Identification

We chose OpenCV Intel (2014) as the computer vision framework for our implementation and collected several feature detection tools for automatically identifying PoIs in background pictures. The computer vision techniques we adopted include: i) object detection: the goal of object detection is to find the locations and sizes of semantic objects of a certain class in a digital image. Viola-Jones object detection framework Viola and Jones (2004) is the first computationally affordable online object detection framework that utilizes Haar-like features instead of image intensities. Each learned classifier is represented and stored as a haar cascade. We collected 30 proven haar cascades from Reimondo (2008) for 8 different object classes including face (head), eye, nose, mouth, ear, head, body, and clock. ii) low-level feature detection: due to the high positive and high negative rates of object detection, we also resorted to some low-level feature detection algorithms that identify standout
Figure 5.6: PoI Identification on Example Pictures in Dataset-2: (a) Original Pictures (b) Circle Detection with Hough Transform (c) Contour Detection (d) Objectness Measure (e) Object Detection

regions without extracting semantics. To identify regions whose colors are different from their surroundings, we first converted the color pictures to black and white, then found the contours using algorithms in Suzuki et al. (1985). For the circle detection, we used Canny edge detector Canny (1986) and Hough transform algorithms Ballard (1981). iii) objectness measure: objectness measure Alexe et al. (2012) deals with class-generic object detection. Different from detecting objects in a specific class, the objectness measure finds the locations and sizes of class-generic objects whose colors and textures are opposed to the background images. Objectness measure could be considered as a technique combining several low-level feature detectors together. We used an objectness measure library from Alexe et al. (2013) that is able to locate objects and give numerical confidence values with its results.

Figure 5.6 displays the PoI detection results on four example pictures in Dataset-2. As we can see in Figure 5.6(b), circle detection could identify both bicycle wheels and car badge, but its false positive rate is a little high. Contour detection is the
most robust algorithm with a low false positive rate which could locate regions whose colors are different as shown in Figure 5.6(c). Objectness measure shown in Figure 5.6(d) could also identify regions whose colors and textures are different from their surroundings. Since most haar cascades we used are designed for facial landmarks, they work smoothly on portraits as does the second picture in Figure 5.6(e). However, the results show relatively high false positive rates on pictures from other categories. In order to identify more PoIs as accurate as possible, our approach in PoI identification leveraged two steps. In the first step, all possible PoIs were identified using different kinds of tools. In the second step, we examined all identified PoIs and removed duplicates by comparing their locations, sizes and attributes. Then, our approach generated a PoI set called $P^1_{A-40}$ and $P^2_{A-40}$ for each picture in Dataset-1 and Dataset-2, respectively. Those PoI sets consisted of at most 40 PoIs with the highest confidences.

Since our attack algorithms are independent from the PoI identification algorithms, we are also interested in examining how our attack framework performs with ideal PoI annotations for pictures. Besides using the automated PoI identification techniques, we manually annotated pictures in Dataset-2 for some outstanding PoIs as well. To annotate the pictures, we simply recorded the locations and attributes of at most fifteen most appealing regions in the pictures without referring to any password in the collected dataset. We call this annotated PoI set $P^2_{L-15}$.

5.5.2 LdGSF Identification

We discuss the identified LdGSFs by linking PoIs and passwords in Dataset-2 with the help of two PoI sets $P^2_{L-15}$ and $P^2_{A-40}$ using our LdGSF identification algorithm discussed in Section 5.4.3. The results from $P_L$ are closer to users’ actual selection processes, while the results from $P_A$ are the best approximations to users’ selection
processes we could get in a purely automated way with state-of-the-art computer vision techniques.

Table 5.5: Top 10 Identified LdGSFs Using $P_{L-15}^2$

| Rank | $Pr(s_k)$                  | $Pr(s_k|D_{s_k} \subseteq \theta)$ |
|------|---------------------------|-----------------------------------|
| 1    | (tap, {head}, \emptyset)  | (tap, {nose}, \emptyset)         |
| 2    | (tap, {*}_c, \emptyset)   | (tap, {mouth}, \emptyset)        |
| 3    | (tap, {circle}, \emptyset)| (tap, {circle}, \emptyset)       |
| 4    | (tap, {eye}, \emptyset)   | (tap, {eye}, \emptyset)          |
| 5    | (circle, {head}, \emptyset)| (tap, {*}_c, \emptyset)          |
| 6    | (tap, {nose}, \emptyset)  | (tap, {head}, \emptyset)         |
| 7    | (circle, {circle}, \emptyset) | (circle, {circle}, \emptyset) |
| 8    | (circle, {eye}, \emptyset) | (tap, {ear}, \emptyset)          |
| 9    | (line, {*}_c, {*}_c)      | (line, {mouth}, {mouth})         |
| 10   | (line, {eye}, {eye})      | (tap, {forehead}, \emptyset)     |

The top ten identified LdGSFs using $P_{L-15}^2$ are shown in Table 5.5 ordered by their $Pr(s_k)$ and $Pr(s_k|D_{s_k} \subseteq \theta)$. It also suggests that ‘tap a head’ is found the most times in the passwords, while ‘tap a nose’ is the most popular one when there is a nose in the picture. The result seems unreasonable at the first glance since there is always a nose in a head. Actually, it is because if the head in the picture is really small, we simply annotate the circumscribed rectangle as head instead of marking the inner rectangles with more specific attributes. Table 5.5 indicates that gestures on human organs are the most popular selection functions adopted by subjects.

The top ten identified LdGSFs using $P_{A-40}^2$ are shown in Table 5.6. By comparing Table 5.5 and Table 5.6, we could notice differences caused by using annotated PoI set and automated detected PoI set. The fact that $s(tap, \{\ast\}, \emptyset)$ is among the top ten LdGSFs is an indicator that the automatic PoI identification could not classify
many PoIs and simply mark them as *. It is surprising to find out there are two LdGs on clock in top ten ordered by \( Pr(s_k|D_{sk} \subseteq \theta) \) at first, because there is no clock in any picture in Dataset-2. The closest guess is OpenCV falsely identified some circle shape objects as clocks, but the number is not very big since there is no LdG on a clock in the top ten ordered by \( Pr(s_k) \).

### 5.6 Attack Evaluation

In this section, we present the evaluation results of our framework for nontargeted attacks and targeted attacks.

#### 5.6.1 Nontargeted Attack Evaluation

In order to attack passwords from a previously unseen picture, the training dataset excluded passwords from the target picture. More specifically, to evaluate Dataset-1 (58 unique pictures), we used passwords from 57 pictures as the training data and at-
Figure 5.7: (a) Percentage of passwords cracked vs. number of password guesses, per condition. (b) Percentage of LdGs cracked vs. number of password guesses, per condition. For Dataset-1, there are 86 passwords that include 258 LdGs. For Dataset-2, there are 10,039 passwords that have 30,117 LdGs.

Tackled the passwords for the last picture. To evaluate Dataset-2 (15 unique pictures), we used passwords for 14 pictures as training data, learned the patterns exhibited in the training data, and generated a password dictionary for the last picture. The same process was carried out 58 and 15 times for Dataset-1 and Dataset-2, respectively, in which the target picture was different in each round. The size of the dictionary was set as $2^{19}$ which is 11-bit smaller than the theoretical password space. We compared all collected passwords for the target picture with the generated dictionary for the picture, and recorded the number of password guesses.

Nontargeted attacks also require that the training dataset does not include previous passwords from the targeted user. However, it turns out very time-consuming to
perform strict nontargeted attacks on our Dataset-2. Instead, in our analyses, training password datasets include a very small number of passwords from the targeted subject. More specifically, in our experiment there were around 9,400 training passwords for which only 14 came from the targeted user. Even though this may affect the results, we believe it is less influential. Since all training passwords were treated equally, the influence brought by the 0.14% training data is low.

**Offline Attacks**

Due to the introduction of a tolerance threshold, picture passwords may be more difficult to store securely compared with text-based passwords that are normally saved after salted hashing. Even though the approach that Windows 8™ is adopting to store picture passwords remains undisclosed, we could consider two attack scenarios where picture passwords are prone to offline attacks. In the first scenario, all passwords which fall into the vicinity (defined by the threshold) of chosen passwords could be stored in a file with salted hashes for comparison. An attacker who has access to this file could perform offline dictionary attacks like cracking text-based password systems. In the second scenario, picture passwords could be used for other purposes besides logging into Windows 8™, where no constraint on the number of attempts is enforced. For example, a registered picture password could be transformed and used as a key to encrypt a file. An attacker who acquires the encrypted file would like to perform an offline attack.

In order to attack passwords from a previously unseen picture, the training dataset excluded passwords from the target picture. More specifically, to evaluate Dataset-1 (58 unique pictures), we used passwords from 57 pictures as the training data and attacked the passwords for the last picture. To evaluate Dataset-2 (15 unique pictures), we used passwords for 14 pictures as training data, learned the patterns exhibited in
Figure 5.8: (a) Percentage of passwords cracked vs. number of password guesses, per condition. (b) Percentage of LdGs cracked vs. number of password guesses, per condition. Only the first chosen password by each subject in Dataset-2 was considered. There are 762 passwords that have 2,286 LdGs.

the training data, and generated a password dictionary for the last picture. The same process was carried out 58 and 15 times for Dataset-1 and Dataset-2, respectively, in which the target picture was different in each round. The size of the dictionary was set as $2^{19}$ which is 11-bit smaller than the theoretical password space. We compared all collected passwords for the target picture with the generated dictionary for the picture, and recorded the number of password guesses.

The offline attack results within $2^{19}$ guesses in different settings are shown in Figure 5.7. There are 86 passwords in Dataset-1, which have a total of 258 LdGs. And 10,039 passwords were collected in Dataset-2, containing a total of 30,117 LdGs. For Dataset-1, BestCover cracks 42 (48.8%) passwords out of 86 while Unbiased cracks 40
Figure 5.9: (a) Percentage of passwords cracked vs. number of password guesses, per condition. (b) Percentage of LdGs cracked vs. number of password guesses, per condition. Only passwords for pictures 243, 1116, 2057, 4054, 6467, and 9899 were considered. There are 4,003 passwords that have 12,009 LdGs.

(46.5%) passwords for the same dataset with $P_{A-40}^1$. For Dataset-1, 178 LdGs (68.9%) out of 258 are cracked with Unbiased and 171 (66.2%) are broken with BestCover. On the other hand, Unbiased with $P_{L-15}^2$ breaks 2,953 passwords (29.4%) out of 10,039 for Dataset-2. This implies Unbiased with $P_{A-40}^2$ cracking 2,418 passwords (24.0%) is the best result for all purely automated attacks on Dataset-2. As Figure 5.7 suggests, BestCover outperforms Unbiased slightly when ample training data is available. The better performance of both algorithms on Dataset-1 is because the password gesture combinations in Dataset-1 are relatively simpler than the ones in Dataset-2 as we discussed in Section 5.3.2.

In Dataset-2, subjects may not choose all 15 passwords with the same care as
they were eager to finish the process. To reduce this effect, we ran another analysis in which only the first chosen password by each subject was considered. There are 762 passwords that have 2,286 LdGs. Like previous analysis, the training dataset excluded passwords from the target picture. As shown in Figure 5.8, results of this analysis are not as good as previous ones. Unbiased with $P^2_{L-15}$ breaks 160 passwords (21.0%) out of 762. Unbiased with $P^2_{A-40}$ cracking 123 passwords (16.1%). BestCover cracks 108 (14.2%) and 116 (15.2%) with $P^2_{L-15}$ and $P^2_{A-40}$, respectively.

Since some pictures in Dataset-2 are similar, we ran an additional analysis in which only passwords for pictures 243 (airplane), 1116 (portrait), 2057 (car), 4054 (wedding), 6467 (bicycle), and 9899 (dog) were considered. There are 4,003 passwords that have 12,009 LdGs. Unbiased with $P^2_{L-15}$ breaks 1,147 passwords (28.6%) while 803 passwords (20.1%) are cracked by Unbiased with $P^2_{A-40}$. BestCover cracks 829 (20.7%) and 875 (21.8%) with $P^2_{L-15}$ and $P^2_{A-40}$ respectively. Results of this analysis are not as good as results with passwords from all pictures.

**Online Attacks**

The current Windows 8 allows five failure attempts before it forces users to enter their text-based passwords. Therefore, breaking a password under five guesses implies the feasibility for launching an online attack. Figure 5.10 shows a refined view of the number of passwords and LdGs cracked with the first five guesses per condition. Purely automated attack Unbiased with $P^2_{A-40}$ breaks 83 passwords (0.8%) with the first guess and cracks 94 passwords (0.9%) within the first five guesses, while BestCover with $P^2_{A-40}$ cracked 20 passwords (0.2%) for the first guess and 38 passwords (0.4%) within five guesses. Additionally, Unbiased with $P^2_{A-40}$ breaks 1,723 LdGs (5.7%) with the first guess. With the help of manually labeled PoI set $P^2_{L-15}$, the results are even better. For example, Unbiased breaks 195 passwords (1.9%) for the first guess and
266 (2.6%) within the first five guesses. In the meantime, Unbiased with $P^2_{L-15}$ breaks 3,022 LdGs (10.0%) with the first guess and 4,090 LdGs (13.5%) with five guesses.

Effects of Training Data Size

In Figure 5.11, we show the password and LdG cracking results with different sizes of training datasets. For each algorithm, we used $P^2_{A-40}$ as the PoI set and performed three analyses with 60, 600, and all available passwords (about 9,400) as training data, respectively. The sizes of 60 and 600 represent two cases: i) a training set (60) is ten times smaller than the target set (about 669); and ii) a training set (600) is almost the same size as the target set (about 669). For training datasets with the sizes of 60 and 600, we randomly selected these training passwords and performed each analysis three times to get the averages and standard deviations.

As Figure 5.11 shows, BestCover with 60 training samples could only break an average of 888 passwords (8.8%) out of 10,039. And the standard deviation is as strong as 673. While Unbiased with 60 training samples can crack 2,352 passwords (23.4%) that is almost the same as the results generated from all available training samples. Also, the standard deviation for three trials is as low as 62. The results from
Figure 5.11: (a) Average number of passwords cracked vs. different training data sizes. (b) Average number of LdGs cracked vs. different training data sizes. $P^2_{A-40}$ is used for this analysis. Average over 3 analyses, with one standard deviation shown.

BestCover with 600 training samples are much better than the counterparts with 60 training samples. All these observations are expected as Unbiased could eliminate the biases considered in BestCover. The results clearly demonstrate the benefit of using the Unbiased algorithm when a training dataset is small.

Effects on Different Picture Categories

We measured the attack results on different picture categories as shown in Figure 5.12 where each subfigure depicts the number of passwords cracked versus the number of password guesses. Each curve in a subfigure corresponds to a picture as shown in the legend. Our approach cracks more passwords for a picture, if the curve is skewed upward. And the cracking is faster (with fewer guesses), if the curve is leaned toward the left.

Figure 5.12(a) provides a view of the attack results on target pictures 243 and 316, each of which has only one airplane flying in the sky. Fewer PoIs in these two pictures make subjects choose more similar passwords. Unbiased with $P^2_{A-40}$ breaks 261 passwords (39.0%) for the picture 243 and 209 (31.2%) for the picture 316. The cracking success rates are much higher than the average success rate in Dataset-2.
under the same condition. Note that the size of generated dictionaries for these two pictures are smaller than $2^{19}$ due to the number of available PoIs.

In Figure 5.12(b), we show the results on two portrait pictures where Unbiased with $P_{A-40}^2$ cracks 389 passwords (29.0%) for both in total. The attack success rate is much higher than the average success rate in Dataset-2. This is due to the fact that state-of-the-art computer vision algorithms work well on facial landmarks and subjects’ tendencies of drawing on these features are high. The results show that passwords on simple pictures with fewer PoIs or portraits, for which state-of-the-art computer vision techniques could detect PoIs with high accuracy, are easier for attackers to break.

Figure 5.12(c) shows the attack results on 5 pictures of people. Some of these pictures only have very small figures of people and others have larger figures but not big enough to be considered as a portrait. Unbiased with $P_{A-40}^2$ cracks 726 passwords (21.7%) for these 5 pictures in total, which is lower than the average success rate in Dataset-2.

Figure 5.12(d) shows the attack results on 4 miscellaneous pictures, two of which are bicycle pictures and the other two are car pictures. The picture, 6412.jpg, has a bicycle leaning against the wall. Different colors on the bicycle and wall in this picture make it cluttered and have lots of PoIs. Unbiased with $P_{A-40}^2$ only cracks 68 passwords (10.1%) for this picture. However, Unbiased with $P_{A-40}^2$ cracked 458 (17.1%) for all 4 pictures.

Performance

We also evaluated the performance of our attack approach. Our analyses were carried out on a computer with dual-core processor and 4GB of RAM. In Figure 5.13, we show the average runtime for our algorithms to order the LdGSF sequences and
generate dictionary for a picture in Dataset-2. Each bar represents the average time in seconds over 15 pictures with the standard deviation using different algorithms and PoI sets. The results show that BestCover is much faster than Unbiased under the same condition. The average runtime for BestCover on $P_A^{2}A_{-40}$ to order LdGSF sequences is only 0.06 seconds and to generate a dictionary is 2.68 seconds, while Unbiased spends 18.36 and 3.96 seconds, respectively. As we analyzed in Section 5.4.4, such a difference is caused by the complexity of each algorithm. With such a prompt response, BestCover could be used for online queries.
Figure 5.13: Average runtime in seconds to order LdGSF sequences using BestCover and Unbiased. Average over 15 pictures in Dataset-2 with one standard deviation shown.

5.6.2 Targeted Attack Evaluation

In this section, we present the evaluation results of our framework for targeted attacks. Because most subjects in Dataset-1 only chose one password, Dataset-1 was excluded from these experiments. We only use the passwords of the subjects who chose two or more passwords in Dataset-2 in these experiments. There are 697 subjects who fall into this pattern resulting in 9,974 passwords. For each of the 697 subjects, we use one of her passwords as the target and the rest of her passwords as training data set to build the model. The average size of training data sets is around 13, which is significantly smaller than the size used, which is around 9,400, in nontargeted attacks. A dictionary is generated in this way for each target password per user. Since each subject only chose one password for each picture, a training data set does not include passwords for the target picture. We recorded the number of password guesses when a password is cracked. Then, we cumulated the results for each user and each target password together in a single figure as illustrated in Figure 5.14.

The offline attack results within $2^{19}$ guesses in different settings are shown in Figure 5.14(a). Unbiased with $P^2_{L-15}$ breaks 2,233 passwords (22.4%) out of 9,974.
Figure 5.14: Offline attacks. There are 9,974 passwords from 697 accounts in this experiment. The average size of training data sets is around 13. (a) Percentage of passwords cracked vs. number of password guesses, per condition. (b) Passwords cracked per account. Each horizontal bar represents a condition. Regions within each bar show the fraction of accounts for which the indicated number of passwords were cracked.

Unbiased with $P_{A-40}^2$ breaks 2,123 passwords (21.3%) out of 9,974. Even though the results are a little bit lower than nontargeted attacks, we should take the significantly smaller training data set sizes into account. In nontargeted attack, the training data size is around 9,400 passwords. However, in targeted attack, the training data sizes range from at least 1 password to at most 14 passwords with an average of 13. In other word, targeted attacks using Unbiased algorithms with around 100 times smaller training data set could achieve almost the same results as nontargeted attacks. BestCover with $P_{L-15}^2$ and $P_{A-40}^2$ breaks 1,096 (10.9%) and 898 (9.0%) passwords,
respectively. Due to the small training data size, the results from BestCover for nontargeted attacks are quite lower than the counterparts for targeted attacks.

For online attacks within 5 guesses that are shown in the left-lower corner of Figure 5.14(a), Unbiased with $P^2_{L-15}$ breaks 434 passwords (4.4%) out of 9,974, and the first guesses could even break 380 (3.8%) passwords. Unbiased with $P^2_{A-40}$ breaks 77 passwords (0.7%) out of 9,974. BestCover with $P^2_{L-15}$ breaks 351 passwords (3.5%), and BestCover with $P^2_{A-40}$ breaks 70 passwords (0.7%).

Figure 5.14(b) shows the fractions of the accounts for which the indicated number of passwords were cracked. Each bar represents one condition. Unbiased with $P^2_{A-40}$ crack at least one password for 61.0% accounts, while Unbiased with $P^2_{L-15}$ could crack 65.0%. Even though BestCover with $P^2_{L-15}$ crack more passwords in total than with BestCover with $P^2_{L-15}$, BestCover with $P^2_{L-15}$ breaks more accounts for at least once. Both Unbiased and BestCover with $P^2_{L-15}$ cracks all 15 passwords for 4 (5.7%) out of 697 accounts.

5.7 Discussion

In this section, we discuss potential usage of our attack framework, other attack approaches and the limitations of our work.

5.7.1 Picture-Password-Strength Meter

Our framework could enhance the security of PGA so it would eventually protect users and their devices by providing a picture-password-strength meter. One way to help users choose secure passwords is to enforce some composition policies, such as ‘three taps are not allowed’. However, a recent effort Kelley et al. (2012) on text-based password found that rule-based password compositions are ineffective because they can allow weak passwords and reject strong ones. The cornerstone of accurate
strength measurement is to quantify the strength of a password. With a ranked password dictionary, our framework, as the first potential picture-password-strength meter, is capable of quantifying the strength of selected picture passwords. More intuitively, a user could be informed of the potential number of guesses for breaking a selected password through executing our attack framework.

5.7.2 Other Attacks on PGA

Besides keyloggers that record users’ finger movements, there are some other attack methods that may affect the security of PGA and other background draw-a-secret schemes. Shoulder surfing, an attack where attackers simply observe the user’s finger movements, is one of them. In our survey, 54.3% participants believe the picture password scheme is easier for attackers to observe when they are providing their credentials than text-based password. Several new shoulder surfing resistant schemes Forget et al. (2010); Zakaria et al. (2011) were proposed recently. However, the usability is always a major concern for these approaches. The smudge attack Aviv et al. (2010) which recovers passwords from the oily residues on a touch-screen has also been proven feasible to the background draw-a-secret schemes and could pose threats to PGA.

5.7.3 Limitations of Our Study

While we took great efforts to maintain our studies’ validity, some design aspects of our studies and developed system may have caused subjects to behave differently from what they do on Windows 8™ PGA. Subjects in Dataset-2 pretended to access their bank information but did not have anything at risk. Schechter et al. Schechter et al. (2007) suggest that role playing like this affects subjects’ security behavior, so passwords in Dataset-2 may not be representative of real passwords chosen by real
users. Besides, we did not record whether a subject used a tablet with touch-screen or a desktop with mouse. The different ways of input may affect the composition of passwords. Moreover, Dataset-2 includes multiple passwords per user and this may have impacted the results. In our analyses, training password datasets include passwords from the targeted subject. Even though this may have affected the results, we believe it is less influential. Because, for each analysis, there were around 9,400 training passwords for which only 14 came from the targeted user. Since all training passwords were treated equally, the influence brought by the 0.14% training data is low. As discussed in Section 5.6, even though our online attack results showed the feasibility of our approach, it still requires more realistic and significant attack cases. As part of future work, we plan to integrate smudge attacks Aviv et al. (2010) into our framework to improve the efficacy of our online attacks.

5.8 Related Work

The security and vulnerability of text-based password have attracted considerable attention because of several infamous password leakage incidents in recent years. Zhang et al. Zhang et al. (2010) studied the password choices over time and proposed an approach to attack new passwords from old ones. Castelluccia et al. Castelluccia et al. (2012) proposed an adaptive Markov-based password strength meter by estimating the probability of password using training data. Kelley et al. Kelley et al. (2012) developed a distributed method to calculate how effectively password-guessing algorithms could guess passwords. Even though the attack framework we presented is dedicated to cracking background draw-a-secret passwords, the idea of abstracting users’ selection processes of password construction introduced in this chapter could also be applicable to cracking and measuring text-based passwords.

The basic idea of attacking graphical password schemes is to generate dictionaries
that consist of potential passwords Thorpe and Van Oorschot (2004a). However, the lack of sophisticated mechanisms for dictionary construction affects the attack capabilities of existing approaches. Thorpe et al. Thorpe and van Oorschot (2007) proposed a method to harvest the locations of training subjects’ clicks on pictures in click-based passwords to attack other users’ passwords on the same pictures. In the same paper Thorpe and van Oorschot (2007), they presented another approach which creates dictionaries by predicting hot-spots using image processing methods. Oorschot et al. Oorschot and Thorpe (2008) cracked DAS using some password complexity factors, such as reflective symmetry and stroke-count. Salehi-Abari et al. Salehi-Abari et al. (2008) proposed an automated attack on the PassPoints scheme by ranking passwords with click-order patterns. However, the click-order patterns introduced in their approach could not capture users’ selection processes accurately, especially when a background image significantly affects user choice. Nontargeted attacks on PGA passwords were discussed in our previous work Zhao et al. (2014).

5.9 Summary

We have presented a novel attack framework against background draw-a-secret schemes with special attention on picture gesture authentication. We have described an empirical analysis of Windows 8 picture gesture authentication based on our user studies. Using the proposed attack framework, we have demonstrated that our approach was able to crack a considerable portion of picture passwords in various situations. We believe the findings and attack results discussed in this chapter could advance the understanding of background draw-a-secret and its potential attacks.
Chapter 6

DISCOVERING UNDERGROUND INTELLIGENCE

6.1 Introduction

Today’s malware-infected computers are deliberately grouped as large scale destructive botnets to steal sensitive information and attack critical net-centric production systems David Anselmi (2010). A recent report by FBI IC3 shows that company losses from the cybercrime rose from 264.6 million dollars to 559.7 million dollars IC3 (2009). The situation keeps getting worse when botnets make use of legitimate social media, such as Facebook and Twitter, to launch botnet attacks Athanasopoulos et al. (2008); Thomas (2010). We even recently observed that the well-known social networks, such as Facebook and Twitter, could be used as platforms to launch botnet attacks Thomas (2010). In order to cope with these emerging threats, both proactive and passive approaches are proposed to gather and analyze information from malicious code samples. Previous research efforts on countering botnet attacks could be classified into four categories: (i) capturing malware samples Bächer et al. (2005), (ii) collecting and correlating network and host behaviors of malwares Gu et al. (2007), (iii) understanding the logic of malwares Chiang (2007); Porras et al. (2009), and (iv) infiltrating and taking over botnets Stone-Gross et al. (2009); Kanich et al. (2008).

Notably, most studies in the area of countering malware and botnets have been focused on detecting bot deployment, capturing and controlling bot behaviors. However, there is little research on examining how these malicious programs are created, rented and sold by adversaries. Even though preventive solutions against thousands of known bots have been deployed on networked systems, and some botnets were
even taken down by law enforcement agencies Mushtaq (2009), the majority of adversaries are still at large and keep threatening the Internet by developing more bots and launching more net-centric attacks. The major reason for this phenomenon is that previous malware-related activities—such as developing, renting and selling bots—were occurred mostly offline, which were way beyond the scope of security analysts.

In recent years, the pursuit of more profit in underground communities leads to the requirement for global collaboration among adversaries, which tremendously changed the division of labor and means of communication among them Dunham and Melnick (2008). (Un)fortunately, adversaries started to communicate with each other, distribute and improve attack tools with the help of the Internet, which leaves security analysts new clues for evidence acquisition and investigation on unwanted program development and trade. Before the widespread use of online social networks (OSNs), adversaries would communicate via electronic bulletin board systems (BBS), forums, and Email systems Goodin (2008).

Content-rich Web 2.0, ubiquitous computing equipments, and newly emerging online social networks provide an even bigger arena for adversaries. In particular, the value of OSNs for adversaries is the capability to cooperate with destructive botnets. The role of OSNs in botnet attacks is twofold: first, OSNs are the platforms to form online black markets, release bots, and coordinate attacks Dunham and Melnick (2008); Bächer et al. (2005); Holt and Bossler. (2010); second, OSN profiles act as bots to perform malicious actions Athanasopoulos et al. (2008) or C&C server nodes coordinates other networked bots Thomas (2010). Although our efforts in this chapter are mainly concerned about the former case, our proposed model for online underground social dynamics and corresponding social metrics can be also utilized to identify compromised and suspicious OSN profiles.

In the former role of OSNs for botnet attacks, a recent study Dunham and Mel-
nick (2008) reveals that many of these online black markets operate out of eastern Europe and Russia, and allow skilled programmers to create and profit from the sale of new malicious codes. These markets even use eBay-style feedback systems to help the suppliers of these services establish good business reputations. Individuals who control existing botnets also sell access to their infected machines for a variety of attacks including spamming and DDoS. As a consequence, these markets enable a great deal of unskilled adversaries and innocent computer users to engage in cybercrime.

Given the great amount of valuable information in online social dynamics, the investigation of the relationships between online underground social communities and network attack events are imperative to tactically cope with net-centric threats. In this chapter, we propose a novel solution using social dynamics analysis to counter malware and botnet attacks as a complement to existing research investments. In order to demonstrate the feasibility and applicability of our mechanism, we describe a proof-of-concept prototype with a case study on real-world data archived from Internet. We formally model online underground social dynamics and propose SocialImpact which is a suite of measures to help identify adversarial evidence by examining social dynamics.

The rest of this chapter is organized as follows. Section 6.3 presents our online underground social dynamics model and addresses SocialImpact, which is a systematic ranking analysis suite for mining adversarial evidence based on the model. In Section 6.4, we discuss the design and implementation of our proof-of-concept system. Section 6.5 presents the results of applying our model and metrics on synthetic social dynamics data. Section 6.6 presents the evaluation of our approach on real-world underground social dynamics data followed by the related work in Section 6.7. Section 6.8 concludes this chapter.
6.2 Patterns in Cybercrime

Figure 6.1 shows a typical workflow of cybercrime. In Step 1, malware programmers develop crafted attack tools. The most prevalent and destructive tool developed to carry out various attacks is a set of bots. Malware programmers turn bots to bot-herders through online black markets or offline channels. In Step 2, bot-herders deploy a botnet through social engineering, drive-by-download or other possible vectors. In Step 3, bot-herders rent a botnet out to other adversaries, from which bot-herders and malware programmers profit, who have targets in mind but do not have the technological expertise to design or administer the botnet. In Step 4, attackers, such as spammers, take control of the botnet. A rented botnet may result in a variety of attacks launched by multiple adversaries who might have different intents. In Step 5, attackers coordinate bot nodes to perform multiple attacks such as spamming, identity theft, DDoS, phishing attacks, etc.

The power of botnets relies on their coordination and the volume of the responses from the bot nodes. In a typical botnet, hundreds to thousands of bot nodes respond to botmaster’s commands. When these nodes are instructed to connect to one webpage at the same time, the aggregated volume of the network traffic would be tremendous for most companies to handle, causing denial-of-service to the targeted
servers. When these nodes are instructed to download banking credentials, the botmaster receives credentials from each bot which can be thousands or even millions in some botnets. Another critical problem caused by botnets is e-mail spamming. Nowadays a spam causes not only a network-clogging problem, but also a means for adversaries to distribute additional malwares.

Most significant botnets today are constantly changing. They are evolved by adding bots, deleting bots, changing to new channels, being upgraded, etc. Attempting to discover their C&C servers may bring immediate benefits but also stimulate the evolution of botnets. Park and Reeves pointed out Park and Reeves (2009), it is also important to monitor botnets for an extended time to learn the purpose of the botnets to develop more effective countermeasures. A good example of this is what happened after the takedown of the largest spamming botnet in the world, the McColo botnet. In November 2008, the major spamming botnet command and control, McColo was shut down. The next day the spam volume was nearly cut in half, but by the end of 2009 the volume was increased higher than ever. Experts agree that the explosion in spamming is a result of the botmasters in charge of McColo regrouping and creating other botnets in which they could spread their spam once again Security (2010).

In other words, existing approaches concentrate on Step 2, Step 4 and Step 5 in Figure 6.1 and largely overlook Step 1 and 3. Furthermore, since malware authors have adopted protection approaches to hide malware-related data from analysis, research results on Step 2, Step 4 and Step 5 which may work smoothly for prior malwares are ineffective to accommodate the new trend of malware evolution. Therefore, we also propose to systematically examine social dynamics to bridge the gap of research efforts on Step 1 and Step 3 and automatically extract internal ciphertext data from malware to complement existing efforts on Step 2, Step 4 and Step 5.
A recent study reveals that rogue programs are sold in black markets built on top of online platforms Holt and Bossler. (2010). Individuals who control existing botnets also sell access to their infected machines for a variety of attacks including spamming and denial of service. As a consequence, these markets enable a great deal of unskilled adversaries and innocent computer users to engage in cybercrime. Before the advent of OSNs, adversaries would communicate via electronic bulletin board systems (BBS), forums and Email systems. More seriously, newly-emerging OSNs provide an even big arena for high-tech criminals.

In Holt (2008), researchers describe a good motivating example to show the value of examining social dynamics for analysis of malware distribution and evolution. Try2DDos is a standalone attack tool for distributed denial of service attack (DDos) which was named one of top 30 DDoS tools in 2008 Vaqxine (2008). As shown in

Figure 6.2: Motivating Example of Malware Distribution and Evolution in Social Dynamics (All screenshots taken from Holt (2008))
Figure 6.2a, *Try2DDos* was first released on a French forum *Underground konnekt* by *libere_ton_espri* in June, 2005. On the same day, it appeared in a Chinese hacker site without any modification as shown in Figure 6.2b. More than one year later, the first public variant of this tool in Spanish appeared on an Argentina hacker forum as shown in Figure 6.2c. Almost another year later, a Chinese variant of this tool developed by *ZzAge* was announced as shown in Figure 6.2d. From 2005 to 2008, this tool and its variants spread to China, Russia, Guatemala, and Argentina and caused damages to a large number of networked systems Holt (2008).

6.3 Using Social Interaction Patterns to Discover Intelligence

In this section, we first address the modeling approach we utilized to represent online underground social dynamics (OUSDs). Unlike existing OSN models Zhelueva and Getoor (2009); Hu *et al.* (2014c); Li *et al.* (2012); Zhao *et al.* (2012a); Hu *et al.* (2012) which emphasize on user profile, friendship link, and user group, our model also gives attention to user-generated contents due to the fact that a wealth of information resides in online adversarial conversations. We also elaborate the design principles of social metrics to identify adversarial behaviors in OUSDs. Then, we present SOCIALIMPACT, which consists of nine indices, to bring order to underground social dynamics based on our OUSD model.

Adversaries choose online social networks which meet their special requirements to form online underground social communities. Online Underground Social Networks (OUSNs) are used to share technical articles and trade malicious tools, rather than photo-sharing or video-sharing, making them different from normal OSNs in the following aspects:

- OUSNs provide a blog-like article-sharing mechanism, which has less constraints on the length of articles a user can post. Length limitation of posts adopted
in traditional OSNs, such as Twitter and Facebook, is unlikely suitable for well-explained technical articles in OUSNs.

- OUSNs have less access and write constraints on posted articles. For instance, Facebook adopts strict policies to protect its users' privacy, in which one user has to be in the others' circle of trust to access and comment on their posts. However, in OUSNs, a user does not need to be a friend of the article author to read the article or give comments on it. This characteristic allows OUSNs to disseminate more knowledge and technical discussions than OSNs.

- OUSNs do not require users to provide their real-world identities. Adversaries prefer not to associate their real-world identities with their online profiles, therefore OUSNs do not claim themselves as real social networks. However, OSNs such as Facebook require users to provide their real names, education backgrounds, and relationship statuses.

6.3.1 Online Underground Social Dynamics Model

As shown in Figure 6.3, an OUSD can be represented by six fundamental entities and five basic types of unidirectional relationships between them.

*Users* are those who have profiles in the network and have the rights to join
groups, post articles, and give comments to others. Groups are those to which users can belong. In an OUSD, groups are mainly formed based on common interests. Articles are posted by users who want to share them with the society. In an OUSD, articles might introduce the latest technologies, analyze recent vulnerabilities, call for participation of network attacks, and trade newly developed and deployed botnets. In terms of the form of articles, they do not have to be literary. They could also contain multimedia contents, such as photos and melodies. Comments are the subsequent posts to articles. Posts are the union of articles and comments. Strings are the elementary components of articles and comments. Strings are not necessarily meaningful words. They could be names, URLs, and underground slangs. A user has a relationship authorOf with each post s/he creates. A user has a relationship followerOf with each user s/he follows. A user has a relationship memberOf with each group s/he joins. An article has a relationship hostOf with each comment it receives. A post has a relationship containerOf with each string it consists of.

The following formal description summarizes the above-mentioned entities and relationships.

**Definition 6.1.** Online Underground Social Dynamics. A OUSD is modeled with the following components:

- $U$ is a set of users;
- $G$ is a set of user groups;
- $A$ is a set of articles;
- $C$ is a set of comments;
- $P$ is a set of posts. $P = A \cup C$;
- $S$ is a set of strings;

- $UP = \{(u, p) | u \in U, p \in P \text{ and } u \text{ has an authorOf relationship with } p\}$ is a one-to-many user-to-post relation denoting a user and her posts;

- $FL = \{(u, y) | u \in U, y \in U \text{ and } u \text{ has a followerOf relationship with } y\}$ is a many-to-many user-to-user follow relation;

- $MB = \{(u, g) | u \in U, g \in G \text{ and } u \text{ has a memberOf relationship with } g\}$ is a many-to-many user-to-group membership relation;

- $AC = \{(a, c) | a \in A, c \in C \text{ and } a \text{ has a hostOf relationship with } c\}$ is a one-to-many article-to-comment relation denoting an article and its following comments; and

- $PS = \{(p, s) | p \in P, s \in S \text{ and } p \text{ has a containerOf relationship with } s\}$ is a many-to-many post-to-string relation.

We focus on the main structure and activities in online underground society and overlook some sophisticated features & functionalities, such as online chatting, provided by specific OSNs and BBS. Hence, our OUSD model is generic and can be a reference model for most real-world OSNs and BBS. As a result, security analysts could easily map real-world social dynamics data archived from any OSNs and BBS to our model for further analysis and investigation.

### 6.3.2 Principles of Metric Design and Definitions

We also address the following critical issues related to evidence mining in underground society: How can we identify adversaries among the crowd of social users? Given the additional evidence acquired from other sources, how can we correlate
them with underground social dynamics? How can we measure the evolution in underground community? To answer these questions, we articulate several principles that the measures for underground social dynamics analysis should follow: 1) The measures should support identifications of interesting adversaries and groups based on both their social relationships and online conversations; 2) The measures should be able to take external evidence into account and support interactions with security analysts; and 3) The measures should support temporal analysis for the better understanding of the evolution in adversarial society.

To this end, we introduce several feature vectors to achieve aforementioned goals. For the mathematical notations, we use lower case bold roman letters such as $\mathbf{x}$ to denote vectors, and uppercase bold roman letters such as $\mathbf{V}$ to denote matrices. We assume all vectors to be column vectors and a superscript $T$ to denote the transposition of a matrix or vector. We also define $max()$ as a function to return the maximum value of a set.

**Definition 6.2. Article Influence Vector.** Given an article $a \in A$, the article influence vector of $a$ is defined as $\mathbf{v}_a^T = (v_1, v_2, v_3)$, where $v_1$ is the length of the article, $v_2 = |\{c \mid c \in C \text{ and } (a, c) \in AC\}|$ is the number of comments received by $a$, and $v_3$ is the number of outlinks it has.

When stacking all articles’ influence vector together, we get the article influence matrix $\mathbf{V}$. We assess an article’s influence by its activity generation, novelty and eloquence Agarwal et al. (2008). More comments an article receives, more influential it is. The number of outlinks and length is used to approximately represent article’s novelty and eloquence without extracting its semantics. More outlinks an article has, less novelty it has. Also, a longer article is assumed to be more eloquent.

**Definition 6.3. Article Relevance Factor.** Given a set of strings $\mathbf{s} = \{s_1, s_2, ..., s_n\} \subseteq$
$S$ and an article $a \in A$, article relevance factor, denoted as $r(a, s)$, is defined as the number of occurrence of strings $s$ in the article $a$.

The strings $s$ could represent an external evidence that security analysts acquired from other sources and query keywords in which security analysts are interested. The given strings $s$ could represent an external evidence that security analysts acquired from other sources and query keywords in which security analysts are interested.

**Definition 6.4. User Activeness Vector.** The user activeness vector of $u$ is defined as $z_u^T = (z_1, z_2, z_3)$, where $z_1 = \{|p | p \in P \text{ and } (u, p) \in UP\}|$ is the number of articles and comments $u$ posted, $z_2 = \{|y | y \in U \text{ and } (u, y) \in FL\}|$ is the number of users $u$ follows, and $z_3 = \{|g | g \in G \text{ and } (u, g) \in MB\}|$ is the number of groups $u$ joins.

We measure a user’s activeness by the number of posts s/he sends, users s/he follows, and groups s/he joins. By aggregating all users’ $z_u$, we get **user activeness matrix $Z$**.

**Definition 6.5. Social Matrix.** Social matrix, denoted as $Q$, is defined as a $|U| \times |U|$ square matrix with rows and columns corresponding to users. Let $v$ be a user and $N_v$ be the number of users $v$ follows. $Q_{u,v} = 1/N_v$, if $(v, u) \in FL$ and $Q_{u,v} = 0$, otherwise.

Social matrix is similar to transition matrix for hyperlinked webpages in PageRank. The sum of each column in social matrix is either 1 or 0, which depends on whether the $v$th column user follows any other user.

**Definition 6.6. $\delta$-n Selection Vector.** A $\delta$-n selection vector, denoted as $y_\delta^n$, is defined as a boolean vector with $n$ components and $\|y_\delta^n\|_1 = \delta$. 

143
A $\delta$-$n$ selection vector is used to select a portion of elements for one set. For example, the top 10 influential articles of a user $a$ could be represented by a selection vector $y_{10}^{\mid A \mid}$ over the article set $A$. By stacking all users’ $\delta$-$n$ selection vectors over the same set together, we get the $\delta$-$n$ selection matrix $Y_\delta^n$.

6.3.3 Ranking Metrics

As shown in Figure 6.4, SOCIAIIMPACT consists of nine indices, which are classified into three categories: string & post indices, user indices, and group indices. Each index in upper categories is computed by the indices from lower categories. For instance, UserInfluence that indicates a user’s influence in the community is calculated based on the ArticleInfluence of user’s articles and the social metrics $Q$ of this community.

To fulfill Principle 1, user and group indices are devised to identify influential, active, and relevant users and groups. We devise personalized PageRank models Chakrabarti (2007) to calculate UserInfluence and UserRelevance, since it could capture the characteristics of both user-to-user relationships and user-generated contents in social dynamics. To accommodate Principle 2, ArticleRelevance, UserRelevance and GroupRelevance are designed to take external strings as inputs, combine them with existing data in social dynamics, and generate more comprehensive results. To fulfill Principle 3, all feature vectors and indices could be calculated for a given time win-
dow and \textit{StringPrevalence} could indicate the topic evolution in the society. Moreover, we believe the combination of \textit{UserActiveness} and \textit{UserInfluence} could also be used to identify suspicious spam profiles in online social networks.

We consider a weighted additive model Keeney and Raiffa (1993) when there exist several independent factors to determine one index. To reduce the bias introduced by different size of sets, we use $\delta$-n selection vector to choose a portion of data in calculation. The followings are the detailed descriptions of indices.

\textbf{ArticleInfluence}, denoted as $x_1(a)$, represents the influence of article $a$. $x_1(a)$ is computed as $v_0^T w_1$, where $w_1$ denotes the weight vector.

By normalizing $x_1(a)$ to $[0, 1]$ and stacking $x_1(a)$ from all articles together, we get a vector $x_1$.

$$x_1 = \frac{V_0^T w_1}{\max_{b \in A} (x_1(b))} \quad (6.1)$$

\textbf{ArticleRelevance}, denoted as $x_2(a, s)$, represents the relevance of the article $a$ to given strings $s$. $x_2(a, s)$ is proportional to the occurrence of the given strings in the article and the influence of the article.

$$x_2(a, s) = \frac{r(a, s)x_1(a)}{\max_{b \in A} (r(b, s)x_1(b))} \quad (6.2)$$

By stacking $x_2(a, s)$ from all users together, we get a vector $x_2(s)$ denoting all articles’ relevance to $s$.

\textbf{UserInfluence}, denoted as $x_3$, represents the influence of a user. $x_3$ can be measured by two parts. One is the impact of the user’s opinions, which is modeled by \textbf{ArticleInfluence}. The other is the user’s social relationships, which is modeled by $Q$. $x_3$ is devised as a personalized PageRank function to capture both parts.

By stacking $x_3$ from all users together, we get a vector $x_3$.

$$x_3 = d_3 Q x_3 + (1 - d_3) Y_{|A|} x_1 \quad (6.3)$$
Where \( d_3 \in (0, 1) \) is the decay factor which makes the linear system stable and convergent. \( Y_{\alpha}^{[A]} \) is the \( \delta - n \) selection matrix corresponding to all users’s top \( \alpha \) influential articles.

**UserRelevance**, denoted as \( x_4(s) \), represents the relevance of a user to strings \( s \).

By stacking \( x_4(s) \) from all users together, we get a vector \( x_4 \).

\[
x_4(s) = d_4 Q x_4(s) + (1 - d_4)(Y_{\alpha}^{[A]} x_2(s))
\]

(6.4)

Where \( d_4 \in (0, 1) \) is the decay factor. \( Y_{\alpha}^{[A]} \) is a \( \delta - n \) selection matrix corresponding to all users’s top \( \alpha \) relevant articles to \( s \).

**UserActiveness**, denoted as \( x_5 \), represents the activeness of a user.

\[
x_5 = Z^T w_5
\]

(6.5)

We use the addition of a group’s top \( \alpha \) members’ influence, relevance, and activeness to model its influence, relevance, and activeness, respectively. As mentioned before, this model can reduce the bias caused by the number of members.

**GroupInfluence**, denoted as \( x_6 \), represents the influence of a group.

By stacking all \( x_6 \) together, we get \( x_6 \).

\[
x_6 = Y_{\alpha}^{[U]} x_3
\]

(6.6)

Where \( Y_{\alpha}^{[U]} \) is the \( \delta - n \) selection matrix corresponding to all groups’ top \( \alpha \) influential users.

**GroupRelevance**, denoted as \( x_7 \), represents the relevance of a group to strings \( s \).

By stacking all \( x_7 \) together, we get \( x_7 \).

\[
x_7 = Y_{\alpha}^{[U]} x_4
\]

(6.7)

Where \( Y_{\alpha}^{[U]} \) is the \( \delta - n \) selection matrix corresponding to all groups’ top \( \alpha \) relevant users.
GroupActiveness, denoted as $x_8$, represents the activeness of a group. By stacking all $x_8$ together, we get $x_8$.

$$x_8 = Y_{\alpha}^{[U]} x_5$$

(6.8)

Where $Y_{\alpha}^{[U]}$ is the $\delta$-$n$ selection matrix corresponding to all groups’ top $\alpha$ active users.

StringPrevalence, denoted as $x_9(s)$, represents the popularity of string $s$.

$$x_9(s) = \sum_{p_j \in P} t_{s,p_j}$$

(6.9)

where $t_{s,p_j}$ is the term frequency-inverse document frequency Salton and Buckley (1988) of string $s$ in post $p_j$.

The computations for UserInfluence and UserRelevance are proven to be convergent Bianchini et al. (2005). And the corresponding time complexity is $O(|H| \log(1/\epsilon))$, where $|H|$ is the number of followerOf relationships in the social dynamics and $\epsilon$ is a given degree of precision Bianchini et al. (2005). The time complexity for calculating StringPrevalence is $O(|P||S|)$, where $|P|$ is the number of posts and $|S|$ is the size of string set. The complexities for all other indices are linear if the underlying indices are calculated.

6.4 System Design and Implementation

In this section, we describe the challenges in analyzing real-world underground social dynamics data. We address our efforts to cope with these challenges and present the design and implementation of our proof-of-concept system.

6.4.1 Challenges from Real-world Data

The first challenge of real-world data is its multilingual contents. The most effective way to coping with this challenge is to take advantage of machine translation.
systems. Our tool utilizes Google Translate to detect the language of the contents and translate them into English. However, machine translation systems may fail to generate meaningful English interpretations for the following cases: i) adversaries may use cryptolanguages that no machine translation system could understand. For instance, *Fenya*, a Russian cant language that is usually used in prisons, is identified in online underground society Yarochki (2009); and ii) both intentional and accidental misspellings are common in online underground society Raymond (1996). In order to cope with this challenge, our tool maintains a dictionary of known jargons, such as *c4n* as *can* and *sUm1* as *someone*.

Another challenge is that the social dynamics data may not be in a consistent format. Different OSNs use different styles in web page design. Even in one OSN, in order to make the web page more personalized, the OSN allows users to customize the format of their posts. Since HTML is not designed to be machine-understandable in the first place, extracting structural information from HTML is a tedious and heavy-labor work. To address this problem, we first cluster data, and then devise an HTML parser for each cluster. We also design a light-weight semi-structure language to store the information extracted from HTML.

Since one major component in social dynamics is the relationships between entities, storing and manipulating social dynamics data in a relational database is relatively time-consuming. We choose graph database Angles and Gutierrez (2008) which employs the concepts from graph theory, such as node, property, and edge, to realize faster operations for associative data sets.

### 6.4.2 System Architecture and Implementation

Figure 6.5 shows a high level architecture of our tool. The upper level of our tool includes several visualization modules and provides query control for security
analysts to provide the additional evidence. In reality, such evidence could be in the format of text, picture, video, audio or any other forms. Yet, representing multimedia contents like pictures and videos in a machine-understandable way is still a difficult challenge. Our tool acts like a modern web search engine in response to keyword queries. Social graph viewer is designed to show social relationships among users and groups. Ranking analysis viewer is used to list the ranking results based on security analysts’ queries. Content viewer can show both original and translated English web resources.

The lower level of the architecture realizes underlying functionalities addressed in our framework. After underground community data is crawled from Internet, the HTML parser module extracts meaningful information from it. If the content is not in English, our translator takes over and generates English translation. All extracted information is stored in a graph database for the rapid retrieval. Analysis modules have two working modes: offline and online. The offline mode generates demographical information with demographical analysis engine (DAE) and intelligence, such as user influence and activeness, with SOCIALIMPACT engine (SIE). When security
analysts provide the additional evidence, SOCIALIMPACT engine switches to online mode and generates analysis results, such as user relevance, based on data in graph database and additional evidence provided by security analysts.

Our tool was implemented in Java programming language. We took advantage of Java swing and JUNG to realize graphical user interfaces and graph visualization. As we mentioned before, our tool uses Google Translate API to translate texts. In most cases, Google Translate could output acceptable translations from original texts.

Our tool stores user profiles, user-generated contents, and social relationships among users in a Neo4j graph database. For each group, user, article, and comment, our tool creates a node in the database, stores associated data—such as the birthday of user and the content of article—in each node’s properties, and assigns the relationships among nodes.

6.4.3 Visualization Interfaces of Our Tool

Figure 6.6 depicts interfaces of our tool. As illustrated in Figure 6.6a, all users in the social group are displayed by a circle. And their followerOf relationships are displayed with curved arrows. It is clear to view that some users have lots of followers while others do not. By clicking any user in the group, our tool has the ability to
highlight this user in red and all his followers in green. In this way, our tool helps analysts understand the social impact of any specific user. Another windows as shown in Figure 6.6b displays the ranking results. Analysts can specify the ranking metric, such as UserInfluence and UserActiveness, to reorder the displayed rank. Clicking a user’s name which is the second column in Figure 6.6b would bring the analysts to the list of all posted articles by the user in descending order of ArticleInfluence. Clicking the user’s profile link which is the third column in Figure 6.6b would bring the analysts to the webpage of the user’s profile archived from the Internet. Analysts could also specify some keywords in query control and our tool would display the results in descending order of ArticleRelevance. As shown in Figure 6.6c, our tool displays both the original and translated texts and highlights the input keywords in red.

6.5 Experiments on Synthetic Social Dynamics Data

We first compared our social ranking approach with a PageRank-based solution Page et al. (1999a) to evaluate the effectiveness of our social ranking mechanism. PageRank uses numerical weights of elements that are linked together to measure their relative importance. Although PageRank is successfully deployed in commercial search engines, as discussed in Agarwal et al. (2008), it is not very suitable to rank sparsely linked elements.

For simplicity, we only included a synthetic case study considering user influence and activeness, as shown in Figure 6.7. In this scenario with five users, the left-hand figure shows the relationships between users and the right-hand tables show information from user-generated contents in five different stages. David and Carl are two of most popular users with four and three followers, respectively. While Edward seems the most eloquent user in the stage one with six articles and twenty posted...
Figure 6.7: An OUSD Scenario

Figure 6.8(a) shows the user influence index sorted in descending order in the stage one. Alice is ranked as the most influential user, mainly because her four articles received nineteen comments from the society. Note that, although David has four followers in the society, his influence is limited due to the fact that he initiated few attractive conversations. Although, Edward has no followers, our model considers he still has some influence in the community because his contributions have attracted other’s attention. Figure 6.8(b) shows the user activeness index in the stage one. The least influential user Edward is ranked as the most active user not only because he is eloquent, but also because he follows everyone in the community. Bob is ranked as the second most active member since he contributed to many conversations. We notice that the most influential user Alice is ranked as the second least active user which verifies that a user does not need to speak much to make a difference. Figure 6.8(c) shows the PageRank-based ranking analysis results. Note that PageRank ignores user-generated contents, and only considers user relationships for ranking analysis. One reason for Carl being more influential than David, even if in the case that David has more followers than Carl, is because in PageRank analysis the value of link-votes is divided among all outbounds. The fact that David only follows Carl indicates Carl’s influence in this model. PageRank fails to identify Edward’s influence as well merely because he has no follower. Another drawback of PageRank analysis is that it
<table>
<thead>
<tr>
<th>Rank</th>
<th>Name</th>
<th>Index</th>
<th>Rank</th>
<th>Name</th>
<th>Index</th>
<th>Rank</th>
<th>Name</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>69.3</td>
<td>1</td>
<td>Edward</td>
<td>29</td>
<td>1</td>
<td>Carl</td>
<td>1.111</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>23.9</td>
<td>2</td>
<td>Bob</td>
<td>17</td>
<td>2</td>
<td>David</td>
<td>0.371</td>
</tr>
<tr>
<td>3</td>
<td>Carl</td>
<td>21.5</td>
<td>3</td>
<td>David</td>
<td>14</td>
<td>3</td>
<td>Alice</td>
<td>0.075</td>
</tr>
<tr>
<td>4</td>
<td>David</td>
<td>13.6</td>
<td>4</td>
<td>Alice</td>
<td>12</td>
<td>4</td>
<td>Bob</td>
<td>0.024</td>
</tr>
<tr>
<td>5</td>
<td>Edward</td>
<td>5.1</td>
<td>5</td>
<td>Carl</td>
<td>10</td>
<td>5</td>
<td>Edward</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Figure 6.8: Indices According to Our Solution and PageRank

cannot generate temporal patterns of the influential users since relationships do not change.

We normalize three different indices calculated for each adversary and show their comparison in Figure 6.9. The significant difference between our solution and PageRank-based solution shows the importance of considering user-generated contents as well as social relationships.

6.6 A Case Study on Real-world Online Underground Social Dynamics

In this section, we present our evaluation on real-world social dynamics. We evaluated our tool on 4GB of data crawled from Livejournal.com which is a popular
online social network especially in the Russian-speaking countries. We anonymized the group names and user names in this OSN for preserving privacy.

All webpages in this OSN could be roughly divided into two categories in terms of content: i) profile and ii) article. The profile webpage contains basic information of a user or a group, which includes name, biography, location, birthday, friends, and members. Every article has title, author, posted time, content, and several comments by other users. The webpages are mainly .html files, along with some .jpeg, .gif, .css, and .js files. Our solution only considers text data from .html files.

We started to crawl group profiles from six famous underground groups in this OSN. Then we crawled all members’ profiles and articles of these six groups. We also collected one-hop friends’ articles of these members. Therefore, we ended up with 29,614 articles posted by 6,364 users which are from 4,220 groups. Based on the information in user profiles, we noticed that about 32.7% and 52.7% users were born in early and mid-late 80’s. This clearly illustrates the age distribution of active users in this community.

6.6.1 Post, User and Group Analysis

Our tool calculated all articles’ ArticleInfluence and identified top 50 articles over a time window of 48 months. Since not all of these articles are related to computer security, we checked these articles in descending order of their influences and picked five articles that are highly related to malware. We could observe some popular words related to malware, such as PE (the target and vehicle for Windows software attacks), exploits (a piece of code to trigger system vulnerabilities), hook (a technique to hijack legitimate control flow) and so on.

Our tool also generated each user’s UserInfluence and UserActiveness and group’s GroupInfluence and GroupActiveness over a time window of 48 months. And, Table 6.2
shows the top five influential/active users/groups for the entire period of our observation. We can notice that there is no overlap between the top five influential users and the top five active users, while there exists similarity for the top five influential groups and the top five active groups.

We calculated the correlation coefficient ($corrcoef$) for the pairs of UserInfluence and UserActivenss, GroupInfluence and GroupActivenss based on the results generated from our tool. Similar to the phenomenon we identified in Table 6.2, in Figure 6.10(a) we observed that the correlation coefficient between UserInfluence and UserActivenss is around 0.52 (the maximum value for correlation coefficient is 1 indicating a perfect positive correlation between two variables), which means one user’s influence is not
Figure 6.10: Correlation Coefficient of UserActiveness & UserInfluence and GroupActiveness & GroupInfluence

highly correlated to her/his activeness. This phenomenon indicates that talking more does not make a user more influential in a community. On the other hand, as shown in Figure 6.10(b) we observed that the correlation coefficient between GroupInfluence and GroupActiveness is around 0.90, which indicates a very strong positive correlation between the influence and the activeness of a group. The application of influence and activeness indices is not limited to identify such a social phenomenon. We could also leverage the high UserActiveness and the low UserInfluence as indicators for the analysis of social spammers in any OSN.

The temporal patterns of the influential/active users/groups could be observed in Figure 6.11, where x-axis denotes the users/groups who were identified as the most influential/active ones for each month. For example, $x = 1$ denotes the most influential/active user/group of the first month in our time window and $x = 48$ denotes the most influential/active user/group of the last month in our time window; $y$-axis denotes the entire 48 months in the time window; and $z$-axis denotes user/group’s influence/activeness value. As shown in Figure 6.11(a), some users maintain their influence status for several months. The large plain area in the right part of this figure indicates most users come as the most influential ones suddenly. This observation implies that a user does not need to be a veteran to be an influential one in the
community. On the other side, we can see from Figure 6.11(b) that most active users remain active before they became the most active ones. The plain area in the left portion of Figure 6.11(b) implies that most users do not always keep active. Normally they keep active for 15-30 months, then get relatively silent. While the smaller plain area in the left part of Figure 6.11(a) shows once a user becomes influential, s/he keeps the status for a long period of time. Figure 6.11(c) shows that there are 2 or 3 groups who maintain the status of influence during the whole 48 months and get even more influential as time goes on. While, other groups only keep influential for a relatively short period of time and just fade out. Figure 6.11(d) shows the similar phenomenon.
6.6.2 Evidence Mining by Correlating Social Dynamics with Adversarial Events

We present our finding with keyword queries on the same dataset in our tool. Those keywords and attack patterns could also be automatically generated from the approaches proposed in Zhao et al. (2011a); Zhao and Ahn (2013). For each query, our tool returns the lists of articles, users, and groups in descending order of ArticleRelevance, UserRelevance and GroupRelevance, respectively. The results we present in this section are with regard to three major adversarial activities: i) botnet; ii) identity theft and credit card fraud; and iii) vulnerability analysis and malicious code development.

Botnet

As we mentioned before, botnet is a serious threat to all networked computers. In order to identify adversaries and their conversations in our dataset related to botnet, we queried the keywords shown in Table 6.3(a) in our tool. Our tool was able to identify 490 articles related to ‘spam’, 44 articles related to ‘botnet’, 9 articles related to ‘zeus’ and 1 article about ‘rustock’.

Then, we checked the results returned by our tool carefully and Table 6.4 shows several interesting articles and their information including the number of comments they received, ArticleRelevance of each article, and authors of these articles. We first noticed one article titled ‘Rustock.C’ with very high ArticleRelevance and ArticleInfluence. This article presented an original analysis of the C variant of Rustock that once accounted for 40% of the spam emails in the world.

Another article titled ‘On startup failure to sign the drivers in Vista x64’ returned by our tool as top relevant article to ‘botnet’ attracting our attention as well. In this article, the author crx.ur discussed about how to load unsigned driver to Windows
<table>
<thead>
<tr>
<th>Keywords</th>
<th>Relevant Articles #</th>
</tr>
</thead>
<tbody>
<tr>
<td>spam</td>
<td>490</td>
</tr>
<tr>
<td>botnet</td>
<td>44</td>
</tr>
<tr>
<td>zeus</td>
<td>9</td>
</tr>
<tr>
<td>rustock</td>
<td>1</td>
</tr>
<tr>
<td>mega-d</td>
<td>0</td>
</tr>
</tbody>
</table>

(a) Results for Botnet

<table>
<thead>
<tr>
<th>Keywords</th>
<th>Relevant Articles #</th>
</tr>
</thead>
<tbody>
<tr>
<td>pin</td>
<td>129</td>
</tr>
<tr>
<td>credit card</td>
<td>93</td>
</tr>
<tr>
<td>carding</td>
<td>1</td>
</tr>
<tr>
<td>credit card sale</td>
<td>0</td>
</tr>
<tr>
<td>ssn</td>
<td>0</td>
</tr>
</tbody>
</table>

(b) Results for Identity Theft and Credit Card Fraud

<table>
<thead>
<tr>
<th>Keywords</th>
<th>Relevant Articles #</th>
</tr>
</thead>
<tbody>
<tr>
<td>vulnerability</td>
<td>418</td>
</tr>
<tr>
<td>shellcode</td>
<td>169</td>
</tr>
<tr>
<td>polymorphic</td>
<td>12</td>
</tr>
<tr>
<td>zero-day</td>
<td>11</td>
</tr>
<tr>
<td>cve</td>
<td>2</td>
</tr>
</tbody>
</table>

(c) Results for Vulnerability Discovery and Malicious Code Development

**Table 6.3:** Results from Our Tool for Queries
Vista x64 by modifying PE file header. The corresponding author claimed that malware vendors would use this technique to build bot and infect thousands of computers. A further investigation on this user shown in Table 6.5 reveals that s/he authored several security-related articles. Her/his profile indicated that s/he was very active in malicious code development and interested in several cybercrime topics, such as rootkit, exploits, and shellcode.

**Identity Theft and Credit Card Fraud**

Identity theft and credit card fraud are both serious issues in nowadays Internet transactions. Online identity theft includes stealing usernames, passwords, social security numbers (ssn), personal identification numbers (PINs), account numbers, and other credentials. Credit card fraud also consists of phishing (the process to steal credit card information), carding (the process to verify whether a stolen credit card is still valid), and selling verified credit card information.

Table 6.3(b) shows results that our tool returned when these keywords are queried. Our tool identified one article that was authored by a user dx_1 related to ‘carding’.
Translated Article Title | # Comments | Translated Interests
--- | --- | ---
The old tale about security | 7 | malware, ring0,
Malcode statistics | 6 | rootkit, botnets,
Cold boot attacks on encryption keys | 2 | asm, exploits,
Wanted Cisco security agent | 2 | cyber terrorism,
Antirootkits bypass | 1 | shellcode, viruses,
Syser debugger | 0 | underground,
Termorektalny cryptanalysis | 0 | Kaspersky,

Table 6.5: Selected Articles by crx_ur and Her/His Information

Table 6.6: Information about dx_ur

in the dataset. A further investigation on this user revealed that s/he was a member of a carding interest group, which had more than 20 members around the world. Table 6.6 shows some basic information of dx_ur. Compared to crx_ur, it is obvious that dx_ur has more interests in financial security issues, such as credit card fraud, web hack, and banking. We could also notice that dx_ur was very active in posting articles and replying others’ posts.
Vulnerability Analysis and Malicious Code Development

We analyzed several keywords related to vulnerability analysis and malicious code development, such as polymorphism (a technique widely used in malware to change the appearance of code, but keep the semantics), cve (a reference-method for publicly-known computer vulnerabilities), shellcode (small piece of code used as the payload in the exploitation of software vulnerabilities), and zero-day (previously-unknown computer vulnerabilities, viruses and other malwares).

As shown in Table 6.3(c), the community is very active in these topics. More than 400 articles related to vulnerabilities were found. However, we noticed most of these articles have low-

ArticleInfluence
. We checked these low-

ArticleInfluence
 articles and discovered that most of them were articles copied from other research blogs and kept the links to original webpages. Our ArticleInfluence index successfully identified these articles were not very novel, thus calculated low ArticleInfluence for them.

At the same time, as shown in Table 6.7, our tool also identified several high-

ArticleInfluence
 vulnerability analysis articles. For example, the article entitled ‘Blind spot’ authored by arx_ur which analyzed a new Windows Internet Explorer vulnerability even attracted 79 replies.

6.6.3 Comparison with HITS Algorithm

In order to evaluate the effectiveness of our approach, we implemented HITS algorithm Kleinberg (1999) in our tool and compared the results with our SocialImpact metrics. HITS algorithm is able to calculate the authorities and hubs in a community by examining the topological structure where authority means the nodes that are linked by many others and hub means the nodes that point to many others. Note that the fundamental difference between SocialImpact and HITS is that SocialImpact
Table 6.7: Selected Top Relevant Articles

<table>
<thead>
<tr>
<th>Translated Article Title</th>
<th># Comments</th>
<th>x2^1</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blind spot</td>
<td>79</td>
<td>793.2</td>
<td>arx.ur</td>
</tr>
<tr>
<td>Seven thirty-four pm PCR</td>
<td>14</td>
<td>146.4</td>
<td>tix.ur</td>
</tr>
<tr>
<td>HeapLib and Shellcode generator under windows</td>
<td>1</td>
<td>15.6</td>
<td>eax.ur</td>
</tr>
<tr>
<td>Who fixes vulnerabilities faster, Microsoft or Apple?</td>
<td>0</td>
<td>5.6</td>
<td>bux.ur</td>
</tr>
<tr>
<td>FreeBSD OpenSSH Bugfix</td>
<td>0</td>
<td>4.2</td>
<td>sux.ur</td>
</tr>
</tbody>
</table>

^1 ArticleRelevance

takes more parameters, such as user-generated content and activity, into account, therefore ranking results are based on a more comprehensive set of social features.

Table 6.8: Top Five Authorities and Hubs by HITS

<table>
<thead>
<tr>
<th>Top Five Authorities</th>
<th>Top Five Hubs</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>auth</td>
</tr>
<tr>
<td>zhengxx.ur</td>
<td>0.506</td>
</tr>
<tr>
<td>crx_xx.ur</td>
<td>0.214</td>
</tr>
<tr>
<td>yuz.ur</td>
<td>0.163</td>
</tr>
<tr>
<td>t1mxx.ur</td>
<td>0.148</td>
</tr>
<tr>
<td>rst.ur</td>
<td>0.143</td>
</tr>
</tbody>
</table>

Comparing the results for authorities and hubs (HITS) shown in Table 6.8 with UserInfluence and UserActiveness (SOCIALIMPACT) in Table 6.2, we can observe that the authorities and hubs have much overlap with HITS algorithm when online conversations are ignored and the results generated by SOCIALIMPACT are different from HITS counterparts.
6.7 Related Work

Computer-aided crime analysis (CACA) utilizes the computation and visualization of modern computer to understand the structure and organization of traditional adversarial networks Xu and Chen (2005). Although CACA is not designed for the analysis of cybercrime, its methods of relation analysis, and visualization of social network are adopted in our work. Zhou et al. Zhou et al. (2005) studied the organization of United State domestic extremist groups on web by analyzing their hyperlinks. Chau et al. Chau and Xu (2007) mined communities and their relationships in blogs for understanding hate group. Lu et al. Lu et al. (2010) used four actor centrality measures (degree, betweenness, closeness, and eigenvector) to identify leaders in hacker community. In contrast, our proposed solution in this dissertation and previous papers Zhao et al. (2011b, 2012b) considers both social relationships and user-generated contents in identifying interesting posts and users for cybercrime analysis.

Systematically bringing order to a dataset has plenty of applications in both social and computer science. With the development of web, ranking analysis in hyperlinked environment received much attention. Kleinberg Kleinberg (1999) proposed a hubs and authorities approach (HITS) by calculating the eigenvectors of a certain matrices associated with the link graph. Almost at the same time, Page and Brin Page et al. (1999b) developed PageRank that uses a page’s backlinks’ sum as its importance index. However, both HITS and PageRank only consider the topological structure of given dataset but ignore its contents Bianchini et al. (2005). Therefore, we devised a ranking system based on personalized PageRank, which is proposed to efficiently deal with ranking issues in different situations Chakrabarti (2007).

In order to provide a safer platform for net-centric business and secure the internet experience for end users, huge research efforts have been invested in defeating malware
and designing defense systems. Hu et al. Hu et al. (2014b,a) proposed systems to enhance firewalls to make sure malicious packets can not bypass existing detecting systems. Gu et al. analyzed botnet C&C channels for identifying malware infection and botnet organization Gu et al. (2007). Zhang et al. Zhang et al. (2013) investigated using high-entropy detectors to detect botnet traffics that employ encryption. Stone-Gross et al. Stone-Gross et al. (2009) took over Torpig for a period of ten days and gathered rich and diverse set of data from this infamous botnet.

6.8 Summary

In this chapter, we have presented a novel approach to help identify adversaries by analyzing social dynamics. We formally modeled online underground social dynamics and proposed SOCIALIMPACT as a suite of measures to highlight interesting adversaries, as well as their conversations and groups. The evaluation of our proof-of-concept system on real-world social data has shown the effectiveness of our approach. As part of future work, we would further test the effectiveness and the usability of our system with subject matter experts.
In this dissertation, we have proposed a pattern-based approach framework, whose goal is to handle previously unknown security events by considering their root causes, randomness and uncertainty and at the same time quantify the confidence of decision making. Different from pre-determined pattern-based approaches, our approach does not rely on fixed bit sequences, but considers randomness and uncertainty. We have shown our approach can cope with previously unknown security events. Different from statistical pattern-based approaches that leave the feature discovery to statistical methods, our pattern-based approaches focuses on using the domain knowledge in each security problem to discover regularities and tackle root causes directly. Therefore, the reasons for such regularities to occur are explainable and interpreted in a security-related way. We have applied our framework to discover and use patterns in code, network, choices, and communities to counter security challenges.

7.1 Contributions

The contributions of this dissertation are as follows:

1. We proposed a framework to discover and use patterns for countering security challenges. We have demonstrated that such a methodology can be used for both defending and attacking in several different stages of the security cycle.

2. We have proposed instruction sequence abstraction as a coarse-grained approach to extract distinguishable features from binary code. We have proposed opcode mnemonic sequence and binary finite-dimensional representation to represent
arbitrary shellcode sample. We have proposed a shellcode detection approach based on Markov model and a shellcode attribution approach with support vector machines. We have investigated the feature selection methods and demonstrate its effectiveness with our collected large set of shellcode samples. With our developed tool, we have demonstrated the identified top opcode transition patterns. We have evaluated the effectiveness of our shellcode detection and attribution approaches with collected real-world shellcode samples. In addition, we have compared the strengths of real-world shellcode engines with several metrics.

3. We have proposed using routing table changes as evidence to measure the risk of node isolation in mobile ad hoc networks. We have identified several categories of routing table change patterns and measure the impact of each routing table change category. We have proposed an extended Dempster-Shafer evidence model with importance factors and articulate expected properties for Dempster’s rule of combination with importance factors. Our Dempster’s rule of combination with importance factors is nonassociative and weighted. We have proposed an adaptive risk-aware response mechanism with the extended D-S evidence model, considering damages caused by both attacks and countermeasures. The evaluation has shown the effectiveness of our approach in multiple MANET attack scenarios.

4. We have performed an empirical analysis on collected PGA passwords to understand user choice patterns in background picture, gesture location, gesture order, and gesture type. We have proposed a selection function model which models the thinking process of users when they choose picture passwords. We have demonstrated our approach to automatically extract picture password compo-
sition processes with this model. We have implemented our attack framework and evaluated its effectiveness on the collected dataset in several different attack models which includes targeted attack, nontargetted attack, online attack and offline attacks.

5. We have studied the role of underground social dynamics in the whole underground economy and collected a dataset of the user interactions of an underground social network. We have presented an online underground social network model which takes users and their interactions into account. Based on this model, we have demonstrated our approaches to identify influential players and to associate security events with their behind-scene players in underground social networks. We have evaluated our approaches on a real-world underground social network dataset and discovered intelligences of underground society from this dataset and linked existing attack events with discussions in this social data.

7.2 Future Work

For future work, we plan to use the proposed pattern-based framework to counter more security challenges. For each challenge we tried to tackle in this dissertation, we plan to explore more sophisticated models by considering more factors.
REFERENCES


David Anselmi, N. S., Jimmy Kuo, Microsoft Security Intelligence Report Volume 9 (Microsoft, 2010).


Pace, Z., “Signing into windows 8 with a picture password”, http://www.youtube.com/watch?v=Ek9N2tQzHOA (2011b).


Ros, F., “UM-OLSR implementation (version 0.8.8) for NS2” , (2007).


Yuille, J. C., Imagery, memory, and cognition (Lawrence Erlbaum Assoc Inc, 1983).


