**IWAISe:** First International Workshop on Artificial Intelligence in Security

An IJCAI Workshop

Melbourne, 20 Aug 2017

Social Media: #IWAISe #IJCAI

Web: [http://iwaise.it.nuigalway.ie/](http://iwaise.it.nuigalway.ie/)
Copyright Notice

These proceedings are licensed under the Creative Commons Attribution 4.0 International License.

Editors

Brett Drury  National University of Ireland Galway, Ireland  
Michael Madden  National University of Ireland Galway, Ireland  
Barry O’Sullivan, University College Cork, Ireland  
Jo Ueyama  University of Sao Paulo, Brazil
## Programme Committee

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution/Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brett Drury</td>
<td>National University of Ireland Galway, Ireland</td>
</tr>
<tr>
<td>Michael Madden</td>
<td>National University of Ireland Galway, Ireland</td>
</tr>
<tr>
<td>Jo Ueyama</td>
<td>University of Sao Paulo, Brazil</td>
</tr>
<tr>
<td>Barry O’Sullivan</td>
<td>UCC, Ireland</td>
</tr>
<tr>
<td>Luis Paulo Reis</td>
<td>FEUP-University of Porto, Portugal</td>
</tr>
<tr>
<td>Charles Wood</td>
<td>Capco, United Kingdom</td>
</tr>
<tr>
<td>Spiros Antonatos</td>
<td>IBM, Ireland</td>
</tr>
<tr>
<td>Stefano Braghin</td>
<td>IBM, Ireland</td>
</tr>
<tr>
<td>Ricardo Morla</td>
<td>University of Porto, Portugal</td>
</tr>
<tr>
<td>Jorge Pinto</td>
<td>University of Minho, Portugal</td>
</tr>
<tr>
<td>Charles Gillan</td>
<td>Queens University Belfast, United Kingdom</td>
</tr>
<tr>
<td>Tahar Kechadi</td>
<td>UCD, Ireland</td>
</tr>
<tr>
<td>Peter Corcoran</td>
<td>National University of Ireland Galway Galway, Ireland</td>
</tr>
<tr>
<td>Noel O’Connor</td>
<td>Dublin City University, Ireland</td>
</tr>
<tr>
<td>Gabriel Pestana</td>
<td>Technical University of Lisbon, Portugal</td>
</tr>
<tr>
<td>Mario Nunes</td>
<td>INESC-TEC, Portugal</td>
</tr>
<tr>
<td>Manuel Ruiz de Quintanilla</td>
<td>Aeorum, Spain</td>
</tr>
<tr>
<td>Jesus Merino</td>
<td>Aeorum, Spain</td>
</tr>
<tr>
<td>Stefano Zampolli</td>
<td>CNR-IMM, Italy</td>
</tr>
<tr>
<td>Veronica Carvalho</td>
<td>UNESP, Brazil</td>
</tr>
<tr>
<td>Marcia Oliveira</td>
<td>Skim Technologies, United Kingdom</td>
</tr>
<tr>
<td>Nhi-en-An Lekhac</td>
<td>UCD, Ireland</td>
</tr>
<tr>
<td>Lilian Berton</td>
<td>UNIFESP, Brazil</td>
</tr>
<tr>
<td>Henrietta Eyre</td>
<td>Adarga, United Kingdom</td>
</tr>
<tr>
<td>Brian Lee</td>
<td>AIT, Ireland</td>
</tr>
<tr>
<td>Suzanne Little</td>
<td>DCU, Ireland</td>
</tr>
<tr>
<td>Aoife Duffy</td>
<td>National University of Ireland Galway Galway, Ireland</td>
</tr>
<tr>
<td>Frank Glavin</td>
<td>National University of Ireland Galway Galway, Ireland</td>
</tr>
<tr>
<td>Ihsan Ullah</td>
<td>National University of Ireland Galway Galway, Ireland</td>
</tr>
</tbody>
</table>
Contents

I Preface 1

II Invited Speakers 3

III Research Articles 6

- Dynamic Decisions and Adaptive Allocations: Robust Planning for Physical and Cyber Threat Screening Games: Josip Bozic and Franz Wotawa ........................................ 7
- Handling Continuous Space Security Games with Neural Networks: Nitin Kamra, Debarun Kar, Fei Fang, Yan Liu and Milind Tambe 15
- Bitcoin's Security Model Revisited: Yonatan Sompolinsky and Aviv Zohar ........................................ 23
- Partially Observable Contingent Planning for Penetration Testing: Dorin Shmaryahu, Guy Shani, Joerg Hoffmann and Marcel Steinmetz 32
- Ranking Vulnerability Fixes Using Planning Graph Analysis: Tom Gonda, Guy Shani, Rami Puzis and Bracha Shapira 40
- Planning the Attack! Or How to use AI in Security Testing?: Josip Bozic and Franz Wotawa ........................................ 49
- Deep Secure: A Fast and Simple Neural Network based approach for User Authentication and Identification via Keystroke Dynamics: Saket Maheshwary, Soumyajit Ganguly and Vikram Pudi 58
- How to Pick Your Friends - A Game Theoretic Approach to P2P Overlay Construction: Saar Tochner and Aviv Zohar 66

IV Posters 74

- An Intelligent Context-Aware Biometrics System Based on Agent Technology: Fatina Shukur and Harin Sellahewa 75
- ROCSAFE: Remote Forensics for High Risk Incidents: Brett Drury, Nazli Bagherzadeh and Michael Madden 81
- Open Social Data Crime Analytics: Ihsan Ullah, Caoilfhionn Lane, Brett Drury, Marc Mellote and Michael Madden 84

V Appendix 87
Acknowledgements

This Workshop has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No. 700264, ROCSAFE (Remotely Operated CBRNe Scene Assessment and Forensic Examination).
Part I

Preface
Introduction

Security, for both digital and physical systems, is at a crossroads where traditional techniques are no longer adequate. The recent high-profile breaches of virtual and physical barriers have demonstrated the need for new approaches to security. The current rapid evolution of artificial intelligence provides an avenue from which security researchers and practitioners can benefit. Artificial intelligence may allow the identification and the deployment of counter-measures. This may range from the simple recognition of social engineering emails to automated physical protection of borders and valuable assets.

The field is at its early stage, and some of the promises and publicity surrounding artificial intelligence does not reflect the current state of research in both the academic and private sector. The International Workshop on A.I. in Security (IWAIse) is designed to bring together researchers from industry and academia in one place to provide a more accurate representation of the state of the art in the area. The workshop is intended to be the first in a series.

In this, its first year, the workshop accepted eight papers for oral presentation and four for poster presentation. Each paper was reviewed by at least two referees. The breadth of AI-based security research is evident from them. For example, the papers by Bozic and Wotawa [2017a], [Bozic and Wotawa [2017b] and Shmaryahu et al. 2017] all consider defence against physical/virtual penetration threats, while those by Tochner and Zohar 2017 and Sompolinsky and Zohar 2017 are concerned with different aspects of Bitcoin’s security model. The papers by Kamra et al. 2017 and Maheshwary et al. 2017 both make use of neural networks, but for different applications. The paper by Gonda et al. 2017 focuses on identifying most critical vulnerabilities. Meanwhile, the accepted posters cover applications ranging from biometrics to robotics, and from security games to social data.

We are also pleased to have two very interesting invited talks: Prof. Noa Agmon of Bar-Ilan University speaking on "Robotic Strategic Behavior in Adversarial Environments", and Prof. Bo An of Nanyang Technological University speaking on "Recent Progress on Computational Game Theory for Security".

Michael G. Madden (National University of Ireland Galway)
Brett Drury (National University of Ireland Galway)

Galway, Ireland, 7th August 2017.
Part II

Invited Speakers
Noa Agmon

Biography

Noa Agmon is an assistant professor at the Computer Science department at Bar-Ilan University (BIU). Her research focuses on multi-robot systems, while using both theoretical and empirical means for evaluation of team performance guarantees on a variety of robotic tasks, for example multi-robot patrol and robot navigation. She received her PhD from Bar-Ilan University (2009), and her MSc from the Weizmann Institute (2004), and spent two years at The University of Texas (UT) at Austin before accepting the faculty position at BIU, where she established and heads the Security Robotics Lab.

Talk: Robotic Strategic Behavior in Adversarial Environments

Robots act in adversarial environments. It is a fact. Unfortunately, little research has been done in the robotics community on strategic robotic behavior considering the existence of an adversary. This talk summarizes recent research achievements in the emerging area of Adversarial Robotics: accounting for adversarial presence in robotic tasks. This will be demonstrated by four different problems: multi-robot patrolling, robotic coverage, robot-team formation, and robot navigation. We have shown that considering an adversary leads to a more general problem, where operating in neutral environments (as has been done so far) is actually an instance of this problem, that assumes a specific (usually simple, random) adversarial model.

Bo An

Biography

Bo An is a Nanyang Assistant Professor with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. He received the Ph.D degree in Computer Science from the University of Massachusetts, Amherst. His current research interests include artificial intelligence, multiagent systems, game theory, and optimization. He has published over 70 referred papers at AAMAS, IJCAI, AAAI, ICAPS, KDD, JAAMAS, AIJ and ACM/IEEE Transactions. Dr. An was the recipient of the 2010 IFAAMAS Victor Lesser Distinguished Dissertation Award, an Operational Excellence Award from the Commander, First Coast Guard District of the United States, the Best Innovative Application Paper Award at AAMAS-12, the 2012 INFORMS Daniel H. Wagner Prize for Excellence in Operations Research Practice, and the Innovative Application Award at IAAI-16. He was invited to give Early Career Spotlight talk at IJCAI-17. He led the team HogRider which won the 2017 Microsoft Collaborative AI Challenge. He is a member of the editorial board of JAIR and the Associate Editor of JAAMAS. He was elected to the board of directors of IFAAMAS.
Talk: Recent Progress on Computational Game Theory for Security

Security is a critical concern around the world, whether it's the challenge of protecting ports, airports and other critical national infrastructure, or protecting wildlife and forests, or suppressing crime in urban areas. In many of these cases, limited security resources prevent full security coverage at all times; instead, these limited resources must be scheduled, avoiding schedule predictability, while simultaneously taking into account different target priorities, the responses of the adversaries to the security posture and potential uncertainty over adversary types. Computational game theory can help design such unpredictable security schedules and new algorithms are now deployed over multiple years in multiple applications for security scheduling. These applications are leading to real-world use-inspired research in computational game theory in scaling up to large-scale problems, handling significant adversarial uncertainty, dealing with bounded rationality of human adversaries, and other interdisciplinary challenges. This talk will discuss some recent research progress on computational game theory for security based on results published at recent AAMAS/AAAI/IJCAI conferences.
Part III

Research Articles
Dynamic Decisions and Adaptive Allocations: Robust Planning for Physical and Cyber Threat Screening Games

Sara Marie Mc Carthy, Phebe Vayanos, Milind Tambe
University of Southern California
{saramarm,phebe.vayanos,tambe}@usc.edu

Abstract

We consider the problem of dynamically allocating screening resources of different efficacies (e.g., X-ray imaging, deep packet inspection, cyber analysts) at checkpoints (e.g., at airports or ports, enterprise networks) to determine the threat level of the incoming screenees. Previously, the Threat Screening Game model was introduced to address this problem under the assumption that screenee arrival times are perfectly known. In reality, arrival times are uncertain, which severely impedes the implementability and performance of this approach. We thus propose a novel framework for dynamic allocation of threat screening resources that explicitly accounts for uncertainty in the screenee arrival times. We model the problem as a multistage robust optimization problem and propose a tractable solution approach using compact linear decision rules combined with robust reformulation and constraint randomization. We perform extensive numerical experiments which showcase that our approach outperforms (a) exact solution methods in terms of tractability, while incurring only a very minor loss in optimality, and (b) methods that ignore uncertainty in terms of both feasibility and optimality.

1 Introduction

Screening for threats is an important security challenge, be it inspecting cargo at ports, alerts generated by computer security systems, passengers at airports, or fans entering a stadium. Given a strategic adversary capable of exploiting gaps in security measures, along with a large number of screenees, it becomes critical to optimize the allocation of limited screening resources. In domains such as cyber security not only can the number of screenees be overwhelmingly large, but there are also many different types of attacks that can occur, which in turn may each require a different screening method to detect.

Threat Screening Games (TSGs) have been previously introduced to model screening domains as bayesian Stackelberg games. These games model situations where the screener attempts to screen for threats, while a strategic attacker attempts to penetrate security. They have been used to model both physical and cyber threat screening; in the context of airport security [Brown et al., 2016; Schlenker et al., 2016] where there may be a terrorist trying to pass through airport screening. A sub-category of these games, known as Cyber Allocation Games (CAG)[Schlenker et al., 2017] looks at screening suspicious activity alerts generated by and intrusion detection systems (IDS) to identify a hacker attempting to penetrate a computer network. Optimizing the defender (mixed) strategy in such games helps optimize the limited screening resources against a strategic adversary. TSGs provide a better model for strategic settings than ones that do not take strategic adversaries into account. [Tambe, 2011; Korzhyk et al., 2010; Yin et al., 2015; Balcan et al., 2015; Basilico et al., 2009; Letchford and Vorobeychik, 2011; Gan et al., 2015; Guo et al., 2016], where a defender protects a set of targets from a strategic adversary. However, TSGs differ significantly because they (i) do not have an explicitly modeled set of targets; (ii) include a large number of non-player screenees that must be screened while a single adversary attempts to pass through undetected; and (iii) encompass screening resources with differing efficacies and capacities that are combined to work in teams. These key differences make TSGs more appropriate for screening settings.

In previous work, despite promising results, previous work in TSG fails in its mission to realistically model real-world settings. Its fundamental limitation is its assumption of perfect fore-knowledge of screenee arrival times (e.g., arrival times of packets in networks, alerts generated by IDS, or passengers at airports). However, in the real-world there is significant uncertainty in arrival times. Addressing this challenge is difficult, as it requires reasoning about all the possible realizations of the uncertainty and coming up with an optimal plan for each of those scenarios. When dealing with a large number of screenees, this result in millions of possible scenarios, making the planning problem extremely difficult.

To address this shortcoming, our first contribution is a new model Robust Threat Screening Games (RTSG), which expresses the required uncertainty in screenee arrival times. In RTSG, we model the problem faced by a screener as a robust multistage optimization problem. We present a tractable solution approach with three key novelties that contribute to its efficiency: (i) compact linear decision rules; (ii) robust re-
formulation; and (iii) constraint randomization. We present extensive empirical results that show that our approach outperforms the original TSG methods that ignore uncertainty, and the exact solution methods that account for uncertainty. While dealing with uncertainty has been previously addressed in security games [Kiekintveld et al., 2011; Yin et al., 2011; Kiekintveld et al., 2013] these novel techniques for handling uncertainty have not been previously explored and thus provide additional contribution not only to TSG, but to the general security game literature.

2 TSG Problem Formulation

2.1 The Case when Screenee Arrivals are Known

Time Windows. We consider a finite planning horizon consisting of \( W \) time windows (periods) \( \mathcal{W} := \{1, \ldots, W\} \).

Screenee Categories. During each period, a known number of screenees arrive, each from a known category \( \kappa := (\rho, \phi) \), \( \rho \in \mathcal{P} := \{1, \ldots, P\} \), \( \phi \in \mathcal{F} := \{1, \ldots, F\} \). The first (second) component of their category, \( \rho \) (\( \phi \)), represents the uncontrollable (resp. controllable) part of the screenee’s category. Thus, each screenee can decide the controllable part of their category, however, they cannot decide the uncontrollable part of their category, which stems from their inherent characteristics. For notational convenience, we let \( \mathcal{K} := \mathcal{P} \times \mathcal{F} \).

We assume that each screenee knows their own category. As an example, in the context of passenger screening at airports, \( \rho \) can represent the risk category of the passenger (e.g., normal boarding versus TSA pre-check), while \( \phi \) can represent a flight type (e.g., international with given departure time) – note that both these components are known to the passenger.

In CAG’s where screenees correspond to alerts, we let \( \mathcal{N}_w^\kappa \) denote the number of screenees in category \( \kappa \) to arrive in time window \( w \). Since the category and arrival time of each screenee is known, the quantities \( \mathcal{N}_w^\kappa \) are perfectly known. Without loss of generality, we assume that \( \mathcal{N}_w^\kappa > 0 \) for all \( w \) and \( \kappa \).

Adversary Actions. One of the screenees is planning on conducting an attack using an attack method \( m \) of his choosing from the set \( \mathcal{M} \). The adversary selects \( a \) his attack method \( m, b \) his attack window \( w \), and \( c \) the components of his category that he can control in \( \kappa \), so as to cause maximum harm.

We refer to such a choice as an attack \( (m, w, \kappa) \). In the context of airport security such an attack may be a concealed weapon, or liquid explosive. In CAG’s this may refer to denial of service attacks, malware, web exploitation, or social engineering attacks.

Defender Actions. The adversary’s attack can be averted by adequate screening. For this reason, the screener is operating a checkpoint comprised of \( T \) teams indexed by \( t \in \mathcal{T} \) and can decide which team should screen each screenee based on their category. Each of these teams consists of various resource types. The set of all available resource types is denoted by \( \mathcal{R} \). The subset of resources composing team \( t \) is denoted by \( \mathcal{R}(t) \subseteq \mathcal{R} \). If a screenee is assigned to team \( t \), then he

must be screened by all resource types allocated to that team. Such resources may be physical screening devices such as x-ray machines, or walk through metal detectors in airports or human analyst assigned to resolve alerts in CAGS. Unfortunately, not all screenees can be screened by the most effective resources as each resource has a capacity \( C_r \) on the number of screenees that it can process in each time window. The attack will be averted if the attack method is identified by any one of the resources screening the attacker. We let \( E_{t,m} \) denote the effectiveness (i.e., probability of interception) of team \( t \) at detecting attack method \( m \), determined by the effectiveness of each resource. Assuming independence of the effectiveness of the resources that make up each team and letting \( E_{t,m} \) denote the probability of detecting an attack of type \( m \) using resource \( r \), we have \( E_{t,m} = 1 - \prod_{t \in \mathcal{R}(t)} (1 - E_{r,m}(t)) \).

Following the (by now standard) approach in the literature, we formalize this problem as a Threat Screening Game, i.e., a Stackelberg game in which the screener, as the leader, commits to mixed strategies, and the attacker acts as the follower [Brown et al., 2016; Schlenker et al., 2016]. The rationale is that the screener acts first by selecting a (randomized) screening strategy, i.e., a feasible assignment of screenees to teams. In response to the choice of screening strategy, the attacker (after observing the screenee allocation) selects an attack \((m, w, \kappa)\). If the attack is caught, the screener receives a utility \( U^\kappa_s \), which depends on the category of the adversary. Accordingly, if the screener is unsuccessful at preventing the attack, he receives the (negative) utility \( U^\kappa_a \). The attacker’s utilities are assumed to be negative of the screener’s utilities, so that the game is zero-sum. We assume that the defender knows the probability that the attacker’s uncontrollable category is \( \rho \), denoted by \( P_\rho \), and we have \( \sum_{\rho \in \mathcal{P}} P_\rho = 1 \).

The objective of the screener is then to select the best randomized allocation (i.e., mixed strategy), in anticipation of the attacker’s best response.

We are now ready to provide a mathematical formulation of the TSG problem in the spirit of [Brown et al., 2016].

Defender Pure Strategy Set. An assignment of screenees to teams occurs at the beginning of each period \( w \in \mathcal{W} \), and corresponds to a decision on the number of screenees from each category \( \kappa \) to allocate to each team \( t \) out of the \( N_w^\kappa \) screenees that arrive in that time window. Letting \( \nu_{w,t}^\kappa \) denote this assignment, the defender pure strategy set is given by

\[
S := \left\{ \nu : \nu_{w,t}^\kappa \in \mathbb{N}_+ \forall t \in \mathcal{T}, \sum_{t \in \mathcal{T}} \nu_{w,t}^\kappa = N_w^\kappa \forall \kappa \in \mathcal{K}, \right. \\
left. \sum_{t \in \mathcal{T}} \sum_{\kappa \in \mathcal{K}} \nu_{w,t}^\kappa \leq C_r \forall r \in \mathcal{R}, w \in \mathcal{W} \right\}.
\]

The first constraint in the set stipulates that the number of screenees must be a non-negative integer. The second ensures that all the screenees are allocated to a team. The third guarantees that resource capacities are not exceeded. Note that \( S \) has finite cardinality, i.e., there are finitely many pure strategies available to the screener. The probability of detecting an
attack \((m, w, \kappa)\) given defender strategy \(s\) is given by
\[
D_w^s\kappa,m := \sum_{t \in T} E_{t,m} \nu_{w,s}\kappa,t / N_{w,\kappa},
\]
where \(\nu_{w,s}\kappa,t\) denotes the number of screenees in category \(\kappa\) screened by team \(t\) in window \(w\) according to pure strategy \(s\).

**Defender Mixed Strategies.** A mixed strategy corresponds to a distribution over pure strategies, i.e., to a choice
\[
q \in Q := \left\{ (q_k)_{k \in S} : \sum_{k \in S} q_k = 1, q_k \geq 0 \right\}.
\]
The probability of detecting an attack \((m, w, \kappa)\) is given by
\[
\sum_{k \in S} q_k D_w^s\kappa,m.
\]

**Robust Linear Programming Formulation.** Since the attacker can select the attack \((m, w, \kappa)\), but cannot select the uncontrollable aspect of his category, the problem faced by the screener is expressible as the following robust optimization problem in variables \(z\) and \(q\)
\[
\begin{align*}
\text{maximize} \quad & \sum_{k \in P} P_k \left[ z_{w,m}^k U_{\kappa}^k + (1 - z_{w,m}^k) U_{\kappa}^k \right] \\
\text{subject to} \quad & z_{w,m}^k = \sum_{\xi \in S} q_{\xi} D_w^{s_{\kappa,m}} \quad \forall \kappa, m, w \\
& \pi \in P
\end{align*}
\]

The first constraint is a direct consequence of the first constraint in Problem (1) combined with the change of variables, and
\[
\begin{align*}
H := \left\{ \pi : \sum_{\xi \in R(t)} \sum_{r \in K} \pi_{\xi,r,k}^w N_{w,\kappa}^r \leq C_r \quad \forall r, W \\
& 0 \leq \pi_{\xi,r,k}^w \leq 1 \quad \forall \xi, r, k, w \}
\end{align*}
\]
denotes the set of all marginal strategies. We note that Problem (2) is equivalent to a moderately sized linear program obtained by linearizing the piecewise linear concave objective function using the standard epigraph reformulation approach.

### 2.2 RTSG: The Case of Uncertain Screenee Arrivals

Insofar, we have assumed that screenee arrival times are perfectly known. Unfortunately, this assumption fails to hold in most threat screening problems. Moreover, ignoring uncertainty in the screenee arrivals during optimization may yield severely suboptimal or even infeasible allocations, see Section 4. We thus develop RTSG (Robust Threat Screening Game), a novel modeling and solution framework for threat screening that is robust to uncertainty in screenee arrival times. Our framework builds upon formulation (2) which enjoys better tractability properties than Problem (1).

**Model of Uncertainty.** We model the number of screenees from each category to arrive in each time window as random variables that are defined on the probability space \((\Xi, F, P)\), which consists of the sample space \(\Xi\), the Borel \(\sigma\)-algebra \(F\) and the probability measure \(P\). The elements of the sample space are denoted by \(\xi := (\xi_0, \xi_1, \ldots, \xi_W)\) where the subvector \(\xi_w := (\xi_0, \ldots, \xi_w)\) denotes the number of people from category \(\kappa\) that arrive in window \(w\). We also let \(\xi^w := (\xi_0, \ldots, \xi_w)\) denote the portion of the \(\xi\) that has been observed by the end of time window \(w\). We assume that \(\Xi\) is a bounded set expressible as
\[
\Xi := \{ \xi : \xi_{w,k} \in \mathbb{N}, \forall \xi \leq h \}
\]
for some matrix \(V \in \mathbb{R}^{l \times W}\) and vector \(h \in \mathbb{R}\), where \(l\) corresponds to the number of constraints in the uncertainty set. Thus \(\Xi\) corresponds to the intersection of the set of all non-negative integers with a polyhedral set. Without loss of generality, we assume that \(\Xi \subseteq \{ \xi : \xi_0 = 1 \}\) (since \(w = 0\) is not a valid time period, we let \(\xi_0\) be a constant, so that affine functions of \((\xi_0, w)\) can be represented compactly as linear functions of \(\xi\)). We assume that \(\Xi\) is bounded. In the spirit of robust optimization, we refer to \(\Xi\) as the uncertainty set. Polyhedral uncertainty sets allow for a lot of modeling flexibility and enable us to capture a wide variety of constraints of practical relevance. In particular we can model the uncertainty present in security screening at airports using such a set.

**Example 1 (Airport Screening).** In the context of security screening at airports, the total number of people to travel in category \(\kappa\) on a given day, denoted by \(N_\kappa\) is known from the flight manifests. At the same time, passenger arrival times are conditioned by the time of their flight category \(\phi\). It is thus natural to assume that all passengers in category \(\kappa\) will arrive in some window \(w \in \Delta_\kappa \subseteq W\) (covering e.g., a couple of hours before their flight time). A suitable choice of uncertainty set is then given by
\[
\Xi_{AS} := \left\{ \xi : \xi_{w,k} \in \mathbb{N}_+, \sum_{w \in \Delta_\kappa} \xi_{w,k} = N_\kappa \forall \kappa \right\},
\]
which we denote by AS for Airport Screening.
In the context of cyber security, we may only have estimates $N_κ$ of the total number of alerts of each type $κ$ and may not require that all the estimated screenee’s or alerts arrive. We can additionally encode dependencies among alert types, so that certain alerts or attacks may only proceed from specific observed sequences of alerts.

In this paper, we take the view of a risk-averse screener that wishes to be immunized against all possible realizations of $ξ ∈ Ξ$. This view point is very natural for the set of applications under consideration that fall under the realm of security. This implies that the attacker can in some sense “strategize with nature” to devise a maximally harmful attack. Equivalently, it can be interpreted as the desire to be immunized against an attacker who would, by his own fortune, select the maximally harmful attack relative to uncertainty in arrivals.

Adaptive Screening. As information about screenee arrivals is revealed sequentially over time, the screener has the opportunity to adjust his screening policy in an adaptive fashion, at the beginning of each time window, in response to these observations. In particular, at the beginning of time window $w$, the screener has observed the sequence of past arrivals $ξ^{w-1}$ and can use that information to reason about uncertainty in remaining time windows and adjust his screening strategy accordingly. Mathematically, the screening decisions made at the beginning of time window $w$ (i.e., $ξ^w$) in Problem (2) must be modeled as functions of the history of screenee arrivals $ξ^{w-1}$. Given a realization $ξ^{w-1}$ of $ξ^{w-1}$, the screener will allocate $π^{w}_κ,ξ,κ,κ,m,w$ percent of screenees in $κ$ to team $t$ in window $w$. Accordingly, the probability of intercepting an attacker from category $κ$ using attack method $m$ in time window $w$ (i.e., $ξ^{w}$) also depends on the realization of $ξ^{w-1}$ and must be modeled as a function of the history of observations, i.e., we have $ξ^{w}$.

Resource Overflow. When arrivals are uncertain, the resource capacity constraint in (2) reads

$$\sum_{t ∈ T} \sum_{r ∈ R(t)} π^w_{κ,m}(ξ^{w-1})ξ_{κ,m} ≤ C_r + \alpha^w_{κ,m}(ξ^{w-1}) + \alpha^{w+1}_{κ,m}(ξ^w)$$

and is enforced for all $r ∈ R, w ∈ W$, and $ξ ∈ Ξ$.

Adaptive Robust Optimization Formulation. We now formulate the screener’s problem as a multi-stage robust optimization problem. We note that if the attacker chooses category $κ$ and time window $w$ for his attack, at least one screenee in category $κ$ (corresponding to the attacker) must arrive in that time window, i.e., it must hold that $ξ_{κ,m} > 0$. The screener’s problem may be formulated in epigraph form as

$$\begin{align*}
\text{maximize} & \quad θ \\
\text{subject to} & \quad \sum_{ρ ∈ P} P_{ρ} u_ρ - \sum_{w∈W} \sum_{r∈R} F_r θ^w_ρ,ξ^w ∀ξ^w \\
 & \quad u_ρ ≤ \pi^w_{κ,m},ξ^w ∀ξ^w,κ,m \in W, ∀w ∈ W, ∀ς \in Ξ, r ∈ R \\
 & \quad θ^w_ρ ≤ \pi^w_{κ,m}(ξ^{w-1}) + (1 - \pi^w_{κ,m})U^w_r ∀ξ^w,κ,m \in W, ∀w ∈ W, ∀ς \in Ξ, r ∈ R \\
 & \quad π^w_{κ,m} ∈ Π_0, \quad ∀w ∈ W, ∀ς \in Ξ.
\end{align*}$$

The decision variables of Problem ( Panthers ) are $θ ∈ R, u_ρ(ξ), θ^w_ρ(ξ^{w-1}), θ^w_{κ,m}(ξ^{w-1}), θ^w_{κ,m}(ξ^{w-1}) ∈ R$, and

$$Π_0 := \left\{ \begin{array}{l}
ξ^w > 0 : \text{Constraint (4)} ∀ξ^w, r, w \\
π^w_{κ,m} ≥ 0 : \text{Constraint (4)} ∀ξ^w, r, w \\
0 ≤ π^w_{κ,m} ≤ 1 ∀t \end{array} \right\}$$

We omit the dependence on $ξ$ to minimize notational overhead. The variables $u_ρ(ξ)$ express the utility of the screener in scenario $ξ$ when the uncontrollable category of the screener is $ρ$. The remaining variables admit the same interpretation as in Section 2.1. In the present setting they are however adaptive. The first set of constraints is used to linearize the piecewise linear concave objective function. The second set of constraints determines the worst-case value of $u_ρ(ξ)$ for each scenario $ξ$. For any given choice of $(φ, ζ)$ by the attacker, this constraint is only enforced over those $ξ ∈ Ξ$ for which $ξ_{κ,m} > 0$ since at least one screenee must arrive in the attacker’s chosen category and attack window. The following Proposition establishes correctness of the above formulation by showing equivalence of Problem ( Panthers ) and an appropriately constructed robust dynamic program.

**Proposition 1.** The multi-stage robust optimization problem ( Panthers ) computes the optimal defender screening strategy, which maximizes his worst-case expected utility when screenee arrivals are uncertain. It is always feasible. 

\(^{1}\)All proofs can be found in the appendix at: https://www.dropbox.com/s/jmwo0nt986wu0p2/appendix.pdf?dl=0
Complexity. Since $\Xi$ is discrete and bounded, Problem ( $\mathcal{P}$) is equivalent to a deterministic linear program obtained by enumerating all possible realizations of $\xi \in \Xi$ and imposing appropriate non-anticipativity constraints, in the spirit of scenario-based stochastic programming [Birge and Louveaux, 1997]. While the numbers of decision variables and constraints in that problem is linear in the number of scenarios, the number of scenarios (cardinality of $\Xi$) can grow very large. In particular for the airport security setting, we show that the number of decision variables and constraints grow exponentially with the number of categories and time windows.

Complexity of Airport Screening. Consider the uncertainty in passenger arrivals. For any fixed screenee category $\kappa$, the number of possible ways in which these screenees may arrive is $g := \binom{N_\kappa+|\Delta_\kappa|-1}{N_\kappa}$.

For fixed $|\Delta_\kappa|$ this quantity is $O(N_\kappa^{|\Delta_\kappa|})$; and for fixed $N_\kappa$, it is $O(|\Delta_\kappa|^{N_\kappa})$. Since passenger arrivals are independent across different categories, the cardinality of $\Xi$ is given by $g^{|K|}$ and is thus exponential in the number of categories. In the context airport screening, the number of scenarios is thus exponential in the number of flight categories. In addition, both the number of flight categories and corresponding number of passengers are generally linear in the number of time windows. This implies that the size of the corresponding scenario problem is exponential in the number of time windows.

3 Proposed Solution Approach

Problem ( $\mathcal{P}$) can become computationally expensive to solve for realistic size instances where the cardinality of $\Xi$ is exponential in the number of time windows, see Example 2.2. We thus propose a solution approach that results in a tractable problem even when $\Xi$ has exponentially many scenarios. In what follows, we describe our approach and main results. The proofs can be found in the Appendix.\(^2\)

3.1 Linear Decision Rule Approximation

Information Aggregation. In Problem ( $\mathcal{P}$), the decision variables $\pi^w$ are modeled as functions of the entire vector of past arrival realizations $\xi^{w-1}$. As a first step to obtain a tractable problem we propose to reduce information available to the screener and only allow his screening policy to adapt to the aggregate number of screenees that have arrived in past windows. Thus, we model the screening policy $\pi^w$ for time window $w$ as a function of the aggregate information $\zeta^{w-1} := \{\xi^{w-1,\kappa}\}_{\kappa \in K}$, where $\zeta^{w-1,\kappa} := \sum_{\xi^{w,m} \in \Xi^{w,m}} \xi^{w,m}$. The following proposition shows that this results in a conservative approximation to the optimal screening policy, since the restricted policy lies within the space of feasible policies.

**Proposition 2.** Restricting the adaptive decision variables $\pi^w$ and $z^w$ for each time window $w \in W$ to be functions of the aggregate information vector $\zeta^{w-1}$ provides a lower bound on the optimal objective value of Problem ( $\mathcal{P}$).

However, even when restricting $\pi$ to be functions of the aggregate arrival $\zeta^{w}$, $a^w$ and $u_\rho$ are still functions of the full passenger arrival $\xi^{w-1}$. The overflow in time window $w$ is a function of not only $\xi^{w}$ but all $\xi^i$, $\forall i \leq w$ since $a^w \geq \sum_{i \leq w} (\pi^i_{\kappa\ell} C_i)$. Additionally, $u_{\rho}$ depends on $\zeta^{w}$ for all $w$, which is equivalent to knowing the actual passenger arrival $\xi$. Since restricting $a^w$ and $u_{\rho}$ to be functions of $\zeta^{w-1}$ would result in further loss of optimality we avoid it here.

Linear Decision Rule. In Problem ( $\mathcal{P}$), the decision variables of the problem are arbitrary (bounded) functions of the uncertain parameter realizations. As a second step to obtain a tractable problem, we propose to restrict the space of feasible adaptive decisions to those that exhibit affine dependence on the data in the spirit of [Ben-Tal et al., 2004]. Thus, we let

\[\pi^w_{\kappa,t}(\zeta^{w-1}) = (\pi^{w}_{\kappa,t})^T \zeta^{w-1} \forall \kappa, t, w, \xi\]
\[z^w_{\kappa,m}(\zeta^{w-1}) = (z^w_{\kappa,m})^T \zeta^{w-1} \forall \kappa, m, w, \xi\]
\[a^w_{\kappa}(\zeta^{w-1}) = (a^w_{\kappa})^T \zeta^{w-1} \forall \kappa, w, \xi\]
\[u_{\rho}(\xi) = u_{\rho}^\top \xi \forall \rho, \xi\]

where the vectors $\pi^w_{\kappa,t}$, $z^w_{\kappa,m}$, $a^w_{\kappa}$, $u_{\rho}$ are still functions of the full passenger arrival $\xi^{w-1}$ but all $\xi^i$, $\forall i \leq w$ since $\sum_{i \leq w} (\pi^i_{\kappa\ell} C_i)$. Additionally, $u_{\rho}$ depends on $\zeta^{w}$ for all $w$, which is equivalent to knowing the actual passenger arrival $\xi$. Since restricting $a^w$ and $u_{\rho}$ to be functions of $\zeta^{w-1}$ would result in further loss of optimality we avoid it here.

**Proposition 3.** Problem ( $\mathcal{P}$) provides a lower bound on the optimal objective value of problem ( $\mathcal{P}$).

3.2 Robust Counterpart

Problem ( $\mathcal{P}$) exhibits only a moderate number of decision variables but still a very large number of constraints. In what follows, we propose to mitigate the number of constraints by using techniques inspired from modern robust optimization [Ben-Tal et al., 2004]. The key observation is that under the linear decision rule approximation, all constraints in the problem (except from (4)) are linear in $\xi$, thus being expressible in the form $a(x)^T \xi \leq 0 \forall \xi \in \Xi$, for some linear function $a$ that maps the collection of all decision rule coefficients (denoted by $\kappa$) to coefficients of $\xi$. The following proposition enables us to reformulate these constraints in a compact fashion.

**Proposition 4.** For any $y \in \mathbb{R}^k$, define:

i) $y^T \xi \leq 0 \forall \xi \in \Xi$

ii) $\exists \lambda \in \mathbb{R}^d$ with $\lambda \geq 0$, $V^T \lambda \geq y$, and $h^T \lambda \leq 0$.

Then ii) implies i).

Applying the above result to each constraint in Problem ( $\mathcal{P}$)(except constraint (4)), we are able to represent

\[^2\text{http://teamcore.usc.edu/papers/2017/smc17 Appendix.pdf}\]
each of these constraints efficiently. We denote the resulting problem by \((P_{1-rc})\). For the airport security setting where \(\Xi\) is defined as set (3), then the multi-stage robust optimization problem \((P)\) is efficiently solvable.

**Proposition 5.** Suppose that the uncertainty set is defined as in (3). Then, statements i) and ii) in Proposition 4 are equivalent. Moreover, Problem \((P)\) (equivalently \((P_{1-rc})\)) is equivalent to a linear programming problem whose size is polynomial in the number of time windows, categories, resources, and teams.

### 3.3 Constraint Randomization

Although we were able to obtain an exact tractable reformulation of Problem \((P)\) under the uncertainty set from Example 1, this is not the case for general uncertainty sets. Indeed, for general \(\Xi\), Problem \((P_{1-rc})\) still involves constraint (4) enforced over a set \(\Xi\) of potentially very large cardinality. We obtain a tractable approximation to \((P_{1-rc})\) by replacing \(\Xi\) with subsets \(\Xi^N \subset \Xi\) of cardinality \(N\). We denote the resulting problem by \((P^{N})\). The following theorem shows that a randomly sampled subset \(\Xi^N\) of moderate cardinality \(N\) will lead a good approximation.

**Theorem 1 ([Campi and Garatti, 2008]).** Suppose that \((\mathcal{P}_{1-rc})\) is feasible and accommodates \(n\) decision variables. For a prespecified violation probability \(\epsilon \in (0,1)\) and confidence \(\beta \in (0,1)\), define

\[
N(\epsilon, \beta) := \min \left\{ N \in \mathbb{N} : \sum_{i=0}^{n-1} \binom{i}{N} \epsilon^i (1 - \epsilon)^{N-i} \leq \beta \right\}
\]

Then, the probability mass of all \(\xi \in \Xi\) whose associated constraints are violated by an optimal solution of \((\mathcal{P}_{1-rc})\), for \(N \geq N(\epsilon, \beta)\), does not exceed \(\epsilon\) with confidence \(1 - \beta\).

The parameter \(\epsilon\) describes the probability that an optimal solution to \((\mathcal{P}_{1-rc})\) violates the overflow constraint. A violation of the overflow constraint implies that the overflows are calculated incorrectly for some samples so that the part of the objective associated with overflow is calculated incorrectly. The theorem states that such miscalculations are rare. Moreover, the size of the resulting sampled problem is polynomial in the number of time windows, categories, resources, and teams, see [Vayanos et al., 2012]. Since the number of samples required is often still large, in order to solve the resulting problem more efficiently we employ a cutting plane method, in the spirit of [Fischetti and Monaci, 2012].

### 4 Evaluation

We evaluate our framework on airport passenger screening problems with uncertainty set \(\Xi_{AS}\).

![Fig 1: Utility improvement over averaged sample and random uniform in (a) worst case and (b) average case.](image)

#### 4.1 Solution Quality

The optimal objective of our solution gives us the performance on the training set of samples we use. We evaluate the solution quality out of sample (both on average and in the worst case) by generating a large test set. We also use the test set to compute an experimental violation probability. We assume that the arrival of passengers is normally distributed in the range \(\Delta_\kappa\). Each data point is averaged over 30 trials, each with randomized parameter settings, with error bars giving the 90% confidence intervals. For each of these trials we generate 10,000 samples from the distribution of passenger arrivals, and evaluate the computed strategy on each sample so that each data point corresponds to 300,000 evaluations.

**Uncertainty model vs Averaged Model.** We compare our solution method to the TSG model for problems with increasing numbers of flights with \(W = 10\). The TSG model optimizes against only the average \(\xi\), so there will be many scenarios where the strategy becomes infeasible. We consider two heuristics to adjust an infeasible strategy: (1) Overflow Heuristic: add excess passengers to the existing overflow queue, or (2) Open-Team Heuristic: send excess passengers to any team with available capacity. Figure 1 summarizes our results. Against both heuristics, we outperform the TSG in worst case (average) by more than 100% (50%). The average violation probability was 98 ± 2% for the averaged sample solutions and 0.5 ± 0.02% for the solution to \((\mathcal{P}_{1-rc})\).

**Uncertainty Model vs Uniform Random.** We compare to a baseline where passengers are assigned to teams uniformly at random. Figure 1 shows our results. In both the average and worst cases, the solution quality of random screening can be arbitrarily bad– we reach around 200% improvement.

**Full Stochastic Program.** We also compare the quality of the solution of \((\mathcal{P}_{N-rc})\) to that of the optimal solution to the full stochastic program associated with \((\mathcal{P})\). Because the full program is exponential in the number of categories, we can only solve for very small problem instances. We fix the number of time windows, with an arrival period of 2 time windows for any flight, and show runtime and solution quality for a small range of categories. The results are shown in Table 2 where near-optimal performance is exhibited.
Fig. 2: Solve & Wall time with increasing number of flights.

Table 1: Experimental violation probability with increasing problem size.

<table>
<thead>
<tr>
<th>(φ, ρ)</th>
<th>% Diff</th>
<th>Solve Time (ms)</th>
<th>Wall Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1.2)</td>
<td>1.3(0.1)</td>
<td>7.4(1.0)</td>
<td>13.1(0.4)</td>
</tr>
<tr>
<td>(1.3)</td>
<td>0.29(0.1)</td>
<td>40(1)</td>
<td>320(10)</td>
</tr>
<tr>
<td>(1.4)</td>
<td>-</td>
<td>110(40)</td>
<td>-</td>
</tr>
<tr>
<td>(2.1)</td>
<td>1.1(0.03)</td>
<td>2.5(0.07)</td>
<td>7(0.2)</td>
</tr>
<tr>
<td>(2.2)</td>
<td>0.8(0.01)</td>
<td>87(1)</td>
<td>213(90)</td>
</tr>
<tr>
<td>(2.3)</td>
<td>-</td>
<td>340(50)</td>
<td>2700(1000)</td>
</tr>
</tbody>
</table>

Table 2: Comparing the ($\mathcal{P}^{V}_{\text{ref}}$) to full stochastic program ($\mathcal{P}$). Blank entries correspond to instances where the full stochastic program could not be held in memory.

4.2 Scalability

Figure 2 shows total solve and wall times for problems with increasing number of flight categories. We are able to efficiently solve for a very large number of flight categories, with polynomial growth with respect to flight categories. Table 1 summarizes our findings. We see that even for very large problems, where the cardinality of $\mathcal{E}_{AS}$ is very large, the computed strategies have very low violation probability.

Deployment to Saturation Ratio. In Figure 3 we explore the space in which the decision problem becomes difficult by comparing the linear decision rule to a constant decision rule, where we make the same decisions regardless of the past arrival of passengers. It is a known phenomenon in security games, that the problem difficulty increases as the deployment to saturation ratio (ratio of defender resources to targets) approaches 0.5 [Jain et al., 2012].

We measure the ratio by comparing the number of passengers to the capacity, for a single flight, so that the maximum number of passengers which can be screened in any time window is clearly defined. Figure 3 shows that as the problem difficulty increases, the gap in solution quality becomes large and the adaptive screening greatly outperforms the constant strategy.

5 Conclusion and Future Work

We address a significant limitation in TSG, where the previous unrealistic assumption of complete certainty renders its solution unusable in real-world settings. We provide a novel framework which is scalable, provides good solutions quality and works for generalized models of uncertainty.

Acknowledgements

This research was supported by MURI Grant W911NF-11-1-0332.

References


Handling Continuous Space Security Games with Neural Networks

Nitin Kamra1∗, Fei Fang2†, Debarun Kar1, Yan Liu1, Milind Tambe1
University of Southern California1, Harvard University2
{nkamra, dkar, yanliu.cs, tambe}@usc.edu1, fangf07@seas.harvard.edu2

Abstract
Despite significant research in Security Games, limited efforts have been made to handle game domains with continuous space. Addressing such limitations, in this paper we propose: (i) a continuous space security game model that considers infinite-size action spaces for players; (ii) OptGradFP, a novel and general algorithm that searches for the optimal defender strategy in a parametrized search space; (iii) OptGradFP-NN, a convolutional neural network based implementation of OptGradFP for continuous space security games; (iv) experiments and analysis with OptGradFP-NN. This is the first time that neural networks have been used for security games, and it shows the promise of applying deep learning to complex security games which previous approaches fail to handle.

1 Introduction
Stackelberg Security Games (SSGs) have been extensively used to model defender-adversary interaction in protecting important infrastructure targets such as airports, ports, and flights [Tambe, 2011]. In SSGs, the defender (referred to as “she”) perpetually defends a set of targets with a limited number of resources, and the adversary (referred to as “he”) is able to surveil and learn the defender’s strategy and then plan an attack based on this information. Exact and approximate approaches have been proposed to find the optimal defender strategy in SSGs, which maximizes her expected utility given that the attacker will best respond to the defender’s strategy [Kiekintveld et al., 2009; Amin et al., 2016].

Recently, there has been an increasing interest in SSGs for green security domains such as protecting wildlife [Yang et al., 2014; Kar et al., 2015; Fang et al., 2015], fisheries [Haskell et al., 2014] and forests [Johnson et al., 2012]. Unlike infrastructure protection domains which have discrete locations, green security domains are categorized by continuous spaces (e.g., a whole conservation area needs protection).

Most previous works discretize the area into grid cells and restrict the players’ actions to discrete sets [Yang et al., 2014; Haskell et al., 2014]. However, a coarse discretization may lead to low solution quality, and a fine-grained discretization would make it intractable to compute the optimal defender strategy, especially when there are multiple defender resources. While [Johnson et al., 2012] addresses continuous space in security games in the domain of forest protection, it focuses on a special case: the objective of the defender is to compute a strategy to minimize the trespassing distance of the adversaries who are located at the boundary of the forest area and therefore the problem can be reduced to a one-dimensional problem.

Our major contributions are as follows:

- a continuous space security game model which considers infinite action spaces over two-dimensional continuous areas with asymmetric target distribution.
- OptGradFP, a novel and general algorithm which leverages recent advances in policy learning and game theoretic fictitious play to optimize parametrized policies in continuous spaces.
- OptGradFP-NN, an application of OptGradFP using convolutional neural networks (CNNs) to represent the players’ mixed strategies.

Finally, we conduct experiments and analysis with OptGradFP-NN and demonstrate the superiority of our approach against comparable approaches such as StackGrad [Amin et al., 2016] and Cournot Adjustment (CA) [Fudenberg and Levine, 1998]. Our analysis shows that standard approaches either (i) take very aggressive steps for both players and never settle to a good strategy (CA), or (ii) makes aggressive best response updates for the opponent and only soft steps for the defender, thus leading to poor average defender rewards (StackGrad). In contrast, our approach effectively allows both players to take soft steps and takes past strategies into account through a replay memory, thereby eventually converging to a good average response for both players. We show for the first time, the promise of applying deep learning to complex continuous domain security games which previous approaches fail to handle.
2 Preliminaries and Related Work

We use small letters (x) to denote scalars, bold small letters (x) to denote vectors, capitals (X) to denote matrices and bold capitals (X) to denote random vectors. R represents the set of real numbers. 0_n and 1_n are vectors of size n of zeros and ones respectively (we will sometimes skip n if the size is evident from context). I_n is the identity matrix of size n×n. Saying x ∈ [a, b] implies that all corresponding elements of x are ≥ those of a and ≤ those of b. The notation [n] is the set of natural numbers up to n i.e. {1, 2, . . . , n}. N(µ, ν^2) is the normal distribution with mean µ and variance ν^2.

The sigmoid function 1/(1+exp(−z)) is denoted by σ(z). The logit function is defined as: logit(x) = log(x/(1−x)) ∀x ∈ [0, 1]. Note that the sigmoid and logit functions are inverses of each other i.e. σ(logit(x)) = x.

2.1 Stackelberg Security Games

A Stackelberg Security Game (SSG) [Kiekintveld et al., 2009; Korzhyk et al., 2011] is a leader-follower game between a defender and an adversary (a.k.a. opponent). An action or a pure strategy of the defender is to allocate the resources to protect a subset of targets in a feasible way (e.g., assign each resource to protect one target). A pure strategy of the adversary is to attack a target. The mixed strategy of a player is a probability distribution over the pure strategies. We use the term mixed strategy and policy interchangeably in the rest of the paper.

The payoff for a player is decided by the joint action of both players, and the expected utility function is defined as the expected payoff over all possible joint actions given the players’ (mixed) strategies. In this paper, we restrict ourselves to zero-sum games while deferring investigation of general-sum games to future work.

An attacker best responds to a defender strategy if he chooses a strategy that maximizes his expected utility. The optimal defender strategy in SSGs is the (mixed) strategy that maximizes her expected utility, given that the attacker best responds to it and breaks ties in favor of the defender. In zero-sum SSGs, the optimal defender strategy is the same as the strategy for the defender in any Nash equilibrium (NE). In most previous work on SSGs, discrete actions for the players are considered, even if the game setting is over continuous space [Amin et al., 2016; Gan et al., 2017]. There are a few exceptions [Johnson et al., 2012; Fang et al., 2013; Yin et al., 2014], but the solution techniques often rely on exploitable spatio-temporal structures of the problem and cannot be generalized to handle continuous spaces as has been handled in this paper.

2.2 Fictitious play in normal form games

Fictitious play (FP) is a learning rule where each player plays best response to the empirical frequency of their opponent’s play. It converges to a NE under various settings including two-player zero-sum games [Fudenberg and Levine, 1998].

2.3 Policy Gradient Theorem

According to the policy gradient theorem [Sutton et al., 1999], given a function f(·) and a random variable X ∼ p(x|θ), the gradient of the expected value of f(·) with respect to a policy parameters can be computed as

\[ \nabla_\theta \mathbb{E}_X[f(X)\log p(X|\theta)] \]

(1)

We can approximate the gradient on the right-hand side by sampling x_1 ∼ p(x|θ), and computing \( \nabla_\theta \mathbb{E}_X[f(X)|\theta] \approx f(x_1)\nabla_\theta \log p(x_1|\theta) \). The only requirement for this to work is that the density p(x|θ) should be computable and differentiable w.r.t. θ for all x. We will use the policy gradient theorem to compute the gradients of the defender and opponent utilities w.r.t. their policy parameters in our algorithm.

2.4 Logit-normal distribution

Logit-normal is a continuous distribution with a bounded support. A vector random variable X ∈ [0, 1] is said to be distributed according to a logit-normal distribution if logit(X) is distributed according to a normal distribution. The density function is given by:

\[ p_\nu(x; \mu, \nu) = \frac{1}{\sqrt{2\pi\nu}} \frac{1}{x(1-x)} e^{-\frac{(\logit(x) - \mu)^2}{2\nu}} \]

(2)

Unlike the normal distribution, logit-normal distribution does not have analytical expressions for its mean and standard deviation. But we can still parametrize the distribution by using the mean (µ) and standard deviation (ν) of the underlying normal distribution. If X ∼ p_ν(X; µ, ν), a sample of X can be drawn by sampling e ∼ N(0, 1) and then outputting x = σ(νe + µ).

3 Continuous Space Security Game

We begin by describing the forest protection game model. Though we consider it as an example domain, the game model is general and can also represent other domains such as wildlife and fishery protection.

Game model: We assume a circular forest with radius 1.0 (i.e. all lengths are normalized with respect to the forest radius), with a prespecified but arbitrary tree distribution. All locations are represented in cylindrical coordinates with the forest center as origin. We assume n lumberjacks who can collaborate to plan their wood chopping locations. We will use the word adversary or opponent interchangeably to refer to the lumberjacks as a group. The defender has m forest guards, which can be allocated to various forest locations to ambush the trespassing lumberjacks.

State representation: One way of specifying the game state (S) is via locations of all trees. This leads to a variable state-size, dependent on the number of trees in the forest. Since a variable length representation is hard to process and we are mostly concerned with the relative density of trees over the forest, we instead summarize the forest state S as a 120 × 120 matrix containing a grayscale image of the forest. This makes the input representation for the defender and opponent policies invariant to the number of trees in the forest and also allows our approach to be used for planning strategies by using satellite images of a forest as input. An example input in RGB is shown in figure 1a (input to the players is a grayscale version).

Defender action: The defender picks m locations, one for each guard to remain hidden, and ambush lumberjacks
with wood. The defender’s action \( a_D \) is a set of \( m \) distances \( d \in [0,1]^m \) and angles \( \theta \in [0,2\pi]^m \) i.e. \( a_D \in \mathbb{R}^{m \times 2} \). The cylindrical coordinates \( (d, \theta) \) specify the guards’ positions in the forest.

**Adversary action:** Following [Johnson et al., 2012], we assume that lumberjacks trespass the forest boundary and move straight towards the forest center. They can stop at any point along this straight line, cut wood in a radius \( R_l \) around the stopping point and come back to their starting location. The model is justified because lumberjacks generally wish to avoid one another as much as possible [Johnson et al., 2012]. Since the lumberjack trajectories are fully specified by their stopping coordinates, the adversary’s action is to decide all stopping points. The opponent’s action \( a_O \) is a set of \( n \) distances \( r \in [0,1]^n \) and angles \( \phi \in [0,2\pi]^n \) i.e. \( a_O \in \mathbb{R}^{n \times 2} \). The cylindrical coordinates \( (r, \phi) \) define the wood chopping locations of all lumberjacks.

**Rewards:** A lumberjack is considered ambushed if his path comes within \( R_g \) distance from any guard’s location. If a lumberjack gets ambushed, he fails in cutting the trees and gets a penalty \(-r_{pen}\). The total utility for the opponent \( (r_O \in \mathbb{R}) \) equals to the total number of trees cut by the lumberjacks. The total utility for the defender is \( r_D = -r_O \).

**Game play:** Given prespecified tree locations, a single run of the game proceeds as follows: (1) The defender gives \( m \) guard locations and the adversary gives \( n \) wood chopping locations, (2) The game simulator computes and returns rewards for the defender and opponent. By playing the game multiple times, the defender gets rewards and uses this information to optimize her strategy. A full game has been visualized in figure 1b.

### 4 Policies and Utilities

**Policies:** A player’s policy (or mixed strategy) is a probability distribution over the player’s actions given the game state \( (S) \). The defender maintains a learnable policy \( \pi_D \) parametrized by weights \( w_D \), from which she can sample the guards’ positions. She also maintains an estimate of the adversary’s policy \( \pi_O \) parametrized by \( w_O \), which helps her learn her own optimal policy. Note that the opponent’s real policy is the best response to the defender’s deployed policy (not the same as \( \pi_O \)). We use the symbols \( \pi_D, \pi_O \) to denote the policies, and expressions \( \pi_D(a_D | S; w_D), \pi_O(a_O | S; w_O) \) to denote the probability of a certain action \( (a_D \text{ or } a_O) \) drawn from the policy \( (\pi_D \text{ or } \pi_O) \).

**Utilities:** The utilities of the defender and the opponent \( (J_D \text{ and } J_O = -J_D \text{ respectively}) \) are the expected rewards obtained:

\[
J_D(w_D, w_O) = \mathbb{E}_{S,a_D,a_O}[r_D(S,a_D,a_O)] = \int_S \int_{a_D} \int_{a_O} P(S) \pi_D(a_D | S; w_D) \pi_O(a_O | S; w_O) r_D(S,a_D,a_O) \, dS \, da_D \, da_O \tag{3}
\]

Note that the integral over \( S \) can be removed if we only require strategies over a specific state (forest), but our method allows learning policies over multiple states if needed.

Both the defender and the opponent want to maximize their utility functions. Yet, their approaches differ since the defender has to deploy her policy first, without knowing the opponent’s policy. The opponent gets to observe the defender’s policy and can use this information to react with a best response to the defender’s deployed policy. Hence the problem faced by the defender is essentially as follows:

\[
w_D^* = \arg \max_{w_D} \min_{w_O} J_D(w_D, w_O) \tag{4}
\]

The opponent’s problem is simpler:

\[
w_O^* = \arg \min_{w_O} J_O(w_D, w_O) \tag{5}
\]

We approach these problems by taking a gradient optimization based approach. The gradient of \( J_D \) w.r.t. the defender parameters \( w_D \) can be found using the policy gradient theorem (see section 2.3) as:

\[
\nabla_{w_D} J_D = \mathbb{E}_{S,a_D,a_O} [\nabla_{w_D} \pi_D(a_D | S; w_D) r_D] \tag{6}
\]

The exact computation of the above integral is prohibitive, but it can be approximated from a batch of \( B \) on-policy samples \( (w.r.t. \pi_D) \) using the following unbiased estimator:

\[
\nabla_{w_D} J_D \approx \frac{1}{B} \sum_{i=1}^{B} \nabla_{w_D} \pi_D(a_D^i | S; w_D) r_D^i \tag{7}
\]

The gradient for the opponent objective \( (w.r.t. \pi_O) \) can be similarly estimated as:

\[
\nabla_{w_O} J_O \approx \frac{1}{B} \sum_{i=1}^{B} \nabla_{w_O} \pi_O(a_O^i | S; w_O) r_O^i \tag{8}
\]

Ideally one can use even a single sample to get an unbiased estimate of the gradients, but such an estimate has a very high variance. Hence, we use a small batch of \( i.i.d. \) samples to compute the gradient estimate.

### 5 OptGradFP: Optimization with policy gradients and fictitious play

We propose our algorithm OptGradFP to solve security game models. Our algorithm leverages the recent advances in
Algorithm 1: OptGradFP

Data: Learning rates \((\alpha_D, \alpha_O)\), decays \((\beta_D, \beta_O)\), training rates \((f_D, f_O)\)

Result: Parameters \(w_D\) and \(w_O\)

Initialize policy parameters \(w_D\) and \(w_O\) randomly;
Fill replay memories \(mem_D\), \(mem_O\) of size \(E\) with randomly played games;

for \(ep \in \{0, \ldots, ep_{max}\}\) do

Get game state \(S\);
Sample \(a_D = (d, \theta) \sim \pi_D(|S; w_D|)\), \(a_O = (\rho, \phi) \sim \pi_O(|S; w_O|)\);
Execute actions \((a_D, a_O)\) and get rewards \((r_D, r_O)\);
Store sample \(\{S, a_D, a_O, r_D, r_O\}\) in \(mem_D\), \(mem_O\);
if \(ep \% f_D == 0\) then

Get samples \(\{S^i, a_D^i, a_O^i, r_D^i, r_O^i\}_{i \in [E]}\) from \(mem_D\);
Replay all \(E\) games \(S^i, a_D^i, a_O^i\) with \(a_D^i \sim \pi_D(|S^i; w_D|)\) to obtain rewards \(r_D^i, r_O^i\);
\(\nabla_{w_D} J_D = \frac{1}{E} \sum_{i=1}^{E} \nabla_{w_D} \pi_D(a_D^i | S^i; w_D) \cdot r_D^i;\)
\(w_D := w_D + \frac{\alpha_D}{1 + \epsilon_D} \nabla_{w_D} J_D;\)

end if

if \(ep \% f_O == 0\) then

Get samples \(\{S^i, a_D^i, a_O^i, r_D^i, r_O^i\}_{i \in [E]}\) from \(mem_O\);
Replay all \(E\) games \(S^i, a_D^i, a_O^i\) with \(a_O^i \sim \pi_O(|S^i; w_D|)\) to obtain rewards \(r_D^i, r_O^i\);
\(\nabla_{w_O} J_O = \frac{1}{E} \sum_{i=1}^{E} \nabla_{w_O} \pi_O(a_O^i | S^i; w_O) \cdot r_O^i;\)
\(w_O := w_O + \frac{\alpha_O}{1 + \epsilon_O} \nabla_{w_O} J_O;\)

end if

Policy gradient learning [Sutton et al., 1999] and those from game theoretic fictitious play [Heinrich et al., 2015; Heinrich and Silver, 2016], to find the optimal defender parameters \(w_D\) which maximize her utility.

The pseudocode for OptGradFP has been provided in algorithm 1. The algorithm computes the optimal policy from the defender side. It essentially gets a game state \(S\) and it generates responses from the player’s current policy estimates \(\pi_D, \pi_O\) to receive rewards \(r_D, r_O\). It keeps storing all these game samples in separate replay memories \(mem_D\) and \(mem_O\) for the defender and the opponent. The replay memories are circular buffers of length \(E\) and once full, they overwrite the least recently stored sample in order to accommodate new incoming samples.

Every few episodes \((f_D\) for defender, \(f_O\) for opponent), the algorithm retrieves all samples stored in the replay memory for a player, replays all the games by resampling that player’s actions for those samples from his/her current policy (while keeping the other player’s actions the same), and improves the player’s policy using the policy gradient update.

Note that the policy gradient update is essentially a soft update towards the best response by changing the player’s policy parameters \((w_D\) or \(w_O\)) in a way that increases their expected reward. We employ learning rate decay to take larger steps initially and obtain a finer convergence towards the end. Yet, it is not a soft update towards a best response to the other player’s current policy, but rather towards a best response to an average of the other player’s current and previous policies. This is because we compute the policy gradient for a player using all \(E\) samples in the player’s replay memory, out of which only \(f_D\) (or \(f_O\)) samples are drawn from the other player’s current policy and the rest are from the other player’s previous policies \((E \gg f_D, f_O)\). But all \(E\) samples employ the player’s current policy, since the player replays all games with his/her current policy before using them to make the policy gradient update. This is an approximation to fictitious play where both players react with a best response to the other player’s average strategy.

Note that replaying all games with the player’s current policy before the policy gradient step is required since policy gradients require on-policy sampling. If a game simulator, which allows playing games by restoring arbitrary previous states is not available, importance sampling can be a viable substitute for this step.

6 OptGradFP-NN: OptGradFP with neural networks

For our OptGradFP implementation, we assume each element of the defender’s and opponent’s actions \((a_D, a_O)\) to be distributed independently according to logit-normal distributions. Our choice of logit-normal distribution meets the requirement of a continuous distribution, differentiable w.r.t. its parameters and having bounded support (since our action spaces are bounded and continuous).

To represent these distributions we need to generate the means and standard deviations of the underlying normal distributions for each element of \(a_D = (d, \theta)\) and \(a_O = (\rho, \phi)\). Though any function approximators can be used for this purpose, we use two convolutional neural networks to generate the means and standard deviations for each player, owing to their recent success in image processing and computer vision applications [Krizhevsky et al., 2012; Zeiler and Fergus, 2014].

Defender policy representation: The defender neural network parameterized by weights \(w_D\) takes as input an image \(S\) of the forest tree locations and outputs means \((\mu_d(S; w_D) \in \mathbb{R}^m, \mu_o(S; w_D) \in \mathbb{R}^m)\) and standard deviations \((\nu_d(S; w_D) \in \mathbb{R}^m, \nu_o(S; w_D) \in \mathbb{R}^m)\) for two \(m\)-dimensional gaussians. For clarity we will skip writing \((S; w_D)\) with these parameters. Each defender action coordinate is then a logit-normal distribution and the joint probability of taking action \(a_D = (d, \theta)\) is given by:

\[
\pi_D(d, \theta | S; w_D) = \prod_{i \in [m]} p_{\theta}(d_i; \mu_{d,i}, \nu_{d,i})
\]

\[
p_{\theta}(\frac{\theta_i}{2\pi}, \mu_{d,i}, \nu_{d,i})
\]

where the product is over all \(m\) elements of the vector.

The defender neural network takes an image of size \(120 \times 120\) as input. First hidden layer is a convolutional layer with 32 filters of size \(16 \times 16\) and strides \(8 \times 8\). The second hidden layer is convolutional with 16 filters of size \(4 \times 4\) and strides \(2 \times 2\). Both convolutional layers have \text{relu}\ activations and
no pooling. Next layer is a fully-connected dense layer with 32$m$ units (where $m$ = number of guards) and tanh activation. Lastly we have four parallel fully-connected dense output layers one each for $\mu_d, \nu_d, \mu_g$ and $\nu_g$. These four layers have $m$ units each, with the layers for means having linear activations and those for standard deviations having relu activations. We add a fixed small bias of 0.1 to the outputs of the standard deviation layers to avoid highly concentrated or close to singular distributions. We also clip all gradients to stay in the range $[-0.5, 0.5]$ to avoid large weight updates and potential divergence [Mnih et al., 2015].

**Opponent policy representation:** The opponent neural network similarly parametrized by weights $\omega_O$ takes as input $S$ as input and outputs means $(\mu_o(S; \omega_O) \in \mathbb{R}^m, \nu_o(S; \omega_O) \in \mathbb{R}^m)$ and standard deviations $(\nu_o(S; \omega_O) \in \mathbb{R}^m, \nu_o(S; \omega_O) \in \mathbb{R}^m)$ for two $n$-dimensional gaussians. For clarity we will skip writing $(S; \omega_O)$ with these parameters. Each opponent action coordinate is then a logit-normal distribution and the joint probability of taking action $a_O = (p, \phi)$ is:

$$
\pi_O(p, \phi|S; \omega_O) = \prod_{i \in [n]} p_n(p_i|\mu_{\phi,i}, \nu_{\phi,i})
$$

where the product is over all $n$ elements of the vector.

The opponent neural network is similar to the defender network, with the only difference being the number of hidden units in the fully-connected dense layers. The fully-connected hidden layer has 32$n$ units (where $n$ = number of lumberjacks) and the four output layers for $\mu_o, \nu_o, \mu_\phi$ and $\nu_\phi$ have $m$ units each.

Finally, note that even though we assumed all elements of $a_D$ (resp. $a_O$) to be independent logit-normal distributions, the means and standard deviations for the underlying normal distributions are computed jointly via the convolutional neural networks and are dependent on each other. This allows the defender and the opponent to plan coordinated moves.

**Hyperparameters:** Our OptGradFP implementation uses a replay memory size of $E = 1000$ samples, maximum episodes $e_{\text{max}} = 10000$, learning rates $\alpha_D = \alpha_O = 10^{-3}$, training rates $f_D = f_O = 50$ and decays $\beta_D = \beta_O = 0.045$. The architectures of all neural networks involved and all algorithm hyperparameters were chosen by doing informal grid searches within appropriate intervals. For more information on choosing convolutional neural network architectures, refer to [Ullah and Petrosino, 2016].

## 7 Experiments and Results

We now present experiments against several baselines. Cournot Adjustment (CA), one of the early techniques used to optimize players’ policies, makes the defender and the opponent sequentially respond to each other’s policy with their best response policies. This method can converge to the nash equilibrium for certain classes of games [Fudenberg and Levine, 1998]. Another method called StackGrad was recently proposed [Amin et al., 2016]. It uses a best response computation for the opponent’s updates, and a policy gradient update similar to ours for the defender (but no fictitious play). We compare our results against those from CA and also with a version of StackGrad in our experiments.

Note that StackGrad uses a best response computation for the opponent in the original paper (approximated by a parametrized softmax distribution). Since it is hard to compute the best response to any policy analytically for our forest domain, we use the following approximation to emulate the opponent’s best response: we play multiple games with random actions for the opponent while drawing the defender’s actions from its current policy. The random action which gets the highest reward against the defender’s policy is chosen as the best response action for the opponent.

We present our results for $m = 8$ guards and $n = 8$ lumberjacks where the numbers have been chosen to provide appropriate coverage of the forest (since fewer guards leave too much open space). We set the ambush penalty $r_{\text{pen}} = 10$, guard radius $R_g = 0.1$ and lumberjack radius $R_l = 0.04 < R_g$ (since guards can scout lumberjacks from long distances). These are just representative values and our algorithm works well for any values of these parameters.

### 7.1 Reward curves

Figure 2 shows a plot of the average reward achieved by the defender on the $E$ replayed games before every training iteration. In OptGradFP, these average rewards measure the utility of the defender’s current policy against the opponent’s average policy. For CA and StackGrad, the average rewards measure the defender’s utility against the opponent’s current policy (no stored history for CA and StackGrad). We observe that StackGrad starts with a random value of average reward for the defender (due to random initialization) and thereafter goes down. The curve mostly stays around the same average reward while displaying consistent oscillations. CA on the other hand, jumps up to a higher average reward value than random and oscillates around it. Finally, OptGradFP smoothly rises and saturates at an average reward value higher than all other baselines.

### 7.2 Opponent’s final utility

Another indicator of performance is the utility achieved by opponent’s final policy after the defender has fixed her own policy. The opponent’s maximum utility was computed approximately (computing the actual value is extremely prohibitive), by sampling 100 random opponent actions and 100 actions from the defender’s final policy. 10000 games were played with each combination of the defender’s and opponent’s actions and the opponent action which led to the maximum reward for the opponent (averaged over all 100 defender actions) was assumed to be the opponent’s final action.

Table 1 gives the opponent’s maximum utility after $e_{\text{max}}$ episodes for each algorithm. OptGradFP clearly gives the least utility to the opponent as opposed to other baselines. Further, StackGrad dominates CA which suggests that though CA quickly finds a best response to the opponent’s estimated current strategy, it’s policy does not provide appropriate coverage of the forest region. After the policy has been deployed, the opponent can still find high utility places to attack.
Table 1: Maximum average utility of the opponent.

<table>
<thead>
<tr>
<th></th>
<th>CA</th>
<th>StackGrad</th>
<th>OptGradFP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Util</td>
<td>567.05</td>
<td>518.34</td>
<td>499.15</td>
</tr>
</tbody>
</table>

7.3 Learned defender policy

We show a visualization of the defender’s final policy for each algorithm in figure 3. Though [Johnson et al., 2012] proposed circular bands as the optimal patrol policy for a uniform tree density, the same still holds for radially symmetric tree densities. Our tree distribution is close to being radially symmetric, and hence the optimal defender policy should be in the form of circular bands centered at the origin. Figure 3c shows the OptGradFP defender policy, and as expected it contains mostly circular bands centered around the origin. The other algorithms find local regions to guard, and leave the lumberjacks lots of space to chop wood without getting ambushed. This is in agreement with the maximum average utility values for the opponent in table 1. One important point to note is the placement of the circular bands. It is easy to see that placing the bands too close to the forest center would leave a huge area to be chopped by the lumberjacks. Also, placing the guards at the boundary would make them very sparsely distributed and most lumberjacks would be able to come and go unambushed. Our algorithm finds a reasonable middle ground by inferring good radii to place the guards.

7.4 Replay memory

We also explored effects of not using fictitious play in OptGradFP. To do this, we use a small replay memory of size $E = f_D = f_O$, only containing games sampled from current policies of both players. This is equivalent to making only policy gradient updates for both players.

The utility achieved by opponent’s best response policy was 555.58, which is only slightly better than CA and worse than both StackGrad and OptGradFP. The resulting defender distribution is shown in figure 3d. The defender policy is not well spread out anymore, since the method does not have memory of opponent’s previous steps. The results resemble those of CA, since policy gradient update is a small step towards the best response (like in CA). Hence we can approximate CA with OptGradFP using $E = f_D = f_O$.

In general, keeping a large replay memory ($E \gg f_D, f_O$) gave us less fluctuation and smoother convergence properties, by allowing the policy gradient update to approximate the best response to the other player’s average policy better. At the same time, a large replay memory led to the replaying time of games becoming a bottleneck for every training step. Hence there exists a trade-off between smooth convergence vs. computation time, which needs tuning to balance the two.

8 Conclusion and Future Work

In this paper, we present for the first time, a neural network based approach to address continuous space security games that previous approaches fail to handle. Our novel approach OptGradFP represents the defender’s strategy by parametrizing it in continuous space and training neural networks using fictitious play and policy gradients to learn the parameters. While we have only trained on a single state, note that our approach OptGradFP-NN is generic enough to train the defender’s policy over multiple distinct game states. Experiments to evaluate this are a promising direction for future work. Generalizing the model to handle arbitrary forest shapes is also a challenge to be addressed in the future.
References


Bitcoin’s Security Model Revisited

Yonatan Sompolinsky and Aviv Zohar
The Hebrew University of Jerusalem, Israel
{yoni_sompo,avivz}@cs.huji.ac.il

Abstract

Cryptocurrencies like Bitcoin provide probabilistic assurances to merchants that payments made to them will not be reversed. We extend the study of payment-reversal (aka double spending) attacks by considering more sophisticated behaviours of attackers. We show that attackers who wish to reverse payments face a decision problem: at each point in time, they must decide whether to continue the attack or to abandon it and launch a new one. Merchants can use our computed optimal attacks to set and adjust their transaction acceptance policies. We analyze and compute optimal attack policies for both single-shot attacks and long term persistent ones. We show that when one considers the decision problem induced by long term attacks, payments can be confirmed faster than previously thought, because an optimal attack involves frequent resets of the attack.

Our analysis utilizes an MDP construction adapted from [Sapirshtein et al., 2015]. Finally, we demonstrate how the attack strategy changes if the merchant does not relay blocks to other Bitcoin nodes, and show that relaying blocks to others strictly improves the security of payments accepted by the node.

1 Introduction

Bitcoin is a novel decentralized system, invented in 2008 by Satoshi Nakamoto [Nakamoto, 2008], which allows users to transfer money to one another by publicly recording payments made with the currency. Transactions are aggregated in batches called blocks, and blocks in turn are sequentially organized in a chain collectively known as the blockchain. Once a transaction has been embedded in a block within the chain, it will be considered a part of the valid history of transactions. The security of a bitcoin transaction depends directly, therefore, on the probability that the block containing it remains part of the blockchain forever. In this work we show that this connection is more complex than commonly described, and we define precisely in what sense bitcoin transactions can be considered secure. We show that attackers face a decision problem, when aiming to reverse as many payments as possible, and we compute optimal attack policies using tools from AI.

The Bitcoin protocol. In order to create a new block, a node (aka miner) is required to solve a difficult cryptographic puzzle. Each new block contains a pointer to its predecessor – usually, the tip of the current chain – effectively extending it with every such additional block. The difficulty of the puzzle regulates the block creation rate, and ensures that the blockchain is extended approximately once every 10 minutes (every 2016 blocks the difficulty of the puzzle is adjusted to keep the growth rate of the chain constant).

In case several chains of blocks form, the Bitcoin protocol dictates that nodes extend the longest chain only (or the one they received first, in case of a tie), and discard and ignore blocks outside this chain. In particular, if an attacker deliberately creates a secret fork and manages to create a longer branch than the current public one, he can publish his branch and thereby replace all blocks in the public chain (after the fork), effectively reversing all payments embedded within them. This scheme is called a double spending attack.

Fortunately, the computational hardness of block creation ensures that a node with less than 50% of the computational power is unlikely to create more blocks than the rest of the nodes, over a long period of time. Consequently, (assuming that blocks propagate much faster than their creation rate), it is highly unlikely that a block buried deep enough in the longest chain would later be removed. Merchants and payment recipients are thus advised to wait for several blocks (aka confirmations) to extend the chain above the block containing their transactions, before considering the payment as finalized.

The classic security analysis. Satoshi in his original work [Nakamoto, 2008], as well as additional works that follow [Rosenfeld, 2014; Sompolinsky and Zohar, 2015], offer several acceptance policies whose security guarantees are given by a theorem of the following “flavour”:

Theorem 1 (informal). As long as the attacker holds less than 50% of the computational power, the probability of a transaction being reversed decreases exponentially with the number of confirmations the block containing it has received.

For instance, a merchant who waits for 4 confirmations before accepting payments is safe against an attacker with 10%
of the computational power, with probability $\approx 1 - 0.00099$.1

**Problems with the classic analysis.** Alas, these analyses apply only when regarding naïve attack strategies, where the attacker tries to create a secret chain only after he broadcasts the transaction to the network. In contrast, consider the following strategy of a 10% attacker against a merchant which regularly waits for 4 confirmations: (i) The attacker continuously tries to create secret extensions to the public chain and to gain a lead of 2 blocks (in Bitcoin, he will usually be successful within 24 hours or less); we term this preparatory stage *pre-mining*. (ii) Once he gains such a lead, the attacker broadcasts the transaction he aims to double spend. The merchant waits for 4 confirmations to appear in the public chain, and then accepts the payment. (iii) Meanwhile, the attacker continues to try and extend his secret chain. (iv) If his chain is longer than the public one at any moment in time after the merchant accepted, he releases his secret chain. This strategy is successful with probability $\approx 0.02728$, i.e., it is 27 times more likely to succeed than the attack considered by others.

It is easy to generalize this strategy and demonstrate that, by controlling the timing of the publication of the victim transaction, the attacker is able to guarantee himself an arbitrarily high probability of success in reversing transactions. Figure 1 illustrates such a pre-mining attack.

Admittedly, increasing the success-probability of the attack comes at the expense of long waiting times before launching it, and of wasting more blocks during the pre-mining stage. However, security guarantees given in the above form (Theorem 1) pretend to provide security even against *irrational* attackers that do not necessarily aim at maximizing their profit. Thus, pre-mining refutes a naïve reading thereof.

**Alternative security model.** Still, these analyses prove that in some sense it is (exponentially) difficult to reverse blocks, and they can be correctly interpreted as follows:

On average, the attacker is able to reverse transactions embedded in only an exponentially small fraction of the blocks created within this period.

By the Strong Law of Large Numbers, there is an immediate connection between the success-probability of a single-shot attack and the fraction of overall successful ones: If an attack carried out in an arbitrary point in time (with no selective timing, as previous analyses assume) succeeds with probability upper bounded by $\epsilon$, then the overall fraction of blocks reversed by a persistent attacker converges to a limit upper bounded by $\epsilon$.

However, by carefully considering the dynamics of a long-run attack, specifically by modeling it as a decision problem, we show that in fact it is possible to produce tighter security guarantees that work in favour of the merchant. Indeed, an attacker that attempts to maximize the fraction of attacked blocks must actively decide when to give up on the attack on a specific block, if the odds are not in his favour, so that he may attack other more recent blocks instead.

Using a Markov Decision Process (MDP), we precisely define the decision problem and the resulting security model (see Section 2 and 3). We compute optimal attack strategies using an adaptation of the MDP-based algorithm from [Sapirshtein et al., 2015].

**Correcting the classic analysis.** Arguably, the fractional “on average” model suggested above fits a merchant that engages frequently in bitcoin transactions, but aims to minimize the number of successful attacks over a long period of time. In contrast, a merchant who uses the blockchain rarely, would probably be focused on defending a payment in a specific block, which corresponds to the security model suggested by the classic analysis.

Unfortunately, as argued above, securing a specific transaction in the blockchain is problematic: The attacker can engage in pre-mining efforts and wait until his lead ensures him a definite success, and only then publish the payment to the victim. Previous analyses which ignored this factor implicitly assume that the attack is carried out in an arbitrary point in time. This assumption might be justified in some scenarios, such as periodic and pre-scheduled payments, or when the attacker is not the entity that initiated the payment.

Still, even under such assumptions, an attack may involve pre-mining, which was not accounted for in previous analyses. We correct this by evaluating the optimal attack analytically. As a separate contribution, we repeat this analysis for the case where the merchant runs a lightweight “Simplified Payment Verification” (SPV) node and does not relay blocks that it receives to others in the network. We show that this provides the attacker with an advantage, and adjust the analysis of the optimal attack strategy in this case.

Our contributions can be summarized as follows:

- We define precisely in what sense bitcoin transactions can be considered secure, taking into account pre-mining and selective timing of transaction publication.
- We use an MDP-based algorithm, adapted from [Sapirshtein et al., 2015], to provide tight upper-bounds on optimal attack strategies. Merchants can use our results and tools to reason about the security of their acceptance policies and adjust them accordingly.
- We correct the classic analysis of Bitcoin’s security by accounting for pre-mining. We do so both for ordinary Bitcoin nodes and for merchants running lightweight clients that do not broadcast blocks they receive. The attacks we analyze are essentially a generalization of the Finney attack [Finney, 2011], and of the (somewhat lesser known) Vector76 [2011] attack.

Our results apply to other blockchain-based systems as well, such as Ethereum [ETH; Wood, 2014].

2 The model

We adopt the original setup analyzed by Satoshi Nakamoto [2008] that has become a standard model of Bitcoin’s operation. The set of all miners creates blocks with
The attacker leads by 2. Transmits 1
Attacker falls behind & resets
2
Publish secret chain
1 has 1 confirmation

Figure 1: The progression of a pre-mining attack on a 1-confirmation merchant. (1) The attacker starts working on a secret chain with \( t_{x_2} \) inside its first block. (2) If the attacker’s chain is shorter than the public one, the attacker gives up and restarts the attack. (3) The attacker manages to gain a lead of 2 blocks. (4) He then transmits the transaction he wishes to double spend, which is included in a block. The transaction gains enough confirmations (here \( k = 1 \)), and the merchant delivers to the attacker the commodity he paid for. (5) The attacker publishes his secret chain and successfully reverses the payment.

Exponential inter-arrival times, with parameter \( \lambda \) (in Bitcoin, \( \lambda = 1/600 \) blocks/second).

Each block contains a reference to a single predecessor block (a cryptographic hash). The entire history of blocks created up to time \( t \) (including blocks not in the longest chain) thus forms a tree, which we denote by \( T_t \). We assume that, in accordance with the Bitcoin protocol, honest participants only keep track of the longest chain they have been given. We further assume that blocks propagate in the network very fast relative to \( 1/\lambda \); under this assumption the honest network’s chain at every point \( t \) in time is uniquely determined, and we denote it by \( C_t \). The length of the chain at time \( t \) is denoted \( \text{height}(C_t) \). The height of a block \( B \), \( \text{height}(B) \), is defined as the number of blocks between it and the first genesis block.

The attacker is assumed to own a fraction \( \alpha \) of the computational power, and the rest \( (1 - \alpha) \) is owned by honest nodes. Thus, the attacker creates blocks at a rate of \( \alpha \cdot \lambda \), and the honest participants at a rate of \( (1 - \alpha) \cdot \lambda \). Following [Eyal and Sirer, 2014], we assume that in case of a tie between the attacker’s chain and the public chain, the attacker’s communication capabilities are such that a fraction \( \gamma \) of the nodes receive his block first and adopt it; for such a race to take place, the attacker must release his matching block immediately after the public chain has extended. A pair \( \alpha, \gamma \) thus characterizes the capabilities of the potential attacker against which the merchant wishes to defend himself. For each such pair, \( \sigma_{\alpha, \gamma} \) denotes the merchant’s acceptance policy, i.e., the (constant) number of confirmations the merchant waits for before accepting the transaction.

e-fractional-robust policies. Given an acceptance policy \( \sigma_{\alpha, \gamma} \), we define the set \( A^t(\sigma_{\alpha, \gamma}) := \{ B \in T_t : \text{height}(C_t) - \text{height}(B) \geq \sigma_{\alpha, \gamma} - 1 \} \); it is the set of blocks which were part of the longest chain at some point in history, had \( \sigma_{\alpha, \gamma} \) confirmations in it (including themselves), but were later removed from it due to a successful attack. For simplicity, we assume that every block contains one payment from the attacker to the merchant.

**Definition 1.** An acceptance policy \( \sigma_{\alpha, \gamma} \) is said to be e-fractional-robust iff for any attacker with parameters \( \alpha \) and \( \gamma \), and under any attack policy:

\[
\lim_{t \to \infty} \frac{|A^t(\sigma_{\alpha, \gamma})|}{|C_t|} < \epsilon
\]  

(1)

The space of possible attack policies will be defined in Section 3. Importantly, note that \( |C_t| \) grows at a constant rate in time: in Bitcoin, the chain is extended approximately once every 10 minutes. Therefore, (1) essentially measures the number of blocks attacked per unit of time, on average. Consequently, in order to bound the overall damage a persistent attacker can cause the merchant on average, the merchant should cap the amount paid to him in each block, and use an e-fractional-robust acceptance policy.

e-arbitrary-robust policies. Let \( tx \) be a transaction that appears in some block \( B \) in the blockchain. It is possible to define the robustness of \( tx \) under the assumption that the attacker was not involved in the choice of \( B \), or, in other words, that the attack was performed in an arbitrary point in time.

**Definition 2.** An acceptance policy \( \sigma_{\alpha, \gamma} \) is said to be e-arbitrary-robust iff for any attacker with parameters \( \alpha \) and \( \gamma \), and under any attack policy:

\[
\Pr(\exists s > t : B \notin C_s \mid \text{height}(C_t) - \text{height}(B) = \sigma_{\alpha, \gamma} - 1) < \epsilon,
\]

where \( B \in C_t \) is a block in the chain whose choice was independent of the attacker’s actions.

3 A persistent attack as a decision problem

We are now ready to formalize the decision problem that the attacker faces, and describe how we compute the opti-
mal policies. The attack is carried out over the entire history of blocks mining. We assume that every block contains one payment from the attacker to the merchant; this is possible, e.g., if the merchant is an online currency exchange. The attacker’s goal is to carry out as many successful double-spending attacks as possible, by overriding the public chain after payments in it were confirmed by the merchant. At each step, he needs to decide whether to extend his secret chain, to publish it, or to abandon it and begin a new one. A successful attack takes place whenever the attacker publishes its secret chain, this chain is longer than (or sometimes, equal to) the public chain, and the public chain is at least $k$ blocks long; indeed, without the latter condition, the merchant hasn’t accepted yet any payment in the public chain, hence overriding it does not harm him.

### 3.1 The underlying Markov Decision Process

We follow [Sapirshtein et al., 2015] and define the attacker’s attack policy as a function that determines his action at every possible state. A state of the MDP is characterized mainly by $l_a$, the length of the attacker’s secret chain, and by $l_h$, the length of the public chain; these lengths are counted from the latest block which both chains agree on. The possible actions are:

- **adopt**– abandoning the current secret chain, so that future chains of the attacker will contain the current tip of the public chain.
- **override**– publishing the current secret chain, in case it is longer than the public one, thereby overriding it.
- **match**– publishing the current secret chain, in case it is the same length as the public one; by our model (following [Eyal and Sirer, 2014]), a fraction $\gamma$ of the honest nodes will adopt the attacker’s chain.
- **wait**– waiting for the next block creation without any additional action.

Every state-transition corresponds to the creation of a new block, either by the attacker (with probability $\alpha$) or by an honest node (with probability $1 - \alpha$). In case of a **match**, the next block will extend the attacker’s chain with probability $\alpha + \gamma \cdot (1 - \alpha)$, because apart from the attacker there is additionally a fraction $\gamma$ of honest nodes that adopts his chain. Since the attacker can only **match** on time if the new block was created by an honest node (otherwise all honest nodes received the honest block first and adopted it), we must encode this information in the state as well. The state is therefore represented by a 3-tuple $(l_a, l_h, fork)$, where **fork** might take the values **rel**, **irrel**, and **active**; **rel** implies that the **match** action is now feasible (i.e., the last block was created by an honest node hence the attacker can publish immediately a matching chain), **irrel** implies that **match** is currently infeasible, and **active** means that the attacker already matched previously and the honest nodes are already split between his chain and the previous honest chain.

To compute an optimal attack correctly, one must normalize by the length of the chain, as in (1), and not by the total number of blocks, because in the long run only the chain growth is kept constant (via adaptation of the block creation rate). We adapt a technique from Sapirshtein et al., who define a reward system with two coordinates, and use a binary-search-based algorithm to compute the optimal policy. We refer the reader to [Sapirshtein et al., 2015], for a more comprehensive description. A succinct representation of the transition and reward matrices appears in Table 1 below.

The main difference from Sapirshtein et al.’s algorithm is that the latter used this technique to compute optimal selfish mining attacks, and to maximize the number of attacker blocks in the chain. In contrast, we reward the attacker differently (as its objective here is different): he is rewarded 1 unit for every block that the merchant accepted – i.e., that had $k$ confirmations – and that was later removed from the chain.

### 3.2 Results

We constructed the MDP described above, for various acceptance policies of the merchant, truncating the state-space to consider chains of length up to 50 blocks. We used the MDP solver from [Chadès et al., 2014], where we utilized the relative value iteration routine to obtain the optimal average reward under an undiscounted average reward scheme. The value of the optimal attack, as computed by the MDP algorithm, defines the degree of fractional-robustness achieved by the corresponding acceptance policy. Table 2 presents the average percentage of blocks that the attacker is able to successfully attack, for several numbers of confirmations. Each cell was computed separately with its own optimal policy. A merchant willing to tolerate a fraction $\epsilon$ of successful double-spend, against an attacker with computational power at most $\alpha$, should choose an acceptance policy such that the corresponding cell in the table is $\epsilon$ or less.

Figure 2 depicts the $\epsilon$-fractional-robustness of the policy $\sigma_{\alpha, \gamma} = 6$, as computed by the algorithm, for different $\gamma$’s. The results of Rosenfeld for the success-probability of an at-

![Figure 2: The fraction of accepted blocks that an optimal attacker can double spend, on average, against the acceptance policy $\sigma \equiv 6$, as a function of the attacker’s hashrate $\alpha$. The different curves correspond to different values of $\gamma$. The probabilistic bound under the classic model (from [Rosenfeld, 2014]) is also plotted for comparison.](image-url)
Table 1: The transition and reward matrices of the MDP. The first coordinate of ‘Rewards’ accumulates the attacker’s rewards, and the second one is used for proper normalization.

<table>
<thead>
<tr>
<th>State (from) × Action</th>
<th>State (to)</th>
<th>Probability</th>
<th>Reward</th>
</tr>
</thead>
<tbody>
<tr>
<td>((l_a, l_b, \cdot)), adopt</td>
<td>((1, 0, 1, rel))</td>
<td>(\alpha)</td>
<td>((0, l_h))</td>
</tr>
<tr>
<td>((l_a, l_b, \cdot)), override(^\dagger)</td>
<td>((l_a - l_h, 0, 1, rel))</td>
<td>(\alpha)</td>
<td>((l_h - k + 1, k))(^\S)</td>
</tr>
<tr>
<td>((l_a, l_b, \cdot)), wait(^\S)</td>
<td>((l_a + l_h, 1, rel))</td>
<td>(\gamma) (\cdot (1 - \alpha))</td>
<td>((0, 0))</td>
</tr>
<tr>
<td>((l_a, l_b, \cdot)), match(^\S)</td>
<td>((l_a - l_h, 1, rel))</td>
<td>(\alpha)</td>
<td>((0, 0))</td>
</tr>
</tbody>
</table>

\(^\dagger\)Feasible only when \(l_a > l_h\). \(^\S\)Feasible only when \(l_a \geq l_h\). \(^\S\)If \(l_h < k - 1\), then the reward is \((0, l_h + 1)\) since no block was actually attacked. \(^\S\)If \(l_h < k - 1\), then the reward is \((0, l_h)\).

<table>
<thead>
<tr>
<th>(\alpha/\text{conf.})</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>2%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>5%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>10%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>14%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>18%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>22%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>26%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>30%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>34%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>38%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>42%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>46%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>50%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

Table 2: The fraction of the network’s blocks that an attacker with a given hashrate (\(\alpha\)) successfully attacks, when using an optimal attack policy, given the number of confirmations the acceptance policy waits for (\(\text{conf.}\)).

4 Revisiting the classic security analysis

As discussed in previous sections, a merchant can consider a specific payment in the blockchain as secure if he is willing to assume that the attack was carried out in an arbitrary point in time. Using this (implicit) assumption, previous works analyzed the security of a transaction \(tx\) in terms of the probability that the block containing it, \(B\), will not forever remain in the blockchain; i.e., in terms of Definition (2).

However, even under this assumption, the attacker can still engage in pre-mining, and try to gain an early advance before \(B\’s\) creation. Previous works that analyze the security of payments do not take this into account (with the exception of [Pass et al., 2016]; see Section 6), and we thus correct the analysis. Our main result (Lemma 1) is a probability distribution over the lead that the attacker may have at the time of the attack on a single block are included for comparison. Interestingly, the fraction of blocks that can be attacked, in the long term, is in fact lower than implied by Rosenfeld. This is because his analysis (and Satoshi’s as well) considers an attack on a single block that goes on infinitely—the attacker is assumed to never give up and to try to catch up with the chain no matter how far behind he is. In contrast, an attacker that aims to maximize the fraction of blocks he successfully attacks must occasionally give up and restart the attack if he is far behind. This effect is demonstrated in these results (note that, on the other hand, our model allows the attacker to double spend several blocks at once. The effective \(\epsilon\) is lower nonetheless). A similar effect occurs under different sizes of attacker or different numbers of confirmations.

3.3 Optimal attack policies

We now present the optimal attack policies returned by the algorithm, in two particular setups. Tables 3.4 describe the attack policy for an attacker with \(\alpha = 0.25\) and with \(\gamma = 0\) or \(\gamma = 0.5\). The row numbers correspond to the length of the attackers branch \(l_a\) and the columns to the length of the honest network’s branch \(l_h\). Actions are abbreviated to their initials: \(\text{adopt, override, match, wait}\), while the token ‘\(\cdot\)’ represents an unreachable state. When \(\gamma > 0\) the match action becomes feasible (see Table 1). Accordingly, each entry in the bottom table contains a string of three characters, corresponding to the possible values of the \(\text{fork}\) variable.

Notice that here the attacker does not override the network’s chain until the honest branch is of length 2 or more, as a successful attack requires that the merchant sees 2 confirmations above his transaction before the attack is released. Note further that the attacker does not give up on his attack when he is just slightly behind. If his chain is relatively long, he will not abandon it unless he is at least 2 blocks behind.
creation of $B$, which terminates the pre-mining attack stage. Using this result, we proceed to calculate in Theorem 2 the probability defined in (2), by augmenting and correcting the analysis from [Rosenfeld, 2014]:

**Lemma 1.** Let $B$ be an arbitrary block in $C^t$, and denote by $(l_a, l_b)$ the respective lengths of the attacker and the public chain at the time of $B$’s creation. Then, for any $l \geq 0$, under the optimal attack policy: $\Pr(l_a - l_b = l) = \frac{1 - 2\cdot \alpha}{1 - \alpha} \cdot \left(\frac{\alpha}{1 - \alpha}\right)^l$.

**Theorem 2.**

$$\Pr(\exists s > t : B \notin C^s \mid \text{height}(C^s) - \text{height}(B) = k - 1) = \sum_{l=0}^{\infty} \frac{1 - 2\cdot \alpha}{1 - \alpha} \cdot \left(\frac{\alpha}{1 - \alpha}\right)^l \cdot \left(\sum_{m=0}^{k-l} \frac{m + k - 1}{m} \cdot \alpha^m \cdot (1 - \alpha)^k \cdot \left(\frac{\alpha}{1 - \alpha}\right)^{k+1-m-l} + \sum_{m=k-l+1}^{\infty} \frac{m + k - 1}{m} \cdot \alpha^m \cdot (1 - \alpha)^k\right)$$

The proofs of Lemma 1 and Theorems 2-4 appear in the full version of the paper. Table 5 shows the difference between the corrected analysis and the original one, for an attacker with $\alpha = 0.3$. The uncorrected analysis is a variation of Rosenfeld [2014]. This result can be used by merchants to correctly choose $\epsilon$-arbitrary-robust acceptance policies.

**5 The Generalized Vector76 Attack**

In this section we present the Generalized Vector76 attack. The attack is aimed against lightweight clients, which typically keep track of the longest chain of blocks but do not relay blocks they receive to other nodes. As a result, even if the chain held by such a node is the longest one, it is not necessarily published—the chain could have originated from an attacker node which hasn’t broadcast it yet. The attack proceeds as follows:

1. The attacker starts working on a secret branch of the chain. It embeds the transaction $tx_1$ (that it later wishes to reverse) in its first block.
2. If the merchant requires $\sigma \equiv k$ confirmations, the attacker needs to build an additional $k - 1$ blocks on top of the one containing $tx_1$ (for a total of $k$ confirmations). He attempts to do so in secret.
3. If his branch of the chain is longer than that of the public chain, at some point after he has $k$ confirmations for $tx_1$, he shows the $k$ confirmations to the lightweight client which then accepts it as the legitimate chain, since it is the longest one.
4. The attacker then transmits a conflicting transaction $tx_2$ to the public network. As the honest network is not aware of the attacker’s chain, the public chain will grow long enough for $tx_2$ to be accepted by all nodes (and eventually even by the attacked one).

Figure 3 depicts the attack. Again, notice that a crucial stage in the success of the attack is that the honest network does not adopt the block containing $tx_1$.

The following theorem states that a merchant running a lightweight Bitcoin node is less secure than a full node:

**Theorem 3.** Let $\epsilon > 0$. If $\sigma_{\alpha, \gamma}$ is an $\epsilon$-fractional-robust policy for a non-relaying Bitcoin node, then there exists an $0 < \epsilon' < \epsilon$ such that $\sigma_{\alpha, \gamma}$ is an $\epsilon'$-fractional-robust policy for a full Bitcoin node.

Fortunately, we can still upper bound the success-probability of attacks on lightweight clients, using the following acceptance policy. Let $\sigma^{spv}_{\alpha, \gamma} := \min \{k \in \mathbb{N} : g(k, \alpha) < \epsilon \cdot (1 - \alpha)^k\}$, where $g(k, \alpha) := \sum_{n=0}^{\infty} \left(\frac{\alpha}{1 - \alpha}\right)^l \cdot \left(\sum_{n=0}^{k} \frac{n + k - 1}{n} \cdot \alpha^k \cdot (1 - \alpha)^k \cdot \left(\frac{\alpha}{1 - \alpha}\right)^{k+1-l} + \sum_{n=k-l+1}^{\infty} \frac{n + k - 1}{n} \cdot \alpha^{k-t} \cdot (1 - \alpha)^{k+t}\right)$.

**Theorem 4.** For any $\epsilon > 0$, the policy $\sigma^{spv}_{\alpha, \gamma}$ is $\epsilon$-fractional-robust, even when run by a non-relaying Bitcoin node.

**6 Related work**

Since Satoshi’s analysis in his white paper, several works have dealt with correcting and extending his analysis [Sompolinsky and Zohar, 2015; Garay et al., 2015; Karame et

---

3A simpler version was suggested by a user named Vector76 in the bitcoinTalk forums to possibly explain a successful double spending attack against the MyBitcoin e-wallet [Vector76, 2011].
<table>
<thead>
<tr>
<th># conf</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>without pre-mining</td>
<td>0.3086</td>
<td>0.2330</td>
<td>0.1801</td>
<td>0.1412</td>
<td>0.1117</td>
<td>0.0891</td>
<td>0.0714</td>
<td>0.0575</td>
<td>0.0465</td>
<td>0.0376</td>
</tr>
<tr>
<td>with pre-mining</td>
<td>0.5069</td>
<td>0.3952</td>
<td>0.3097</td>
<td>0.2463</td>
<td>0.1973</td>
<td>0.1388</td>
<td>0.1283</td>
<td>0.1040</td>
<td>0.0846</td>
<td>0.0688</td>
</tr>
</tbody>
</table>

Table 5: The success-probability of an attack on an arbitrary block, for an attacker with $\alpha = 0.3$, $\gamma = 0$, with and without pre-mining.

As the attack begins the attacker starts working on a significant secret chain with $tx_1$ inside its first block (1). If the attacker’s chain is far behind it may restart the attack (2). The attacker manages to gain a lead of 1 blocks, and has the two confirmations on his $tx_1$ needed to convince the victim (3). He then reveals the secret chain to the victim (that does not relay it), and collects an item in exchange. He then transmits the double spending transactions $tx_2$ to the network which is then included in a block (4). The network continues to mine atop $tx_2$ and it eventually prevails (5).

Figure 3: The progression of a generalized Vector76 attack on a 2-confirmation merchant

7 Conclusion

In this work we provided a formal security model for bitcoin transactions. We demonstrated that it is not enough to analyze blocks’ robustness, namely, the probability that a given block remains forever in the longest chain. Indeed, an attacker can selectively embed transactions in blocks whenever the conditions are in his favor. Specifically, he can wait for his secret pre-mined forks to obtain a sufficient lead over the public chain, before carrying out the attack.

In our fractional security model, transactions are secure if the attacker cannot reverse a significant portion of them. We devised an algorithm to compute the worst-case attacker, under this model. The resulting analysis is more tight, as an attacker must trade-off his current attack with future ones.

We additionally revisited the classic security model which assumes that the attack is carried out in an arbitrary point in time (which was not controlled by the attacker). We’ve shown that transactions are less secure than previously claimed, due to the pre-mining attack stage.

Finally, we’ve formalized the Vector76 attack, which proves that Bitcoin nodes that do not relay blocks can more easily be defrauded—an attacker can feed to such a node a chain of blocks which will not become part of the public chain. Thus, such nodes need to wait longer in order to meet the same level of security as full nodes which do relay blocks to their peers.
References


PARTIALLY OBSERVABLE CONTINGENT PLANNING FOR
PENETRATION TESTING: DORIN SHMARYAHU, GUY SHANI, JOERG
HOFFMANN AND MARCEL STEINMETZ
Partially Observable Contingent Planning for Penetration Testing

Dorin Shmaryahu and Guy Shani
Information Systems Engineering
Ben Gurion University, Israel

Joerg Hoffmann and Marcel Steinmetz
Department of Computer Science
Saarland University, Germany

Abstract
Penetration Testing (pentesting), where network administrators automatically attack their own network to identify and fix vulnerabilities, has recently received attention from the AI community. Smart algorithms that can identify robust and efficient attack plans may imitate human hackers better than simple protocols. Classical planning methods for pentesting model poorly the real world, where the attacker has only partial information concerning the network. On the other hand POMDP-based approaches provide a strong model, but fail to scale up to reasonable model sizes. In this paper we offer a middle ground, allowing for partial observability and non-deterministic action effects, by modeling pentesting as a partially observable contingent problem. We experiment with a real network of a large organization, showing our solver to scale to realistic problem sizes. We also experiment with sub-sampled networks, comparing the expected reward of a contingent plan graph to that of a POMDP policy.

1 Introduction
Penetration testing (pentesting) is a popular technique for identifying vulnerabilities in networks, by launching controlled attacks (Burns et al. 2007). A successful, or even a partially successful attack reveals weaknesses in the network, and allows the network administrators to remedy these weaknesses. Such attacks typically begin at one entrance point, and advance from one machine to another, through the network connections. For each attacked machine a series of known exploits is attempted, based on the machine configuration, until a successful exploit occurs. Then, this machine is controlled by the attacker, who can launch new attacks on connected machines. The attack continues until a machine inside the secure network is controlled, at which point the attacker can access data stored inside the secured network, or damage the network.

In automated planning the goal of an agent is to produce a plan to achieve specific goals, typically minimizing some performance metric such as overall cost. There are many variants of single agent automated planning problems, ranging from fully observable, deterministic domains, to partially observable, non-deterministic or stochastic domains. Automated planning was previously suggested as a tool for conducting pentesting, exploring the two extreme cases — a classical planning approach, where all actions are deterministic, and the entire network structure and machine configuration are known, and a POMDP approach, where machine configuration are unknown, but can be noisily sensed, and action outcomes are stochastic.

The classical planning approach scales well for large networks, and has therefore been used in practice for pentesting. However, the simplifying assumptions of complete knowledge and fully deterministic outcomes results in an overly optimistic attacker point-of-view. It may well be that a classical-planning attack has a significantly lower cost than a real attack, identifying vulnerabilities that are unlikely to be found and exploited by actual attackers. MDPs (Durkota et al. 2015; Hoffmann 2015) provide a slightly more realistic description of the problem, allowing for actions to fail. Still, like classical planning, MDPs do not measure the partial information that an attacker may have, and the sensing actions that may be needed.

The POMDP approach on the other hand (Sarraute et al. 2011), models the problem better, and can be argued to be a valid representation of the real world. One can model the prior probabilities of various configurations for each machine as a probability distribution over possible states, known as a belief. Probing actions, designed to reveal configuration properties of machines are modeled as sensing actions, and a probability distribution can be defined for the possible failure in probing a machine. The success or failure of attempting an exploit over a machine can be modeled as a stochastic effect of actions.

This approach, however, has two major weaknesses — first, POMDP solvers do not scale to the required network size and possible configurations. Second, a POMDP requires accurate probability distributions for initial belief, sensing accuracy, and action outcomes. In pentesting, as in many other applications, it is unclear how the agent can reliably obtain these distributions. In particular, identifying an accurate probability distribution over the possible OS for the machines in the network. Prior work (Sarraute et al. ) has devised only a first over-simplifying model of "software updates".

In this paper we suggest an intermediate model between classical planning and POMDPs. We replace the POMDP definition with partially observable contingent planning, a qualitative model where probability distributions are re-
placed with sets of possible configurations or action effects (Albore et al. 2009; Muise et al. 2014; Komarnitsky and Shani 2014). Solvers for this type of models scale better than POMDP solvers, and can be used for more practical networks. As these models require no probabilities, we avoid the guesswork inherent in their specification.

Contingent planners attempt to find a plan tree (or graph), where nodes are labeled by actions, and edges are labeled by observations. This plan tree is a solution to the problem if all leaves represent goal states.

We experiment with a network of a large organization, with the real vulnerabilities that were found in a scan of that network, showing that our contingent planner computes a plan graph for this network. In addition, we compare plan graphs to POMDP policies for much smaller networks that were sub-sampled from the real network data that we collected.

2 Networks and Pentesting

We begin by providing a short background on pentesting.

We can model networks as directed graphs whose vertices are a set $M$ of machines, and edges representing connections between pairs of $m \in M$. Like previous work in the area, we assume below that the attacker knows the structure of the network. But this assumption can be easily removed in our approach. We can add sensing actions that test the outgoing edges from a controlled host to identify its immediate neighbors. From an optimization prespective, though, not knowing anything about the network structure, makes it difficult to create smart attacks, and the attacker is forced to blindly tread into the network. It might well be that some partial information concerning the network structure is known to the attacker, while additional information must be sensed. We leave discussion of interesting forms of partial knowledge to future work.

Each machine in the network can have a different configuration representing its hardware, operating system, installed updates and service packs, installed software, and so forth. The network configuration is the set of all machine configurations in the network.

Machine configuration may be revealed using sensing techniques. For example, if a certain series of 4 TCP requests are sent at exact time intervals to a target machine, the responses of the target machine vary between different versions of Windows (Lyon 2009). In many cases several different such methods must be combined to identify the operating system. Sending such seemingly innocent requests to a machine to identify its configuration is known as fingerprinting. Not all the properties of a target machine can be identified. For example, one may determine that a certain machine runs Windows XP, but not which security update is installed.

Many configurations have vulnerabilities that can be exploited to gain control over the machine, but these vulnerabilities vary between configurations. Thus, to control a machine, one first probes it to identify some configuration properties, and based on these properties attempts several appropriate exploits. As the attacker cannot fully observe the configuration, these exploits may succeed, giving the attacker full control of the target machine, or fail as some undetectable configuration property made this exploit useless.

The objective of penetration testing (pentesting) is to gain control over certain machines that possess critical content in the network. We say that a machine $m$ is controlled if it has already been hacked into, and the attacker can use it to fingerprint and attack other machines. A reached machine $m$ is connected to a controlled machine. All other machines are not reached. We assume that the attacker starts controlling the internet, and all machines that are directly connected to the internet are reached.

We will use the following (small but real-life) situation as an illustrative example (Sarraute et al.):

Example 2.1. The attacker has already hacked into a machine $m'$, and now wishes to attack a reached machine $m$. The attacker may try one of two exploits: SA, the “Symantec Rtvscan buffer overflow exploit”; and CAU, the “CA Unicenter message queuing exploit”. SA targets a particular version of “Symantec Antivirus”, that usually listens on port 2967. CAU targets a particular version of “CA Unicenter”, that usually listens on port 6668. Both work only if a protection mechanism called DEP (“Data Execution Prevention”) is disabled. The attacker cannot directly observe whether DEP is enabled or not.

If SA fails, then it is likely that CAU will fail as well because DEP is enabled. Hence, upon observing the result of the SA exploit, the attacker learns whether DEP is enabled. The attacker is then better off trying other exploits instead. Achieving such behavior requires the attack plan to observe the outcomes of actions, and to react accordingly. Classical planning which assumes perfect world knowledge at planning time cannot model such behaviors.

2.1 POMDPs for Pentesting

Partially observable Markov decision processes (POMDPs) (Sondik 1978) were previously suggested as a strong modeling tool for pentesting (Sarraute et al.).

A POMDP is a tuple $(S, A, \Omega, tr, O, R, b_0)$, where $S$ is a set of states, $A$ is a set of actions, $\Omega$ is a set of possible observations, $tr(s, a, s')$ is the probability of transitioning from a state $s$ to a state $s'$ using action $a$, $O(a, s', o)$ is the probability of observing $o \in \Omega$ after executing action $a$, arriving at state $s'$, $R(s, a, s')$ is the reward (or cost) for executing action $a$ in state $s$ arriving at state $s'$, and $b_0$ is the initial belief — a probability distribution over the possible initial states.

Sarraute et al. model pentesting using a POMDP where the states are the possible configuration of the network. That is, a state defines for each machine in the network its operating system (OS), open ports, running software, and vulnerabilities. The initial belief is hence a probability distribution over the possible network configurations.

There are sensing actions, that do not change the state of the world, that is, $tr(s, a_{sense}, s) = 1$, but provide information about certain machine properties, such as its OS, through the observations distribution. The states change when using exploit actions, resulting in a new state where an attacked machine becomes controlled by the attacker. In addition, there is a special terminate action, that moves the POMDP to a terminal state.
Sarraute et al. set costs to all actions, a reward when taking control of any machine, and a larger reward when taking control of some important machines. They experiment with deterministic POMDPs, where actions have deterministic effects, and observations are also deterministic, but POMDPs can also be used to model sensing noise, and stochastic success of exploits.

Sarraute et al. show that the constructed POMDPs can be solved by a POMDP solver (Kurniawati et al. 2008) only for small problems.

3 Contingent Planning Model and Language

A contingent planning problem is a tuple \( \langle P, A_{act}, A_{sense}, \phi_I, G > \), where \( P \) is a set of propositions, \( A_{act} \) is a set of actuation actions, and \( A_{sense} \) is a set of sensing actions. An actuation action is defined by a set of preconditions — propositions that must hold prior to executing the actions, and effects — propositions that hold after executing the action. A sensing action \( a_{sense} \) has preconditions, but no effects. Instead, \( a_{sense} \) reveals the value of a proposition. \( \phi_I \) is a propositional formula describing the set of initially possible states. \( G \subseteq P \) is a set of goal propositions.

In our pentesting application, \( P \) contains propositions describing machine configuration, such as \( OS(m_i, winxp) \), denoting that machine \( m_i \) runs the OS Windows XP. Similarly, \( SW(m_i, IIS) \) represents the existence of the software IIS on machine \( m_i \). In addition, the proposition \( controlling(m_i) \) denotes that the attacker currently controls \( m_i \), and the proposition \( hacl(m_i, m_j) \) denotes that machine \( m_i \) is directly connected to machine \( m_j \).

The set \( A_{sense} \) in our pentesting model represents the set of possible queries that one machine can launch on another, directly connected machine, probing it for various properties, such as its OS, software that runs on it, and so forth. Each such sensing action requires as pre-condition only that the machines will be connected, and reveals the value of a specific property. In some cases there are certain “groups” of operating systems, such as Windows XP with varying service packs and updates installed. In this case we can allow one property for the group \( (OS(m_i, winxp)) \) and another property for the version, such as \( (OSversion(m_i, winxp, pl)) \) which may not be observable by the attacker.

The set \( A_{act} \) in our pentesting model contains all the possible exploits. We create an action \( a_{act}(m_{source}, m_{target}) \) for each exploit \( e \) and a pair of directly connected machines \( m_{source}, m_{target} \). If an exploit \( e \) is applicable only to machines running Windows XP, then \( OS(m_{target}, winxp) \) would appear in the preconditions. Another precondition is \( controlling(m_{source}) \) denoting that the attacker must control \( m_{source} \) before launching attacks from it. The effect of the action can be \( controlling(m_{target}) \), but we further allow the effect to depend on some hidden property \( p \) that cannot be sensed. This is modeled by a conditional effect \( (p, controlling(m_{target})) \) denoting that if property \( p \) exists on \( m_{target} \) than following the action the attacker controls \( m_{target} \).

Belief states in contingent planning are sets of possible states, and can often be compactly represented by logic formulas. The initial belief formula \( \phi_I \) represents the knowledge of the attacker over the possible configurations of each machine. For example oneof \((OS(m_i, winxp), OS(m_i, winnt4), OS(m_i, win7)) \) states that the possible operating systems for machine \( m_i \) are Windows XP, Windows NT4, and Windows 7.

Like Sarraute et al., we assume no non-determinism, i.e., if all properties of a configuration are known, then we can predict deterministically whether an exploit will succeed. We do allow for non-observable properties, such as the service pack installed for the specific operating system. We support actions for sensing whether an exploit has succeeded. Hence, observing the result of an exploit action reveals information concerning these hidden properties.

Example 3.1. We illustrate the above ideas using a very small example, written in a PDDL-like language for describing contingent problems (Albore et al. 2009).

We use propositions to describe the various properties of the machines and the network. For example, the predicate \( (hacl ?m_1 ?m_2) \) specifies whether machine \( m_1 \) is connected to machine \( m_2 \), and the predicate \( (HostOS ?m ?o) \) specifies whether machine \( m \) runs OS \( o \). While in this simple example we observe the specific OS, we could separate OS type and edition (say, Windows NT4 is the type, while Server or Enterprise is the edition). We can then allow different sensing actions for type and edition, or allow only sensing of type while edition cannot be directly sensed.

We define actions for probing certain properties. For example, the \( probe-os \) action:

\[
\begin{align*}
: & \text{action ping-os} \\
& : \text{parameters (?src - host ?target - host ?o - os)} \\
& : \text{precondition (and (hacl ?src ?target) (controlling ?src)} (\text{not (controlling ?target)}) \\
& : \text{observe (HostOS ?target ?o)} \\
\end{align*}
\]

allows an attacker that controls host \( s \) connected to an un-controlled host \( t \), to probe it to identify whether it’s OS is \( o \).

We allow for a similar probe action for installed software. The \( exploit \) action attempts to attack a machine exploiting a specific vulnerability:

\[
\begin{align*}
: & \text{action exploit} \\
& : \text{parameters (?src - host ?target - host ?o - os ?sw - sw ?v - vuln)} \\
& : \text{effect (when (ExistVuln ?v ?target) (controlling ?target)}) \\
\end{align*}
\]

The preconditions specify that the machines must be connected, that the OS is \( o \) and the software \( sw \) is installed, and that the vulnerability \( v \) which we intend to exploit matches the specific OS and software.
The success of the exploit depends on whether the vulnerability exists on the target machine, which manifests in the conditional effect. The attacker cannot directly observe whether a specific vulnerability exists, but can use the CheckControl action to check whether the exploit has succeeded:

\[
\text{: action CheckControl}
\]
\[
\text{ parameters ( ? src = host ? target = host )}
\]
\[
\text{ precondition ( and ( hacl ? src ? target ? p )}
\]
\[
\text{ ( controlling ? src ) )}
\]
\[
\text{ observe ( controlling ? target )}
\]

The initial state of the problem describes the knowledge of the attacker prior to launching an attack:

\[
\text{: initial}
\]
\[
1: \ ( \text{controlling internet} )
\]
\[
2: \ ( \text{hacl internet host0} )
\]
\[
( \text{hacl internet host1} )
\]
\[
( \text{hacl host1 host2} )
\]
\[
( \text{hacl host0 host2} )
\]
\[
3: \ ( \text{oneof (HostOS host0 winNT4ser) (HostOS host0 winNT4ent) )}
\]
\[
( \text{oneof (HostOS host1 win?ent) (HostOS host1 winNT4ent) )}
\]
\[
\text{...}
\]
\[
4: \ ( \text{oneof (HostSW host0 IIS4) (HostSW host1 IIS4) )}
\]
\[
\text{...}
\]
\[
5: \ ( \text{Match winNT4ser IIS4 CVE-X-Y) }
\]
\[
\text{...}
\]
\[
6: \ ( \text{or (ExistVuln CVE-X-Y host0) (ExistVuln CVE-Z-W host0) )}
\]
\[
\text{...}
\]

We state that initially the attacker controls the “internet” only (part 1). In this case the structure of the network is known, described by the hacl statements (part 2). Then, we describe which operating systems are possible for each of the hosts (part 3). Below, we specify that either host0 or host1 are running the software IIS (part 4). We describe which vulnerability is relevant to a certain OS-software pair (part 5), and then describe which vulnerabilities exist on the various hosts (part 6).

The above specification may allow for a configuration where no vulnerability exists on a host (machine) that matches the host OS and software. Hence, none of the exploits will work for that specific host.

4 Contingent Plan Trees for Pentesting

A solution to a contingent planning problem is a plan tree, where nodes are labeled by actions. A node labeled by an action is applicable if there is a single child, and a node labeled by a sensing action has two children, and each outgoing edge to a child is labeled by a possible observation.

An action \( a \) is applicable in belief state \( b \), if for all \( s \in b \), \( s \models \text{pre}(a) \). The belief state \( b \) resulting from the execution of \( a \) in \( b \) is denoted \( a(b) \). We note the execution of a sequence of actions \( a_i^1 \ldots a_i^n \) starting from belief state \( b \) by \( a_i^n(b) \). Such an execution is valid if for all \( i \), \( a_i \) is applicable in \( a_i^{i-1}(b) \).

Plan trees can often be represented more compactly as plan graphs (Komaritsky and Shani 2014; Muise et al. 2014), where certain branches are unified. This can lead to a much more compact representation, and to scaling up to larger domains. Still, for ease of exposition, we discuss below plan trees rather than graphs.

In general contingent planning, a plan tree is a solution, if every branch in the tree from the root to a leaf, labeled by actions \( a_i^1 \ldots a_i^n \), \( a_i^n(b) \models G \). In pentesting, however, it may not be possible to reach the goal in all cases, because there may be network configurations from which the target machine simply cannot be reached. To cater for this, we need to permit plan trees that contain dead-ends. We define a dead-end to be a state from which there is no path to the goal, given any future sequence of observations. That is, any plan tree starting from a dead-end state would not reach the goal in any of its branches. For example, a dead-end state arises if no exploit is applicable for the goal machine. It is clearly advisable to stop the plan (the attack) at such states. On the other hand, if a state is not a dead-end, then there still is a chance to reach the target so the plan/attack should continue.

There is hence need to define contingent plans where some of the branches may end in dead-ends. A simple solution, customary in probabilistic models, is to introduce a give-up action which allows to achieve the goal from any state. Setting the cost of that action (its negative reward) controls the extent to which the attacker will be persistent, through the mechanism of expected cost/expected reward.

In a qualitative model like ours, it is not as clear what the cost of giving up (effectively, of flagging a state as “dead-end” and disregarding it) should be. It may be possible to set this cost high enough to force the planner to give up only on dead-ends as defined above. But then, the contingent planner would effectively need to search all contingent plans not giving up, before being able to give up even once.

We therefore employ here a different approach, allowing the planner to give-up on states if it can prove that \( s \) is a dead-end. Such proofs can be lead by classical-planning dead-end detection methods, like relaxation/abstraction heuristics, adapted to our context by determining the sensing actions, allowing the dead-end detector to choose the outcome. In other words, we employ a sufficient criterion to detect dead-end states, and we make the give-up action applicable only on such states. As, beneath all dead-ends, eventually the pen-test will run out of applicable actions, eventually every dead-end will be detected and the give-up enabled.

In general, this definition would not be enough because the planner could willfully choose to move into a dead-end, thereby “solving” the task by earning the right to give up. This cannot happen, however, in the pentesting application, as all dead-ends are unavoidable, in the following sense. Say \( N \) is a node in our plan tree \( T \), and denote by \( |N| \) those initial states from which the execution of \( T \) will reach \( N \). If \( N \) is a dead-end, then every \( I \in |N| \) is unsolvable, i.e., there does not exist any sequence of \( A_{i,w} \) actions leading from \( I \) to the goal. In other words, any dead-end the contingent plan may encounter is, in the pentesting application, inherent in the initial state.
5 Network Data Acquisition

To test our approach we created realistic models using data obtained from scanning the network of a large organization, containing several subnets. Using the machine configurations and existing exploits discovered using the scan, we can create real world models that allow us to provide an empirical evaluation of our approach. We now provide some explanations of the model and the network, unfortunately omitting many details due to confidentiality restrictions.

To collect the needed information for our models, we began by running a scan of the various subnets using the Nessus scanner\(^1\). Nessus starts its scan from a given computer, and identifies all reachable hosts from that computer, including desktops, gateways, switches, and more.

As Nessus does not actually launches attacks to control a host, a Nessus scan identifies only hosts that are directly logically reachable from the source machine where the scan is running, possibly through several switches and gateways. We hence executed several such scans, each from a different subnet within the organization, as well as one scan from outside the organization network.

The resulting scans contain the set of machines that are visible from each source machine. The machines inside a subnet are all visible to each other. Hence, we assume that all machines within a subnet can directly access the machines that the representative source machine can access. Only a part of the machines outside the subnet are visible from within the subnet, due, e.g., to firewall restrictions. We model the accessibility of machines identified through the scans as direct edges in the network graph. That is, machine \(m_1\) is connected in our model to machine \(m_2\), if \(m_2\) is visible from \(m_1\) or vice versa.

In addition, Nessus reveals for each identified host its operating system. The Network contains hosts running Windows and Linux (with a few versions of each operating system). Nessus also identifies softwares with potential vulnerabilities that run on the machines. Our model contains about 50 such software, including well known applications such as openssh, tomcat, pcanywhere, ftp services, and many more.

Nessus identifies potential vulnerabilities in the scanned machines. These vulnerabilities may not actually exist, but the only way to know is by performing an exploit for that vulnerability, which of course we did not do. As we explain above, we model the uncertainty about the existence of the vulnerability directly in our model. The agent must attempt an exploit and check afterwards whether the exploit was successful, and hence, whether the vulnerability actually exists.

Nessus finds vulnerabilities of varying importance. For the purpose of this experiment we ignored all the lesser vulnerabilities, which do not allow an attacker control of the system. We remain with about 60 serious types of vulnerabilities that exist in the network. We remove from the network all hosts that do not run any software for which a serious vulnerability exists, remaining with about 35 hosts.

For constructing our pnetesting goal, we took two random hosts from the innermost subnet, and set them as the target hosts. The problem goal is to gain control over one of these two hosts. To add uncertainty into the model, we specify in the initial belief for each host, aside from the true operating system and the real applications running on it, one more potential operating system, and 3 more possible applications.

5.1 POMDP Model

We use the above data to create a POMDP formulation of the network, following the overall guidelines set by Sarraute et al (2009), by differing on some details to be more compatible with our contingent planning modeling. We follow Sarraute et al, defining a state for each possible configuration of all network machines. We use deterministic vulnerability exploit actions that take control of a given machine. These action result also in a deterministic success or failure observation.

Our sensing actions support only binary observations. Hence, we replace Sarraute’s ProbeOS(host) sensing action with a set of sensing actions ProbeOS(host,os) sensing actions. The Nessus output relates a vulnerability to a software, rather than a port. We hence do not use ProbePort actions, but rather ProbeSW(host,software) actions. We ignore ports in our problem formalization, but open ports can also be taken into account using a slightly more complex description. The network connectivity structure is embedded into the transition and observation probabilities. Probing a machine that has no controlled neighbor results always in a false observation. Attempting an exploit on a machine that has no controlled neighbor does not change the state, even if the exploited vulnerability exists on the host.

Our reward structure is substantially different than suggested before. First, the Nessus output contains costs for the exploits that were identified (Lai and Hsia 2007). We use these costs for the exploit actions in our model. Probing a host for its operating system can be done by only listening to the network traffic from that host (Yarochkin et al. 2009). We hence set a very low cost (0.1) for ProbeOS(host,os) actions. Probing for running software is more costly, because it requires sending requests to that software awaiting responses. We hence set the cost of software probing to be equal to the least costly exploit (1 in our data).

Finally, we focus on modeling a goal directed approach, where the attacker goal is to take control over some target machine, perhaps containing some important information. We model this by rewarding the attacker only when terminating (using the terminate action), when one of the target machine is controlled, using a large reward (1000). There can be deadend states from which the target machine cannot be controlled (for example, when it has no vulnerabilities). The agent receives no penalty when terminating at a dead-end state. We add, however, a substantial penalty (-1000) for terminating when no target machine is controlled, in a state which is not a deadend. We consider these rewards to be the weakest part of our modeling approach, because they are not supported by the data, but induced by us to motivate the agent towards a desirable behavior. A deeper investigation into setting such rewards is left for future work.

---

\(^1\)https://www.tenable.com/products/nessus-vulnerability-scanner
6 Empirical Study

We now provide an empirical study of the contingent planning approach to modeling penetration testing. To obtain a plan tree (graph) for the pentesting contingent problems we use a modification of the offline planner CPOR (Komarnitsky and Shani 2014), which constructs an efficient plan graph by identifying states for which a solution was already computed. CPOR uses an online solver as a heuristic. We replace this component by a domain-specific heuristic.

Our heuristic analyzes the network graph, and identifies the next accessible host closest to the target machines. Then, the heuristic probes that host attempting to discover its operating system and running software. Finally, the heuristic attempts possible exploits for the identified operating system and software. Once the host is controlled, or if all exploits have failed, the heuristic chooses the next host to attack.

We augment CPOR with a simple deadend detection mechanism. If all paths from the set of controlled hosts to the target machines contain a host for which all possible exploits have failed, then there is no possible path to the target machines, and an unavoidable deadend is declared.

6.1 Real Network

When running the contingent planner, we avoid traversing branches that correspond to these additions, and cannot be reached in reality. For example, if the true operating system of host $m_1$ is Windows 7, we may add Windows NT as a second operating system. The planner then uses a ping action to sense the operating system. For instance, the planner may choose to probe for Windows NT first. In reality, the host was running Windows 7, and hence, the attacker will receive a false observation for this action. We allow the planner to execute the probe action, but then traverse only the branch where the observation conformed to the true one. That is, in this example we will only traverse the false child, ignoring the true child. This better simulates the attacks of a real attacker on this real network, ignoring impossible branches. As such, although the plan $s$ contains many ping actions, our plan tree branches only on the success or failure of exploits.

We ran our contingent planner over the resulting problem. The planner computed a plan graph in 116 seconds, containing 1453 nodes. The equivalent POMDP with 35 hosts, 2 operating system, 50 software, and 60 vulnerabilities, assuming 25 software for each operating system, and 5 average vulnerabilities per software, has a state space of about $2^{35} \times 2^5 \times 5^6 = 28750$, which is far beyond the capabilities of POMDP solvers. Even factored POMDP solvers (Shani et al. 2008; Veiga et al. 2014) do not scale up to these sizes.

Table 1: Comparing expected discounted reward and time (secs), over small networks sampled from the real network distributions. $n$, $k$, $m$ are the number of machines, software, and vulnerabilities, respectively. $|S|$ and $|A|$ are the number of states and actions in the POMDP problem.

| $n$ | $k$ | $m$ | $|S|$ | $|A|$ | Time | $E(R)$ |
|-----|-----|-----|------|------|------|-------|
| 2   | 4   | 6   | 203  | 22   | 0.75 | 291.27 |
| 3   | 2   | 5   | 693  | 25   | 5.11 | 265.75 |
| 3   | 2   | 4   | 125  | 19   | 0.25 | 393.33 |
| 2   | 4   | 4   | 4913 | 34   | 5.78 | 269.72 |
| 3   | 4   | 6   | 8323 | 32   | 3.34 | 139.23 |
| 3   | 5   | 7   | 20213| 38   | N/A  | 114.51 |

| $n$ | $k$ | $m$ | $|S|$ | $|A|$ | Time | $E(R)$ |
|-----|-----|-----|------|------|------|-------|
| 2   | 4   | 6   | 203  | 22   | 0.75 | 291.27 |
| 3   | 2   | 5   | 693  | 25   | 5.11 | 265.75 |
| 3   | 2   | 4   | 125  | 19   | 0.25 | 393.33 |
| 2   | 4   | 4   | 4913 | 34   | 5.78 | 269.72 |
| 3   | 4   | 6   | 8323 | 32   | 3.34 | 139.23 |
| 3   | 5   | 7   | 20213| 38   | N/A  | 114.51 |

Table 1: Comparing expected discounted reward and time (secs), over small networks sampled from the real network distributions. $n$, $k$, $m$ are the number of machines, software, and vulnerabilities, respectively. $|S|$ and $|A|$ are the number of states and actions in the POMDP problem.

connected machines from the various subnets of the network. We sample a set of $m$ software for each machine, from its real set of running software, following the frequency of the software given an operating system in our scan. We then sample $k$ vulnerabilities for each software, again following the frequency of vulnerabilities in our data. Each software and vulnerability can either exist on a machine or not. This process provides a set of configurations, and we assume that each machine can have any of these possible configurations.

We compute the probabilities of a configuration following the distribution of operating systems, software, and vulnerabilities in our scan. The probability of a machine running operating system $o$, software $s$, and vulnerability $v$, is computed using $pr(o) \times pr(s|o) \times pr(v|s)$, where the probabilities are the maximum likelihood estimators from the data, normalized to the reduced sample in the particular instance.

Table 1 compares the expected discounted reward of our plan graph to that of the SARSOP policy for the equivalent POMDP model. As can be seen, the expected reward of the contingent planning solution is lower than the expected reward of POMDP solution. This can be attributed in part to the heuristic in our planner, that intentionally ignores costs and probabilities. Adding these factors to the heuristic selection of actions is left for future research.

On the other hand, the POMDP solver fails even on very small networks, with only 3 machines, 4 software, and 6 vulnerabilities. This is clearly far below acceptable model sizes.

7 Conclusion and Future Work

We suggest contingent planning as an alternative for modeling pentesting. This model allows for partial observability of various properties, such as a machine operating system and installed software, that can be sensed by probe actions. Thus, contingent planning offers a richer model than classical planning, while being able to scale up better than POMDP-based approaches. We show that our approach scales to real network sizes far beyond the capabilities of current POMDP solvers, and compare its expected reward to that of a POMDP over smaller sub-sampled networks.

In the future we intend to create smarter heuristics for ordering actions given states, to achieve better expected rewards. In addition, we intend to experiment with a factored representation of the POMDP problem, to try scaling up to
more reasonable problem sizes.

References


Carlos Sarraute, Olivier Buffet, and Jörg Hoffmann. POMDPs make better hackers: Accounting for uncertainty in penetration testing.

Carlos Sarraute, Olivier Buffet, and Jörg Hoffmann. Penetration testing == POMDP solving? In *SecArt'11*, 2011.


Ranking Vulnerability Fixes Using Planning Graph Analysis

Tom Gonda, Guy Shani, Rami Puzis, Bracha Shapira
SISE Department
Ben Gurion University, Israel
{tomgond,shanigu,puzis,bshapira}@bgu.ac.il

Abstract
During the past years logical attack graphs were used to find the most critical vulnerabilities and devise efficient hardening strategies for organizational networks. Most techniques for ranking vulnerabilities either do not scale well, e.g. brute-force attack plan enumeration, or are not well suited for the analysis of logical attack graphs, e.g. centrality measures.

In this paper we suggest an analysis of the planning graph (from classical planning) derived from the logical attack graph to improve the accuracy of centrality-based vulnerability ranking metrics. The planning graph also allows efficient enumeration of the set of possible attack plans that use a given vulnerability on a specific machine. We suggest a set of centrality based heuristics for reducing the number of attack plans and compare with previously suggested vulnerability ranking metrics. Results show that metrics computed over the planning graph are superior to metrics computed over the logical attack graph or the network connectivity graph.

1 Introduction
Large organizations use a vast and diverse set of software [Morrow, 2012]. As such, ensuring that all installed software are completely safe is an impossible task. The computer networks of large organizations can hence be penetrated by exploiting vulnerabilities in the installed software, operating system, or their combinations. Indeed, research has shown that even organizations whose core business is in developing security software have many vulnerabilities in their networks[Zhang et al., 2014].

These vulnerabilities can often be fixed. For example, when a vulnerability in a given program is identified, the software company maintaining the software often issues a patch fixing the vulnerability. Alternatively, if a given software is found to be too vulnerable, the security administrator can choose to move from Windows XP to Windows 10, or to move from Windows to Linux or vice versa. Of course, replacing the software often involves a significant cost [Shostack, 2003]. When moving from Windows to Linux one has to install software versions appropriate to Linux instead of the Windows versions. Thus, the system administrator must prioritize the fixes such that the more important vulnerabilities will be fixed first [Cukier and Panjwani, 2009].

Most research focuses on analyzing possible attacks on the network in order to rank the vulnerability fixes. A common data structure for conducting such analysis is the logical attack graph (LAG), whose nodes represent assets or vulnerability exploits, and edges represent which assets are needed before an exploit can be used, or which assets an exploit produces [Ou et al., 2005]. One can analyze the attack graph [Albanese et al., 2012] to gain better understanding of which vulnerabilities can be used to gain a specific sensitive information from a given starting point (e.g. when controlling only machines outside the organization).

Hoffmann et al [Hoffmann, 2015] suggested a different approach for identifying vulnerabilities, by computing directly attack plans for penetration testing (pentesting). An attack plan is a sequence of actions (e.g. exploits) that allow the attacker to achieve its goals, such as access to specific sensitive information. The system administrator can use these attack plans to decide which vulnerabilities to patch. We suggest taking this approach to the extreme, computing all possible attack plans together. Then, the set of all attack plans can be used for, e.g., identifying vulnerabilities that appear in more plans, ranking their fixes higher.

We explain how the (relaxed) planning graph — a data structure often used in the classical planning community, mainly to compute forward search heuristics — can be used to compute the set of all possible plans in our application. We provide an algorithm for enumerating all such plans using a backward scan of the planning graph.

Previous research has suggested various node centrality measures, such as betweenness [Hong and Kim, 2013], and pagerank [Sawilla and Ou, 2008], for ranking the vulnerabilities to be fixed. We demonstrate here that over a range of benchmarks including a scan of a large organization network, metrics computed over the planning graph provide a much better ranking compared to metrics computed over the logical attack graph. We focus here on the task of increasing the cost of the minimal attack. We rank different vulnerabilities by the various metrics, and show that ranking using metric over the planning graph increases the minimal attack cost compared
to rankings based on the logical attack graph. Moreover, for this task, pagerank has shown the best results.

2 Background
We now briefly review relevant background, starting with attack graphs and their use in ranking vulnerability fixes, and then discussing the planning graph data structure, and how it is used for classical planning in general, and for pentesting in particular.

2.1 Logical Attack Graphs
Logical attack graphs (LAGs) are graphs that represent the possible actions and outcomes of actions applied by an attacker trying to gain a goal asset in a system. An example of possible actions and outcomes of actions applied by an attacker is an attack graph can be seen in Figure 1.

We now describe the attack graph structure. The graph contains 3 types of nodes:

1. **Primitive fact nodes** represent facts about the system. For example, they can represent network connectivity, firewall rules, user accounts on various computer and more. In the example graph (Figure 1) primitive fact nodes are represented by rectangular nodes.

2. **Derivation nodes** (or action nodes) represent an action the attacker can take in order to gain a new capability in the system. The outcome of performing an action, is an instantiation of a new derived fact. Action nodes are represented in Figure 1 by ovals.

3. **Derived fact nodes** (or privilege nodes) represent a capability an attacker gains after performing an action (derivation phase). For example, a node stating that the attacker can execute arbitrary code on a specific machine with certain privileges. Derived fact nodes are represented by diamonds in Figure 1.

Edges in the LAG from a fact node to an action node represent the dependency of the action on the facts, and edges from an action to fact represent the derivation of that fact following the action.

**Definition 2.1.** Logical attack graph. Formally, a logical attack graph is a tuple:

$$G = (N_p, N_c, N_e, E, L, g)$$

Where $$N_p, N_c$$ and $$N_e$$ are three sets of disjoint nodes in the graph, $$E$$ is a set of directed edges in the graph where

$$E \subseteq (N_e \times N_p) \cup ((N_p \cup N_c) \times N_e)$$

$$L$$ is a mapping from a node to its label, and $$g \in N_p$$ is the attacker’s goal (multiple goals can be transformed into a single goal using an action with preconditions as the multiple goals). $$N_p, N_c$$ and $$N_e$$ are the sets of privilege nodes, action nodes and primitive fact nodes, respectively.

The edges in an LAG are directed. There are two types of edges in attack graph: $$(p, a)$$ an edge from an action node to a derived fact node, stating that by applying $$a$$ an attacker can gain privilege $$p$$. $$(p, a)$$ is an edge from a fact (either primitive or derived) node to an action node, stating that $$p$$ is a precondition to action $$a$$. For example, in order to apply exploit $$e$$ on machine $$m_2$$ from machine $$m_1$$, there must be a connection from $$m_1$$ to $$m_2$$ (represented by a primitive fact node $$p$$), and the user must have already gained access to code execution on $$m_1$$ (represented by a derived fact node $$d$$). Hence, there will be edges from $$p$$ to $$e$$ and from $$d$$ to $$e$$. In addition, if using exploits $$e$$ results in obtaining code execution privileges on $$m_2$$, represented by a derived fact node $$c$$, then there will be an edge from $$e$$ to $$c$$.

The labeling function maps a fact node to the fact it represents, and an action node to a rule that defines the derivation of new facts. Formally, for every action node $$a$$, let C be $$a$$’s child node and P be the set of $$a$$’s parent nodes, then

$$\forall a \in G \Rightarrow L(P) \Rightarrow L(C)$$

is an instantiation of interaction rule $$L(a)$$. [Ou et al., 2006] LAGs are a special case of And/Or Graphs [De Mello and Sanderson, 1990] where each action can instantiate only one fact (or derived fact). We will use this notation from [Gefen and Brafman, 2012]

- $$\text{pre}(a) = \{v \in N_p \cup N_c : (v, a) \in E\}$$
- $$\text{add}(a) = \{v \in N_p : (a, v) \in E\}$$
- $$\text{ach}(v) = \{a \in N_c : v \in \text{add}(a)\}$$

Where $$\text{pre}(a)$$ is the set of facts which are preconditions to the action $$a$$. $$\text{add}(a)$$ is the set of facts gained by applying the action $$a$$ (in LAGs this set contains only one node). $$\text{ach}(v)$$ is set of actions which can achieve derived fact node $$v$$.

An attack plan $$G'$$ is a subgraph of $$G$$. The attack plan must hold the following:

- $$g \in G'$$
- $$\forall a \in G'_{N_c} : \text{pre}_{G}(a) \subseteq G'$$
- $$\forall v \in G'_{N_p} : \exists a \in \text{ach}_{G}(v)s.t.a \in G' \land |\text{ach}_{G}(v)| = 1$$

Meaning, an attack plan is a sub-graph of $$G'$$ that contains the goal node of graph $$G$$. Each action $$a$$ in $$G'$$ is fulfilled by all of the preconditions of $$a$$ in $$G$$. Each fact is achieved by exactly one action. Attack plan represents a scenario in which an attacker infiltrates the organization and achieves his goals.

![Figure 1: Example of an attack graph](image-url)
2.2 Graph Centrality Measures

Graph centrality is a sub-field in graph theory research. Centrality measures try to capture how important a node is within a graph. This is useful in many domains, such as social network (finding prominent members in social networks) and more.

Previous research on ranking vulnerability fixes has used some centrality measures in order to identify which vulnerabilities should be fixed [Hong and Kim, 2013] [Sawil and Ou, 2008] following is a list of such measures that were previously used to that effect.

One of the basic centrality measure in graphs is the degree centrality. mainly because it is easy to compute. Although it is easy to compute (\(O(1)\)), degree centrality often poorly represents the true importance of a node in a graph.

\[ C_D(v) = \text{degree}(v) \]

Betweeness centrality captures a more delicate aspect of the importance of the node in a given graph. This measure represents for each node, the number of shortest paths between any two nodes that passes through that node. Formally, betweeness is:

\[ C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \]

Where \( \sigma_{st} \) is the number of shortest paths between nodes \( s \) and \( t \), and \( \sigma_{st}(v) \) is the number of shortest paths between nodes \( s \) and \( t \) that pass through node \( v \). It was used in few researches aiming to find important nodes in attack graphs [Hong and Kim, 2013]. The \( C_B \) values of the nodes can be computed using [Brandes, 2001] in \( O(|V||E|) \).

We have also used a variation of betweenness centrality in which only paths starting from a subset of the nodes in the graph, or ending in a subset of nodes in the graph were counted.

Another commonly used graph centrality measure is the Closeness Centrality. This centrality measure captures how close a certain node is to the rest of the nodes in the graph. In this centrality method, nodes on the fringe of the graphs should score lower than nodes in the center of the graph. Formally closeness centrality is defined by:

\[ C_C(v) = \frac{1}{\sum_u d(u, v)} \]

Where \( d(u, v) \) is the shortest distance between \( u \) and \( v \). The running time for finding \( C_C \) is \( O(nm + n^2 \log(n)) \) where \( n \) is the number of nodes in the graph, and \( m \) is the number of edges in the graph [Wang, 2006].

Reasearches also used Google’s PageRank to rank important nodes in a graph. Initially used to rank the importance of web pages in the Internet, PageRank’s essence is measuring how likely for a web-surfer to be at page \( j \) [Page et al., 1999]. \( 0 < d < 1 \) is a damping factor, representing how likely a web surfer will get bored and move to another web page which is not directly linked to the current node.

The metric is given by:

\[ C_{PR} = \frac{1 - d}{N} + d \sum_{j \in \text{Out}(j)} \frac{\pi_j}{|\text{Out}(j)|} \]

Where \( N \) is the number of nodes in the graph, \( \text{Out}(j) \) are the outgoing neighbors of \( j \), \( \text{In}(j) \) are the ingoing neighbors of \( j \), and \( \pi_j \) is the probability that the web-surfer will be at nodes \( j \). Fast, distributed algorithms for approximation of the PR values exists [Sarma et al., 2015]. This algorithm runs in \( O(\frac{\log(n)}{\epsilon}) \) rounds. where \( n \) is the number of nodes in the network and \( \epsilon \) is the damping factor.

2.3 Planning Graphs

Planning graphs [Blum and Furst, 1997] are a data structure from the automated classical planning community. A planning graph is a directed, layered graph with two types of nodes and two kinds of edges. The layers change between fact layers, containing only fact nodes, and action layers containing action nodes.

In general planning problems, already obtained facts can be removed by other actions. Planning graphs hence include additional information, such as which facts cannot be achieved at the same time (mutexes). However, in pentesting, once a fact is obtained, it is never lost. We can hence focus on the relaxed planning graph, where obtained facts cannot be lost, which is much simpler to represent and reason about.

The first layer of the relaxed planning graph is a fact layer, and contains one node for each condition node \( c \in C_c \). The next layer is an action layer, containing all actions that can be executed using the facts at the previous layer. That is, all actions whose preconditions appear in the previous layer. The third layer contains all the effects of the actions at the second layer.

In addition, we add for each fact \( p \) a special no-op action, that takes \( p \) as precondition, and generates \( p \) as output. Hence, each fact layer is a superset of the preceding fact layer. Once no new facts have been obtained in a fact layer, we can stop the expansion of the planning graph.

Edges in a planning graph represent relations between actions and facts. The action nodes in action-layer \( i \) are connected by “precondition-edges” to their preconditions in fact layer \( i \). The action nodes are also connected to their add-effects facts in layer \( i + 1 \) by “add-edges”.

Action nodes may exist at layer \( i \) only if all of their preconditions exist at fact layer \( i \). A fact may exist at fact-layer \( i + 1 \) if it is an effect of some action in action layer \( i \). Thus, the planning graph avoids cycles by allowing repeated fact and action nodes at different layers.
Facts often appear in multiple layers in the planning graph — once a fact has appeared at layer $i$, it will appear in all fact layers $j > i$. We denote each fact by its layer, that is, for fact $p$ at layer $i$, we write $p_i$.

Figure 3 shows a planning graph for the graph $G$ presented in Figure 1. We omit some of the edges between the facts and "no-op" actions for ease of presentation.

![Planning Graph of Graph G](image)

To conclude, planning graphs in delete-free domains capture the same information as an attack graph, in a slightly different format. Below, we suggest analyzing the planning graph, replacing the standard centrality measures with an analysis of all attack plans.

### 3 Enumerating All Attack Plans

We now explain how one can use the planning graph in order to enumerate all possible attack plans. Once we have enumerated all possible attacks, we can, e.g., identify fact or action nodes that participate in many such plans, which may be good candidates for an early fix.

We analyze the planning graph, rather than the LAG, because LAGs contain cycles, which are avoided in the planning graph by using repeated nodes. We use a BFS-style algorithm, moving backward from the goal node $g_n$ at the last fact layer $n$.

We maintain a set of plans. For each plan there is a set of unsatisfied facts, initialized with the goal. To expand a plan backwards from layer $i$, for each unsatisfied fact $p_{i+2}$, we identify an action $a$ (possibly a no-op) that has $p$ in its effects. We remove $p_i$ from the list of unsatisfied facts, and for each fact $q$ in the preconditions of $a$ we add $q_i$ to the set of unsatisfied facts. If a provides an additional unsatisfied fact $r$, it is also removed from the list of unsatisfied facts. That is, we will not search for another action $a'$ to satisfy $r_i$.

There can be many potential actions that satisfy a needed fact $p$, each corresponding to a different plan. Thus, for each action $a$ that satisfy $p$ we create a copy of the plan and add $a$ to the copy. Thus, the expanded plan is split into multiple identical plans, differing on the last added action only.

More precisely, let $P_i$ be the set of unsatisfied facts of the expanded plan at layer $i$, and $A_{i-1}^F = \{a : \exists p \in P_i, p \in effects(a)\}$ be the set of actions at layer $i - 1$ that satisfy at least one fact in $P_i$. We create a copy of the plan for each minimal subset $A_{i-1}^F \subseteq A_{i-1}^F$ such that $P_i = \bigcup_{a \in A_{i-1}^F} effects(a)$, and add $A_{i-1}^F$ to the copy.

Algorithm 1: Enumerating All Attack Plans

```plaintext
EnumeratePlans(PG, t):
Input: Planning graph PG, target node t
Output: Set of all the attack plans in the graph
1 $P \leftarrow \{t\}$; // Solution plans
2 cur_layer $\leftarrow$ lastLayer(PG);
3 while cur_layer $\neq$ t do
4   if cur_layer.type = action then
5     for $p \in P$ do
6       t $\leftarrow$ $t$ + $a$; predecessors;
7     end
8   end
9   else if cur_layer.type = fact then
10      for $p \in P$ do
11         $P \leftarrow P \cup \{p \cup a\};$ predecessors;
12      end
13   end
14 return cur_layer - 1;
end
```

Once we have reached the initial layer we have enumerated all possible plans. Let II be the set of all such plans. II may contain some redundancies, due to the use of no-ops. More specifically, given $P_i = \{p_i, q_i\}$, and two action $a_p, a_q$ that produce $p, q$, respectively, we may have 4 different alternatives — $(a_p, a_q)$, $(a_p, noop_q)$, $(noop_p, a_q)$, and $(noop_p, noop_q)$ for expanding the plan backwards. Then, at layer $i - 2$, we can choose $a_p$ where $noop_p$ was selected and $a_q$ where $noop_q$ was selected. Ignoring the no-ops, which are not real actions to be executed, we obtain 4 identical plans. To remove such duplicates, once we have obtained the set of all plans, we remove no-ops from all plans, and then remove duplicate plans, ignoring the action order within a plan.

Using the above planning graph construction and plan enumeration method only yields plans with bound number of actions (which is the number of action layers in the planning graph). In order to allow plans in various lengths, additional edges should be added to the planning graph between the final fact layer and the final actions layer. At this point we have chosen to use only the plans with the shortest length, and not allow plans with larger amount of actions that needed.

This process is obviously $np$ hard, but in the real world graph that we have obtained, it runs sufficiently fast to be useful. Creating the planning graph and enumerating the plans took less than a second on both graphs Table. 1. The ex-
peersments where performed on a Virtual Machine using one Intel Xenon E5-2620 v2 @ 2.10 GHz processor, with 8 GB of RAM. In the future, we will explore sampling techniques, originating from research in AND-OR graphs, to provide a rapid estimation of the needed statistics.

Using the set of all plans we can compute useful measurements. For example, we can count for each action \(a\) the number of plans in which \(a\) participates:

\[
C_\Pi(a) = |\{\pi : \pi \in \Pi, a \in \pi\}|
\]

We can also compute this over a subset of plans, such as only over the set of plans of length at most \(k\), thus identifying actions that appear in shorter, more efficient, plans.

### 4 Network Data Acquisition

To test our approach we created realistic models using data obtained from scanning the network of a large organization, containing several subnets. Using the machine configurations and existing exploits discovered using the scan, we can create real world models that allow us to provide an empirical evaluation of our approach. We now provide some explanations of the model and the network, unfortunately omitting many details due to confidentiality restrictions.

To collect the needed information for our models, we began by running a scan of the various subnets using the Nessus scanner [Beale et al., 2004]. Nessus starts its scan from a given computer, and identifies all reachable hosts from that computer, including desktops, gateways, switches, and more.

As Nessus does not actually launch attacks to control a host, a Nessus scan identifies only hosts that are directly logically reachable from the source machine where the scan is running, possibly through several switches and gateways. We hence executed several such scans, each from a different subnet within the organization, as well as one scan from outside the organization network.

The resulting scans contain the set of machines that are visible from each source machine. The machines inside a subnet are all visible to each other. Hence, we assume that all machines within a subnet can directly access the machines that the representative source machine can access. Only a part of the machines outside the subnet are visible from within the subnet, due, e.g., to firewall restrictions. We model the accessibility of machines identified through the scans as direct edges in the network graph. That is, machine \(m_1\) is connected in our model to machine \(m_2\), if \(m_2\) is visible from \(m_1\) or vice versa.

In addition, Nessus reveals for each identified host its operating system. The network contained hosts running Windows and Linux (with a few versions of each operating system). Nessus also identifies softwares with potential vulnerabilities that run on the machines. Our model contains about 50 such software, including well known applications such as openssh, tomcat, pcanywhere, ftp services, and many more.

Nessus finds vulnerabilities of varying importance. For the purpose of this experiment we ignored all the lesser vulnerabilities, which do not allow an attacker control of the system. We remain with about 60 serious types of vulnerabilities that exist in the network. We remove from the network all hosts that do not run any software for which a serious vulnerability exists, remaining with about 23 hosts.

For constructing our attack goal, we took six random hosts from the innermost subnet, and set them as the target hosts. The problem goal is to gain control over one of these six hosts.

![Figure 4: Connectivity graph of a network. Each node is a host computer, a directed edge between two hosts \((u, v)\) means host \(u\) can initiate a connection to node \(v\)](image)

### 5 Evaluation

We now compare the utility of various graph centrality measures in ranking the set of possible machine vulnerabilities to be fixed. In this paper we focus on the task of increasing the cost of the minimal attack plan. That is, we use the various metrics computed over the LAG or the planning graph to rank the vulnerabilities to be fixed, and check which ranking induces an increase in the minimal attack plan cost using less fixes.

#### 5.1 Domains

We experiment with a benchmark network from previous research [Hoffmann, 2015] [Durkota et al., 2015], as well as the network of the large organization that we scanned. The network scanned contained 23 hosts including a host representing the Internet. The hosts had 144 critical vulnerabilities which an attacker could leverage. The dataset from the literature - LocalPlus-20 originally contained 23 hosts, we added 3 more hosts and slightly altered it’s connectivity graph. In the end the graph contained 26 hosts and 26 vulnerabilities. The statistics about the LAG produced from the above networks are presented in Table 2.

We have searched for additional publicly available networks and found none. We also explored additional simulated benchmarks, but these presented very artificial networks (e.g.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Nodes</th>
<th>Plans</th>
<th>Time(Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>localPlus</td>
<td>394</td>
<td>48</td>
<td>0.04</td>
</tr>
<tr>
<td>ScannedNetwork</td>
<td>1013</td>
<td>2012</td>
<td>0.52</td>
</tr>
</tbody>
</table>

Table 1: Enumeration running time in respect to LAG size
where each machine had only a single vulnerability), and the results over these networks were uninteresting.

5.2 Methods
We compare here the following metrics:
1. Plan count
2. Betweenness
3. PageRank
4. Closeness
5. Random ranking

Plan count is the number of shortest plans in which a vulnerability participates. This is computed using the plan enumeration procedure. Vulnerabilities are ranked by decreasing number of plans in which they participate.

All metrics are computed over both the LAG and the planning graph, except for the plan count, which is computed only over the planning graph. As a node in the LAG can appear multiple times in the planning graph (for instance between no-op actions, e.g: $p \rightarrow \text{noop} \rightarrow p$), we count the different appearances of a node in the planning graph.

5.3 Procedure
We performed the experiments in the following manner; For each centrality method we begin with the original graph (LAG or planning graph), and compute the metric for all vulnerability nodes in the graph. Then we rank all the nodes according to the centrality method by decreasing value.

We select the node with the highest centrality measure to be fixed first. The vulnerability corresponding to this node is now removed, and we recompute the LAG or the planning graph without this node. Then, we recompute the metric over the new, revised, graph.

In addition, we enumerate the set of attack plans over the original planning graph using Algorithm 1. We identify the subset of plans with the minimal cost (shortest plans). Whenever we remove a vulnerability following the above procedure, we also remove all the shortest plans that use this vulnerability. We end when there are no more shortest plans left.

5.4 Results
Figure 5a presents the reduction in the number of shortest attack plans after every patch (removal of a vulnerability on a specific host) on the simulated Local+20 benchmark. On this network, the only two metrics that supply any useful information are the shortest plan count, and Betweenness over the planning graph. Both methods allow the administrator to remove all shortest attack plans after patching only 4 vulnerabilities. All other centrality metrics do not perform better than a random ranking.

6 Related Work

Attack Graphs have been used to depict possible ways for an attacker to compromise a computer network. Almost two decades ago DARPA created attack graph manually as part of red-team analysis. Initially, attack graphs were used to better visualize the paths an attacker can take in the network. Once attack graph could easily be generated automatically, researches have used them to improve the security of the networks. They did so by a number of different methods. We presented works that can help determine what vulnerabilities to patch in an organizational network. Additional works exist [Durkota et al., 2015] to help determine countermeasure placements in the network (like IPS or Honeypots) but they are out of scope for this work.

6.1 Finding Optimal Attack Plans

Many researches have assumed some metric on actions in the attack graph [Obes et al., 2013]. The metrics usually represent cost, like the time it takes to launch exploit, risk of detection and so on. Another common metric for actions is the probability of success when performing the action. [Wang et al., 2008] Researchers then tried to find attack plans which minimize/maximize the suggested metric. The assumption is that rational attacker will first try to launch attacks that minimize the cost for the attacker. The downside of many of those models are simplifying assumptions on the attacker, which are not always realistic in the real world. Example of those assumptions are the fact the attacker needs to know beforehand the structure of the network, or that he cannot change the structure of the network. When trying to relax those assumptions, using Contingency Planning, MDP or POMDP to achieve realistic results, the runtime for deriving conclusions makes it not practical for real-size networks [Hoffmann, 2015] [Shmaryahu, 2016]. More over, test shows [Sommesstad and Sandström, 2015] that attack graphs do not always represent all the actual paths an attacker can take. So trying to eliminate numerous plans in respect to some metric might not always prevent an attacker from achieving his goals.

6.2 Denying Access to the Goal
Another common use of attack graphs is finding a set of conditions to patch which will prevent the attacker from reach-
ing the goal. Works in this area use various methods such as minimum-cost SAT solving [Huang et al., 2011] and specialized methods [Albanese et al., 2012] invented for this task. The down-side of this method is that in practice, even after finding minimal set of conditions to patch it may still contain a significant amount of vulnerabilities to patch, which will not be possible without significant IT resources. This leads to our goal to minimize the number of attack plans in the graph that could be used to reach the goal.

6.3 Patching to Minimize Paths to the Goal
In their work [Hong and Kim, 2013] [Hong et al., 2014] propose to use network centrality measures to find which vulnerabilities to patch first. To our understanding, this work proposes using a two-layer graph. The first layer, representing the hosts in the system. A directed edge between two nodes \((a, b)\) means that when controlling node \(a\), an attacker is able to advance to node \(b\) (using exploit, or a similar manner). The second layer is an AND-OR tree containing all the ways to compromise a machine from an arbitrary other machine. The authors then compute different network centrality measures on the first layer to find which vulnerabilities to patch. To our understanding, the absence of the knowledge on how to gain access to a host (which is depicted in layer two) when computing centrality measures on the first layer, can often yield sub-optimal results.

7 Conclusion and Future Work
We discuss in this paper metrics for ranking the vulnerabilities to patch in a computer network. We focus on the problem of increasing the cost of the shortest attack plan. We show that metrics computed over the planning graph — a data structure from automated planning — provide much better rankings. In addition, the planning graph allows us to enumerate the set of shortest plans, providing a new ranking metric based on vulnerability appearance in shortest plans.

We experiment with a real world network, which we defined using a scan of a large organization computer network. As such, this is one of the first papers to report results over an attack graph of a real network. We also experimented with a standard simulated benchmark. It is interesting to see that the results over the simulated network are very different than results over the real network, emphasizing the urgent need for
additional real world networks for experiments.

In the future we intend to experiment with additional interesting real world networks. We would also explore additional optimization problems, such as removing all identified attack plans. The plan enumeration procedure that we used has an exponential complexity, and we intend to explore sampling methods to allow for rapid estimation of the set of shortest plans.

References


Planning the Attack! Or How to use AI in Security Testing?

Josip Bozic and Franz Wotawa∗
Institute for Software Technology
Graz University of Technology
A-8010 Graz, Austria
{jbozic,wotawa}@ist.tugraz.at

Abstract
Testing is one effective method for quality assurance. Generating and executing tests is a labor consuming task and there has been a lot of effort spent in test automation where the focus has been mainly on functional or penetration testing but not specifically on security testing. In this paper, we discuss two already introduced approaches for automated security testing that are based on AI planning. The approaches map attack models and security protocol definitions to AI planning problems in order to generate test cases. Furthermore, utilizing plan execution together with generated plans allows also for automating the test execution. The objective of the paper is to further stimulate research in this field. Thus we not only discuss the foundations behind and their applications, but also outline challenges and further research directions.

1 Introduction
Our modern society relies more and more on communication infrastructure. Its importance increases due to the availability of services like e-government, online banking, online shops, or social media applications that are used more or less on an everyday basis. All these applications including their communication backbone need to be reliable and secure preventing malicious entities from harming us, e.g., stealing money from our bank accounts or gaining access to our private data. Although, the security issue has gained a lot of attention over the past decades, our systems including infrastructure are still vulnerable against attacks. One reason behind is definitely the system’s complexity comprising interacting heterogenous systems where it is very difficult to assure no vulnerabilities that an attacker can exploit during an attack. Vulnerability prevention – for example during programming – is, therefore, an important topic. [Hoglund and McGraw, 2004] give a very nice introduction into how attackers identify and make use of weaknesses in software in order to come up with an exploit. One challenge behind preventing from vulnerabilities is that an attack may utilize not only one weakness but a combination of shortcomings. Hence, identifying all vulnerabilities would require finding arbitrary sequences of interactions with a system that can be used for an exploit.

Focusing on finding new vulnerabilities is an important topic of security where the objective is to identify the vulnerability before an attacker can make use of it. This fight between the attacker and the system’s defenders may never stop but it would still be an advantage to avoid known vulnerabilities to be in new systems. It is astonishing that this is not the case. When looking at the top ten list of OWASP1 over the past 4 years, the first three attacks, i.e., injection, broken authentication and session management, and cross-site scripting (XSS), have not changed. Hence, it seems that it is necessary not only to focus on new vulnerabilities but to make sure that known ones are detected before a new software is deployed. This observation has been our main motivation behind our security testing research activities of the last 5 years, where we have been working on applying AI planning to security testing with the aim of automating security testing for avoiding known vulnerabilities in new system releases.

So, why relying on AI planning? When looking at attack description it is obvious that an attack comprises a sequence of steps to be carried out. For example, when performing an XSS over the web, an attacker has to access a web page, obtain the text fields, and afterwards, try to communicate source code to the server using these text fields. When introducing malicious code the attacker may try different program language fragments in order to identify the one that is used on side of the server. A sequence of such activities is obviously close to a plan, which comprises a set of actions each having pre-conditions and effects. This is very much similar to attack activities, where a certain activity can only be carried out when its pre-condition is fulfilled leading to its effect when being executed.

In this paper, we report on our work on planning for security testing. In particular, we show how AI planning can be used for modeling attacks thus allowing to automate security testing in detail. We further discuss two application areas. One is the domain of web applications and the other one is security protocol testing. Although, the main focus of the work is on finding known vulnerabilities it might be the case that AI planning is also capable of identifying unknown vulnerabilities in new system releases.

∗ Authors are listed in alphabetical order.

ones. This might be the case when specifying a lot of actions and using a planner generating arbitrary plans.

This paper is organized as follows. First, the applications and related research are discussed in Section 2 where we consider other research papers dealing with the use of AI planning for testing. In particular, we recall approaches dealing with pre- and post-conditions of methods to be used for test data generation, and using planning in the context of GUI testing. In Section 3 there is a discussion of planning and its use in security testing. Afterwards, the planning model is described in great detail in Section 4. The case study in Section 5 visualizes the approach on an example. Finally, we conclude the paper and discuss open challenges in Section 6.

2 Related Work

Planning, i.e., searching for actions that lead from a goal state to a final state, has been an active research field since the beginnings of AI. [Fikes and Nilsson, 1971] introduced the planning problem comprising an initial state, a goal state, and actions, and planning methodology STRIPS for generating sequences of actions for a planning problem. [Howe et al., 1997] are one of the first that utilize the planning problem for testing. In their paper, the authors describe test case generation as a planning problem. The introduced the system Sleuth relies on the planner UCPOP 2.0. [Howe et al., 1997] also explain the process of different plan generations according to changing states in the specification.

[Leitner and Bloem, 2005] introduce a planning-based testing strategy combining planning with a learning-based algorithm to generate plans at runtime. [Leitner and Bloem, 2005] suggested first, to extract a simplified model from a system under test (SUT). Afterwards, the planning problem is inferred from the generated model and a planner generates a weak solution, which represents a test case. The interpreter executes the test and the traversed state transitions are recorded. In case the execution leads to the specified goal state, the routine can be tested. Otherwise, the state transitions are included into the problem specification and a new plan is generated according to stronger pre-conditions. In this way, the resulting new test case is supposed to bring the test execution closer to success. The main difference to our approach is the absence of problem specification inference and the learning strategy. On the contrary, in both cases different plans are generated according to dynamic changes in the specification.

[Memon et al., 2000b] and [Memon et al., 2000a] introduce PATHS, an automated GUI testing system that relies on a planning. The paper puts emphasis on the test case generation part of the system. The specification of the planning problem is based on GUI applications, whereas the system relies upon the planner IPP [Koehler et al., 1997], an extension of [Blum and Furst, 1997]. Here a planner generates multiple partial-order plans that are executed, eventually leading from an initial to the goal state. The test case generation process is divided into two phases, respectively the setup and the plan-generation phase. In the first part, PATHS makes an abstract model of the GUI and extracts GUI events, which represent abstract operators. Then the tester assigns corresponding pre- and post-conditions to these operators. Afterwards, in the plan generation phase, the initial and goal states are specified. Finally, PATHS generates the test cases that are executed on the concrete level. When an error is detected, the execution of the plan terminates. In this case, test oracles are used after each testing step. An important feature of this approach is the generation of alternative plans. This is done by creating sub-plans that represent a decomposition of an abstract operator from the high-level plan. The authors elaborate their algorithm for test case generation and present some experiments, which were conducted on WordPad. In addition to test case generation, PATHS also deals with test oracles and regression testing. The main difference to our approach, besides the application domain, is the generation of sub-plans, i.e. the generation of plans at two levels. On the other hand, the concept of using events for the action specification is done for protocol testing as well. But instead of using GUI events, we apply the TLS event structure from the protocol specification.

[Armando et al., 2010] introduced an adaptation of planning to testing of security protocols. Other papers that deal with planning-based testing include [Galler et al., 2010] and [Schneel and Guidali, 2010]. [Beurdouche et al., 2015b] introduce FlexTLS, a testing framework for TLS. Besides emulating the processing of TLS events, the implementation offers the possibility to manipulate the concrete parameter values and sequence of events. Some other works that address testing of protocols include, but are not restricted to [de Ruiter and Poll, 2015; Mavrogianopoulos et al., 2012; Morais et al., 2009] and [AlFardan and Paterson, 2013].

3 Planning in Security Testing

The basic motivation behind the adaptation of planning is the fact that every interaction between a client and a system can be represented as sequence of actions. An interaction consists of multiple states, where each of the states is connected with another state when applying an action. Every action has some specific pre-condition that has to be fulfilled in order to be triggered. When triggered, the execution of this action leads into a new state. If this concept is put into the context of security testing, then every attack against a system can be seen as a sequence of actions as well. In the real world, an attacker has to carry out a set of malicious interactions with the victim system in order to cause some damage or retrieve data. Thus, the actions are specified formally in a way that they should resemble the individual attack steps undertaken by the attacker. Then, a planner generates a sequence of actions, i.e. a plan, which represents a full attack and, as such, a test case. In fact, this depiction acts as a blueprint for an attack. By automating the test case generation and execution, the attacker is emulated in an iterative manner. An initial state defines the starting point from where the attack is started. On the other side, the final state specifies the condition where an attack has been successfully carried out. The result of a test case can be either a FAIL or PASS. If an attack was successful, the tester can track the set of actions and the corresponding values that lead to the vulnerability detection. Afterwards, steps can be taken in order to cover the issue. We take the following un-
derlying definitions of the planning problem from [Bozic et al., 2017]:

**Definition 1** The planning problem is a quadruple \((P, I, G, A)\) where \(P\) denotes the set of predicates, \(I\) the initial state, \(G\) the goal state and \(A\) the set of actions. A predicate \(p \in P\) presents a first order logic formula and acts as a condition in an action definition. An action \(a \in A\) is defined by its parameters, set of pre-conditions and effects. The corresponding functions \(\text{pre}(a)\) and \(\text{post}(a)\) consist of predicates from \(P\). The initial and goal states are defined with predicates that are true in these states.

The plan generation starts from the initial state \(I\) that is given by some initial values. Then, the set of action is traversed in order to check whether a pre-conditions is satisfied by the initial configuration. In general, if some predicate of an action \(a\) in a state \(S\) is valid in \(\text{pre}(a)\), then \(\text{post}(a)\) will lead the execution into the next state \(S'\), i.e. \(S \xrightarrow{a} S'\).

**Definition 2** A plan is a sequence of actions \((a_1, \ldots, a_n)\) and presents a solution to a planning problem \((P, I, G, A)\) so that \(I \xrightarrow{a_1} S_1 \xrightarrow{a_2} \ldots \xrightarrow{a_{n-1}} S_{n-1} \xrightarrow{a_n} G\).

From the implementation point of view, we assume specifying the planning problem using two files, namely the domain description file and the planning problem file. We also assume that the specification is done using the planning language PDDL [McDermott et al., 1998]. The overall implementation can be divided into two parts: First, the type of the SUT is chosen, which in our case is either a web application or a security protocol. Then, according to the analysis of the structure of the system, an insight is obtained about its (mal-)functionality. Afterwards, descriptions of known system-related attacks are analyzed and an attack model is crafted according to inferred information. Whereas in some of the previous works (e.g. [Bozic and Wotawa, 2014b]) this is realized in form of UML state machines, we construct a planning model. Figure 1 depicts this procedure. This model depicts all necessary data definitions in order for the planner to generate a plan. It should be noted that the plan by itself represents only an abstract test case. Second, an execution framework reads the plan and executes it against a concrete SUT. During execution, concrete values are assigned to the parameters of the individual actions. This represents the concretization phase of the approach. Since the plan depicts a way to a successful security breach, if the execution reaches the goal state, it is assumed that a vulnerability has been detected. In case that an execution does not reach the final state, a new plan is generated with a different sequence of actions. Finally, the overall execution terminates after the last test case has been carried out.

4 Modeling the Planning Specification

When using planning for security testing, the construction and adaptation of the model depends heavily upon the type of the SUT and the attacks to be consider like SQL injection (SQLI) and cross-site scripting (XSS) in case of web applications and man in the middle attacks (MITM) in case of security protocol SSL/TLS. Both applications will be explained below in greater detail. Although the resulting model differs in both case studies, the methodology is the same in both cases. This leads to the assumption that the proposed approach can be adapted to testing of other systems and attack types as well. [Bozic and Wotawa, 2014a] and [Bozic and Wotawa, 2015] explained the planning of protocol testing. 

4.1 Web Applications

SQLI is for many years the most common injection attack on web applications (see Footnote 1). This attack targets the database behind a web application. So basically every implementation on the internet could be addressed. Despite sophisticated prevention mechanisms the attack still represents a major threat. Therefore, automating the testing process for the detection of this type of vulnerability still constitutes an important task.

First, let’s take a look at a typical injection scenario. Usually an application sends SQL queries to the database, thus retrieving, deleting or manipulating its content. SQL by itself has a valid syntax that is submitted to the database. If the user gives the instruction to the implementation for accessing the data, where the number of possible access commands is usually restricted by the application. A simple query does look like:

```
SELECT * FROM workers WHERE password = "+pField+
```

This command would usually retrieve data from the table `workers` that matches the password. However, an attacker could trick the application to access the database even if s/he does not have a valid password. For example, s/he could insert the malicious SQL code into the input field:

```
x' OR 1 = 1;--
```

In this case, the executed query will look like:
SELECT * FROM workers WHERE password = "x' OR 1 = 1;-- "

Since the statement OR 1=1 will always make the query to be true, this would result in obtaining of potentially sensible data.

The injected malicious code represents a concrete value, which is meant for one test case.

As already mentioned, it’s necessary to specify the planning problem via two different PDDL files. The domain encompasses data that is present in every problem. The necessary data includes, among others, the definition of types, predicates and actions. A reduced domain for a SQLI model is shown below.

```
(define (domain sqli-domain)
  (:requirements :strips :typing :fluents :adl)
  (:types
   active
   address
   status-lo
   status-si
   status-se)
  (:constants
   init - active
   no yes - status-lo
   sqli rxss sxss - type)
  (:predicates
   (inInitial ?x)
   (Empty ?url)
   (inAddressed ?x)
   (inSentReq ?x)
   (Logged ?lo)
   (statusinit ?si))
  (:functions
   (Logged ?lo - status-lo)
   (sent ?se - status-se)
   (statusinit ?si - status-si))
  (:action Start
    :parameters(?
x - active
    ?url - address
    ?lo - status-lo)
    :precondition (and
      (inInitial ?x)
      (not (Empty ?url)))
    :effect (and
      (inAddressed ?x)
      (not (inInitial ?x))
      (Logged yes)))
  (:action SendReq
    :parameters(?
x - active
    ?lo - status-lo
    ?se - status-se
    ?si - status-si
    ?lo - status-lo)
    :precondition (and
      (inAddressed ?x))
    :effect (and
      (Logged yes))
    :effect (and
      (inSentReq ?x)
      (not (inAddressed ?x))
      (assign (sent ?se) 1)
      (statusinit two)))
```

Domain description for SQLI

As can be seen on this small sample, the individual actions and its parameters are modeled on the HTTP protocol between a client and the server. While the predicates indicate the current status of an object, functions are responsible for manipulation of data. For example, in order to start the testing process, an URL address needs to be given. An excerpt from the corresponding problem file reads as follows.

```
(define (problem sqli-problem)
  (:domain sqli-domain)
  (:objects
   x - active
   si - status-si
   lo - status-lo
   se - status-se
   url - address)
  (:init
   (inInitial x)
   (Logged no)
   (not (statusinit two))
   (= (sent se) 0)
   (not (Empty url)))
  (:goal
   (inFinal x)))
```

Problem description for SQLI

The problem file instantiates the objects of a type that has been defined in the domain. Here the initial and goal states can be found as well. Whereas the initial one defines the starting point, e.g. the idle state in execution, the final counterpart indicates a vulnerability breach according to the test oracle. The plan generation depends heavily upon these two definitions. If the initial values are changed, either manually by the tester or dynamically during execution, a different plan may be generated. Such a generated plan is depicted below.

```
0: START X URL LO
1: SENDREQ X LO SE SI
2: RECREQ X SI
3: PARSE X M USERNAME PASSWORD TYPE
4: CHOOSESQLI X TYPE
5: ATTACKSQLI X SQLI M UN PW EXP
6: RECEIVERESP X RESP
7: PARSERESP SQL X EXP RESP
8: PARSERESP SQLCHECK X EXP RESP
9: FINISH X
```

Plan for SQLI

If the final state cannot be reached by any arrangement of action, then no plan will be generated. In the proposed ap-
proach the planning system Metric-FF\(^2\) is used, although the results can be obtained with other planners like Fast Downward\(^3\) or LPG\(^4\).

### 4.2 SSL/TLS Protocol

Security protocols play an important role in ensuring of a secure communication channel between client and server. One of such protocols is TLS and its predecessor, SSL. A private channel is established between the peers by applying a handshake procedure. Here encryption keys are exchanged, after which the data can be exchanged between both sides.

The current TLS standard [Dierks and Rescorla, 2008] defines the message flow and data structure of TLS events in a handshake procedure. There is a set of TLS event types with its own parameters that are exchanged between two peers. The procedure begins with the ClientHello message and finalizes with the server’s Finished message.

The current version of TLS is version 1.2 but despite its security measures, the protocol still proves to be vulnerable to attacks. Only recently new vulnerabilities like Heartbleed\(^5\) and DROWN [Aviram et al., 2016] have been discovered. This leads to the conclusion that various TLS implementations, e.g. OpenSSL\(^6\) or GnuTLS\(^7\), still need to be tested against known attacks to check whether they are secure.

In order to specify protocol security testing as a planning problem, necessary information has to be obtained from the TLS standard. The proposed approach targets the handshake procedure, because many of the known security leaks are found there. Here individual TLS events are specified as actions in the planning problem. Here the example from [Bozic et al., 2017] is taken, where the initial ClientHello message does usually have the following structure.

---

**Protocol Version:** TLS12  
**Client Random:** 90 68 ...  
**Session ID:**  
**Supported Cipher Suites:** 00 30  
**Supported Compression Methods:** 01

This message contains a set of variables, like the protocol version etc., with corresponding concrete values. In order to specify the problem in the planning way, the structure of the event is adapted to the object-type hierarchy of PDDL. The resulting action encompasses all of the event’s original parameters, whereas the tester defines the pre- and postconditions. The resulting action is shown below.

```plaintext
(:action ClientHello  
  :parameters(  
    ?pv - ProtocolVersion  
    ?crandom - ClientRandom  
    ?sid - SessionID  
    ?ccs - ClientCipherSuite  
    ?ccm - ClientCompressionMethod  
    ?x - active)  
  :precondition (inInitial ?x)  
  :effect (and (inClientHelloSent ?x)  
            (not (inInitial ?x))) )
```

---

Basically, this mapping between TLS events and PDDL’s action is assigned to the other messages as well. In this way, the plan execution may resemble the usual message flow as specified in the standard. An excerpt of the domain specification, which encompasses the client’s initial message is given below.

```plaintext
(define (domain tls-domain)  
 (:requirements :strips :typing :equality :adl)  
 (:types  
   active  
   Priority  
   ProtocolVersion  
   ClientRandom  
   SessionID  
   ClientCipherSuite  
   ClientCompressionMethod)  
 (:constants  
   rsa dh - KeyAlg)  
 (:predicates  
   (inInitial ?x)  
   (inClientHelloSent ?x)  
   (inClientHandshakeFinished ?x)  
   (inFinal ?x)  
   (KEY ?key))  
 (:functions  
   (KEY ?key - KeyAlg))  
 (:action ClientHello  
  :parameters(  
    ?pv - ProtocolVersion  
    ?crandom - ClientRandom  
    ?sid - SessionID  
    ?ccs - ClientCipherSuite  
    ?ccm - ClientCompressionMethod  
    ?x - active)  
  :precondition (inInitial ?x)  
  :effect (and (inClientHelloSent ?x)  
            (not (inInitial ?x))) )
```

---

A piece of problem file with the definitions of objects and initial and goal states looks as follows.

```plaintext
(define (problem tls-problem)  
 (:domain tls-domain)  
 (:objects  
   x - active  
   pv - ProtocolVersion  
   crandom - ClientRandom
```

---

\(^2\)https://fai.cs.uni-saarland.de/hoffmann/metric-ff.html, accessed: 2016-12-02  
\(^3\)http://www.fast-downward.org/, accessed: 2017-13-03  
\(^4\)http://lpg.unibs.it/lpg/, accessed: 2017-13-03  
\(^5\)see http://heartbleed.com/, accessed: 2016-12-06  
\(^6\)https://www.openssl.org/, accessed: 2017-05-14  
\(^7\)https://www.gnutls.org/, accessed: 2017-05-14
Problem description for TLS

Finally, a planner generates a plan. As already mentioned, the depicted sequence of actions represents an abstract test case. Usually the next step will be concrete value assignment although some vulnerabilities, e.g. SKIP [Beurdouche et al., 2015a], can be triggered on the abstract level alone by relying only on default values.

Plan for TLS handshake

Plan for SKIP

The test oracle for this attack represents a check whether the client has received server’s Finished message, albeit previously omitted events. If the testing framework is able to execute all actions without triggering some unintended notification, it is assumed that an unencrypted communication channel is established. As notices, concrete values do not need to be assigned for this attack.

Although this depicts a simple example, it demonstrates the connection between the planning specification, test case generation and execution. Planning problems can be defined for every attack. Also, by changing the definition, unintended sequences of actions could lead to exposition of security leaks. Thus, negative testing can be realized in the planning way as well.

5 Case Study

Here a demonstration is given how to execute an attack according to its planning specification in case of the security protocol TLS. As a simple example the already SKIP attack is chosen. Attacks that target the protocol’s vulnerability usually happen when the attacker intercepts the communication channel between the peers. In this way she or he acts as a man-in-the-middle and takes the role of a client towards the server and vice versa. The vulnerability happens when the handshake procedure concludes with establishing an encryption-free communication channel. This should not happen because TLS is meant to enforce an encryption on exchanged data. However, in this scenario no notification is triggered and it is assumed that the handshake has been finalized in an appropriate manner.

The standard event flow, as depicted in Section 4.2, demands that after the server sends a certificate, a finishing message is generated. However, in this attack some obligatory messages and parameters are skipped, i.e. the server signature is omitted and no encryption is established. Finally, if the client accepts the server’s Finished message, data in an unencrypted form can be exchanged. Such situation offers an attacker the possibility to read data in plain text format. In order to generate such a test case, the planning specification has to be adapted.

Until now, the precondition of the sever’s Finished message has been the previous sending of its ChangeCipherSpec message. This means that a server sends its final message only after the client has received its mentioned message. So the precondition has been specified as:

:precondition(inServerCertificateSent ?x)

But since some of the messages have to be skipped in order for SKIP to succeed, the precondition of the final server-side message is changed to:

:precondition(inServerCCS ?x)

Then, the invoked plan will generate a small plan that looks like follows.

Plan for TLS handshake

Plan for SKIP

The test oracle for this attack represents a check whether the client has received server’s Finished message, albeit previously omitted events. If the testing framework is able to execute all actions without triggering some unintended notification, it is assumed that an unencrypted communication channel is established. As notices, concrete values do not need to be assigned for this attack.

Although this depicts a simple example, it demonstrates the connection between the planning specification, test case generation and execution. Planning problems can be defined for every attack. Also, by changing the definition, unintended sequences of actions could lead to exposition of security leaks. Thus, negative testing can be realized in the planning way as well.

6 Conclusion and Future Work

AI planning offers very certain means for representing attack models and security protocols and thus provides a very good bases for automating the security testing challenge. In particular, attack models can be very nicely represented as AI planning problem from which attacks can be obtained. In contrast to direct representation of attacks, for example, using a directed graph, the representation as AI planning problem may not only specify one attack but many. Every plan that can be obtained from the attack representation is itself a new attack. Hence, when combining different attack representations, we might also obtain new attacks that are combinations of already known attack scenarios. The same holds for security protocols where different kind of knowledge can be nicely integrated, e.g., knowledge about certain actions and actions that have to be performed because of the specification of the protocol.

In addition, to plan generation for obtaining abstract test cases, plan execution provides means for executing the security tests. For this purpose, the abstract plans have to be concretized. For example, instead of using a place holder of a user name or password, a real user name and password has to be used during execution. This mapping, however, can be
done easily and the whole concretization can be fully automated. One challenge that still remains to a certain extent is the test oracle problem. Identifying when a test case is failing and passing is not always that straightforward. Hence, further research has to be done for providing general solutions for the oracle problem in the context of security testing and AI planning.

Additional challenges are the granularity of actions and a more reactive plan execution. Granularity in this context means to identify the basic actions used to generate plans. Is it better to make an action login that submits user name and password to a server, or to set both of them separately before invoking a call? The granularity obviously depends on the application domain. However, when trying to integrate various attacks the actions must be reusable and easy to develop. Hence, there is a need for further investigating on the right representation of actions. Reactive plan execution is another weakness of the current implementation. In many cases a plan trying to exploit a known vulnerability cannot be executed. Whenever, the execution has to be terminated, it would sometimes be good not to start the whole planning and execution phase again but to restart at the current situation. A closer integration of plan generation and execution, maybe similar to teleo-reactive planning [Nilsson, 1994] might be appropriate.

References


DEEP SECURE: A FAST AND SIMPLE NEURAL NETWORK BASED APPROACH FOR USER AUTHENTICATION AND IDENTIFICATION VIA KEYSTROKE DYNAMICS: SAKET MAHESHWARY, SOUMYAJIT GANGULY AND VIKRAM PUDI
Deep Secure: A Fast and Simple Neural Network based approach for User Authentication and Identification via Keystroke Dynamics

Saket Maheshwary, Soumyajit Ganguly, Vikram Pudi
Data Sciences and Analytics Center, Kohli Center on Intelligent Systems
International Institute of Information Technology Hyderabad, India
{saket.maheshwary, soumyajit.ganguly}@research.iiit.ac.in, vikram@iiit.ac.in

Abstract
In this paper we investigate the problem of user authentication and identification based on their keystroke timing patterns. To this end, we propose Deep Secure: a neural network architecture to authenticate a specific user or identify different users based on their keystroke statistics. A portion of prior literature try to tackle this problem using manually engineered feature combinations while others use complex and hard to train (time-consuming) models like Deep Belief Networks. Our proposed approach is computationally up to 10 times faster when compared with previous neural network models and does not require any manually engineered features. We show how Deep Secure can be used for both user authentication and user identification problem settings with a simple change separating the models. The efficacy (both training and testing) and effectiveness of our proposed model is demonstrated through comparisons with existing methods using the CMU keystroke dynamics benchmark dataset. We achieved an Equal Error Rate(EER) of 0.030, (12.39% improvement) for the user authentication task and an overall accuracy of 93.59% (12.39% improvement) for user identification over state-of-the-art techniques.

1 Introduction
In this era where everyone wants secure, faster, more reliable and easy to use means of communication, there are many instances where user information such as personal details and passwords get compromised thus posing a threat to system security. In order to tackle the challenges posed on the system security biometrics, recent research [Giot et al., 2010] prove to be a vital asset. Biometric systems are divided into two classes namely physiology based ones and behavior based ones. Physiology based approach allows authentication via use of retina, voice and fingerprint touch. In contrast, behavior based approach includes keystroke dynamics on keyboard or touch screens and mouse click patterns.

In this paper we propose Deep Secure: an algorithm to deal with keystroke dynamics – a behavior based unique timing patterns in an individuals typing rhythm which is used as a protective measure. These rhythms and patterns of tapping are idiosyncratic [A. Dvorak and Ford, 1936] the same way as hand writings or signatures are, due to their similar governing neuro-physiological mechanisms. Back in the 19th century, telegraph operators could recognize each other based on ones specific tapping style [Leggett and Williams, 1988].

Based on the analysis of the keystroke timing patterns, it is possible to differentiate between actual user and an intruder. By keystroke dynamics we refer to any feature related to the keys that a user presses such as key down time, key up time, flight time etc. In this paper, we concentrate on authenticating and identifying users based on static text such as user password. The mechanism of keystroke dynamics can be integrated easily into existing computer systems as it does not require any additional hardware like sensors thus making it a cost effective and user friendly technique for authenticating and identifying users with high accuracy.

It is appropriate to use keystroke dynamics for user authentication and identification as studies [Syed et al., 2011; 2014] have shown that users have unique typing patterns and style. Moreover, [Syed et al., 2011; 2014] has proven some interesting results in their research work as well. In their first hypothesis, they proved that the users present significantly dissimilar typing patterns. Second, they have shown details about the relationship between users occurrence of sequence of events and their typing style and ability. Then they explained sequence of key up and key down events on the actual set of keys. They have also shown that there is no correlation between users typing skills and the sequence of events. Hence all these factors make it difficult for intruders to match with the actual users typing patterns. Keystroke dynamics is concerned with users timing details of typing data and hence various features could be generated from these timing patterns. In this paper we are using timing features only on static text.

Neural network based models are frequently used off late in the field of computer vision, speech signal processing and text representation. They are now being adopted in the domains of security as well [Deng and Zhong, 2013]. Unlike classical methods which rely on complex distance metrics or manual feature engineering, these neural network based techniques have multiple advantages over the previous both in task specific performance and scalability. Motivated by the superior results obtained by the neural network models, we present Deep Secure for authenticating and identifying users.
based on keystroke timing pattern. Our main contributions in this paper are as follows:

- We propose Deep Secure, a novel neural network architecture to perform user-based keystroke authentication and identification.
- The design of our proposed model is simple, as it is based on a simple feed-forward neural network. This makes it easy to interpret, implement, maintain, embed and modify as the situation demands.

2 Related Work

Classifying users based on keystroke timing patterns has been in limelight when [Forsen et al., 1977] first investigated whether users could be distinguished by the way they type on keyboard. Researchers have been studying the user typing patterns and behavior for identification. Then [Gaines et al., 1980] investigated the possibility of using keystroke timings as to whether typists could be identified by analyzing keystroke times as they type long passages of text. Later [Monrose and Rubin, 2000] extracted keystroke features using the mean and variance of digraphs and trigraphs. A detailed survey was published [Peacock et al., 2004] on the keystroke dynamics literature using the Euclidean distance metric with Bayesian like classifiers. Initially [Bergadano et al., 2002] and later [Gunetti and Picardi, 2005] proposed the usage of relative order of duration times for different n-graphs to extract keystroke features that was found to be more robust to the intra-class variations than absolute timing. Great results for text-free keystroke dynamics identification was published [Gunetti and Picardi, 2005] where the authors merge relative and absolute timing information on features.

[Zhong et al., 2012] proposed a new distance metric by combining Mahalanobis and Manhattan distances. Many machine learning techniques have also been proposed for keystroke dynamics as an authentication system. These vary from the likes of Decision Trees [Brieman et al., 1984], Support Vector Machines [Cortes and Vapnik, 1995], Neural Networks [Kubat, 1999], Nearest Neighbor Algorithms [Cover and Hart, 1967] and Ensemble Algorithms [Schapire, 1999] among others. Recently [Deng and Zhong, 2013] introduced two new algorithms namely the Gaussian mixture model with the universal background model (GMM-UBM) and the deep belief nets (DBN). Unlike most existing approaches, which only use genuine users data at training time, these two generative model-based approaches leverage data from background users to enhance the models discriminative capability without seeing the imposters data at training time.

One problem faced by researchers working on these type of problems is that majority of the researchers are preparing their own dataset by collecting data via different techniques and the performance criteria is not uniform as well hence comparison on similar grounds among the proposed algorithms becomes a difficult task. To address this issue, keystroke dynamics benchmark dataset is publicly provided (impostor) accordingly. We aim to learn a complex, non-linear hypothesis \( h_{W,b}(x) \) that fits to our data. Here \( W \) is the weight matrix which we learn and \( b \) the bias term. As can be deciphered from Figure 1, our network has 4 weight matrices \( W_1, W_2, W_3, W_4 \). All the hidden layers and our input labels as

2009] collected and published a keystroke dynamics benchmark dataset containing 51 subjects with 400 keystroke timing patterns collected for each subject. Besides, they also evaluated fourteen available keystroke dynamics algorithms on this dataset, including Neural Networks, KNNs, Outlier Elimination, SVMs etc. Various distance metrics including Euclidean distance, Manhattan distance and Mahalanobis distance were used. This keystroke timing pattern dataset along with the evaluation criteria and performance values stated provides a benchmark to compare the progresses of new proposed keystroke timing pattern algorithms on same grounds.

3 Deep Secure

We first formally define our neural-network architecture which we use as a binary classification problem - given a keystroke pattern and an user, does this pattern belong to this user or not. Towards the end of this section we discuss various configurations of our neural network and how to use them in the keystroke based user identification setting.

Let us define our input feature vectors as \( x \in \mathbb{R}^D \) and our labels as \( y \in \{1, 0\} \). Our input training data are sampled either from one particular user or impostors (any other user in the dataset). Similarly the training labels are 1 (user) or 0 (impostor) accordingly. We aim to learn a complex, non-linear hypothesis \( h_{W,b}(x) \) that fits to our data. Here \( W \) is the weight matrix which we learn and \( b \) the bias term. As can be deciphered from Figure 1, our network has 4 weight matrices \( W_1, W_2, W_3, W_4 \). All the hidden layers and our input features

![Figure 1: Illustration of deep secure architecture. Here \( W_1, W_2, W_3 \) and \( W_4 \) represents the weight matrices which are our parameters to be learnt. \( H_1, H_2 \) and \( H_3 \) are the hidden layers of varying sizes. \( I \) and \( O \) are input and output layers respectively.](image-url)
put layer is followed by the rectified linear activation function given by \( f(z) = \max(0, x) \). We define the loss by the negative log-likelihood function which maximized the probability that sample \( x^{(i)} \) gets classified as user or impostor - formally shown in Equation 1 where \( W \) represents the parameters for our model. Learning is done through back-propagation of the losses through our network [Hecht-Nielsen, 1989].

\[
P(y^{(i)} = 1|x^{(i)}; W) = \frac{1}{1 + \exp(-W^T x^{(i)})} \tag{1}
\]

The above explained architecture can be used for a particular user authentication. We need to train \( n \) such models for each in Table 3.

At a high-level we are using a simple feed-forward neural network with an input layer with the size of our feature vectors and three hidden layers of 100, 400 and 100 dimensions respectively. All the layers are fully connected. We gradually increase our hidden layer size followed by a gradual decreasing to the output layer. The higher size of hidden layers introduces sparsity in our network and helps in capturing the inter-feature relations which might be present. The stacking up of hidden layers yield hierarchy and compositional features. By this process we eliminate the need of feature engineering as we let our network take care of it. As is shown later in the Results section, this model of ours is more robust and less prone to overfitting on this key-stroke recognition problem when compared to a simpler 1 hidden layer model. Following subsections explain other building blocks of Deep Secure. We later discuss ablation studies for each in Table 3.

3.2 Dropout

We use dropout [Srivastava et al., 2014] after our hidden layers which act as a regularizer and restricts over-fitting. During our training stage we randomly delete the nodes of each hidden layer with a certain probability \( p(0, 1) \) for each input sample. These neurons do not participate in the back-propagation learning. In testing time, the weights are correspondingly divided by \( 1 - p \). Using dropout, forces the rest of the neurons in the hidden layers to learn more robust features and depend lesser on other specific neurons. In [Srivastava et al., 2014] more details are provided which show that using dropout can be an economic alternative to ensembling various network architectures.

3.3 Batch Normalization

After every fully-connected layer, we use batch normalization [Ioffe and Szegedy, 2015] before the respective activation functions. Using batch normalization we monitored the gap between training and testing loss over epochs narrowed down. This lead to better generalization.

3.4 Leaky ReLU

Non-linear function Rectified linear unit (ReLU) is preferred to sigmoid or hyperbolic-tan because it simplifies backpropagation, makes learning faster while also avoiding saturation. However for large gradients, ReLU can cause particular neurons to die and not participate in learning at all [Maas et al., 2013]. LeakyReLU’s have a small positive gradient \( f(z) = \max(0.01x, x) \) which prevent this dying of a neuron. We applied Leaky ReLU as our activation function after the fully connected layers.

3.5 Adam

In recent times, several algorithms (with implemented software tools) are available for training a deep neural network. While stochastic gradient descent (SGD) for quite some time have been the top choice, there has been study which indicate some of the obvious flaws [Le et al., 2011] in the vanilla implementation. There have been some attempts to automatically tune its learning rate thus resulting in much faster convergence. For Deep Secure we use Adam [Kingma and Ba, 2014] instead of SGD which required a lot of fine-tuning with the learning rate and over 500 epochs to converge.

3.6 Inputs to Deep Secure

The dataset [Killourhy and Maxion, 2009] provides three types of timing information namely the hold time, key down key down time and key up-key down time. Following are the details of three categories of timing features which is used to generate features using keystroke timing dataset [Killourhy and Maxion, 2009]. Figure 2 illustrates various timing features given as input to Deep Secure where up arrow indicates key press and down arrow indicates key release. The dataset consists of keystroke timing information of 51 users, where each user is made to type `die5Roanl` as password. All the 51 users enrolled for this data collection task typed the same password in 8 different sessions with 50 repetitions per session thus making each user to type 400 times in total.

- **Hold Time** also known as dwell time, is the duration of time for which the key is held down i.e. the amount of time between pressing and releasing a single key. In figure 2, \( H \), represents the hold time. Hold time contributes to eleven features (where ten features are corresponding to the ten characters of password and one feature corresponds to the return key).
- **Down-Down Time** key down key down time is the time from when key1 was pressed to when key2 was pressed.
4 Experimental Setup

In this section we discuss the experimental setup, evaluation criteria used and the performance of our proposed model. We evaluated our approach on the CMU keystroke dynamics benchmark dataset [Killourhy and Maxion, 2009] where 51 users were designated for this task. We demonstrate the effectiveness of our model with the average equal error rate (EER) [Killourhy and Maxion, 2009] and accuracy. We compare the results with various proposed anomaly detectors/classifiers which have been used in literature. We used the python library Keras [Chollet, 2015] for building our neural network architecture. All our experiments were carried out on a Pentium 4th generation machine with 4GB memory.

4.1 Training

We frame keystroke dynamics based authentication both as a one-class and multi-class classification problem.

User Authentication: For authentication, Deep Secure learns one model per user, rejects anomalies to the learned model as impostors, and accepts inliers as the genuine user. Consider a scenario in which a user long-time password has been compromised by an impostor. We assume the user to be practiced in typing their own password, while the impostor is unfamiliar with it (e.g., typing it for the first time). We measure how well each of our detectors is able to discriminate between the impostors typing and the genuine users typing in this scenario. We start by designating one of our 51 subjects as the genuine user, and the rest as impostors. We train an anomaly detector by extracting 200 initial timing feature vectors for a genuine user from the dataset. We repeat this process, designating each of the other subjects as the genuine user in turn thus creating models equal to number of distinct subjects or users. Unlike most existing approaches, which only use actual users data at training time, our model leverages data from background users to enhance the models discriminative capability thus improving the prediction performance. We randomly took 5 samples from each background users as negative samples. Note that these 5 random samples were carefully chosen such that no impostor samples that were used in testing were shown during the time of training. For this problem setting, we use the evaluation criteria as mentioned in [Killourhy and Maxion, 2009].

User recognition: This scenario can be viewed as a multiclass classification problem where we need to identify any particular user from a given pool of users. We train only 1 model which learns to discriminate between all the users from our dataset. For experiments, we took the same 200 initial timing feature vector per-user as before. 10% of training data was kept aside as validation data for hyper-parameter tuning.

4.2 Testing

Authenticate Users: We take last 200 passwords typed by the genuine user from the dataset. These 200 timing feature vectors acts as test data. Scores generated in this step acts as the user scores. Next, we take initial 5 passwords typed by each of the 50 impostors (i.e., all subjects other than the genuine user) from the dataset which acts as the impostor scores. Thus we form a test dataset of 200 positive samples and 250 negative samples per user which we provide to Deep Secure and record the output predictions. If $s$ denotes the predictions, the corresponding anomaly score was calculated as $1 - s$ [Killourhy and Maxion, 2009]. Intuitively, if $s$ is close to 1.0, the test vector is similar to the training vectors, and with $s$ close to 0.0, it is dissimilar.

Identify Users: For the multiclass classification problem setting as described above, we test our model by taking the last 200 timing feature vector of each user and combine them to form our test set. Note that here we need not explicitly provide our model with positive and negative samples as the neural network handles it by design of softmax loss function. We need to perform only a forward pass with all features of the full test-set and get the corresponding predictions per sample. We calculate the final accuracy of our network which measures the percentage of correctly classified user based on their input features.
4.3 Empirical Evaluation for user authentication

Based on the genuine user scores and impostor scores generated in the steps above, we generate the ROC curve for the actual (genuine) user. Then we calculate the equal error rate from the ROC curve where the equal error rate corresponds to that point on the curve where the false positive rate (false-alarm) and false negative rate (miss) are equal. We repeat the above four steps, in total of 51 times where every time each of the subsequent user is taken as the genuine user from the 51 distinct users in turn, and calculate the equal-error rate for each of the genuine users. Finally we compute the mean of all 51 equal-error rates which gives us the performance value for all users, and the standard deviation which will give us the measure of its variance across subjects. In order to ensure comparison on same grounds we have used exactly the same evaluation criteria as stated by [Killourhy and Maxion, 2009].

Table 3: Comparison with other top proposed models in terms of Accuracy for user identification

<table>
<thead>
<tr>
<th>Model/Algorithm</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep Secure</td>
<td>93.59</td>
</tr>
<tr>
<td>Manhattan(scaled)</td>
<td>81.20</td>
</tr>
<tr>
<td>Nearest Neighbor(Mahalanobis)</td>
<td>73.60</td>
</tr>
<tr>
<td>Random Forest</td>
<td>69.67</td>
</tr>
<tr>
<td>SVM</td>
<td>66.40</td>
</tr>
<tr>
<td>Deep Belief Nets (DBN)</td>
<td>65.60</td>
</tr>
</tbody>
</table>

While using a simpler 3-layer model (31-400-51) for this problem, we observe overfitting characteristics using our held-out validation dataset. This can be observed in Figure 4. The training loss approaches 0 much faster and after 100 epochs, the validation loss slowly start increasing. There is also a noticeably larger gap between the training and testing losses. Gradually increasing and decreasing the hidden layer sizes, (also adding more hidden layers) helps alleviate this problem and we can see a clear convergence of both the training and validation loss in Figure 5. The final 51-class accuracy obtained with the 3-layer was 80.11% while with our 5-layer model was 83.59% on the testing set. A faster learning rate for Adam helps in reaching the desired accuracy within 50 epochs but results are quite fluctuating both on the validation and testing data. For the activation function, LeakyReLU relatively gave minor improvements over ReLU but performed quite better than sigmoid and hyperbolic-tan as can be seen in Table 2, our ablation study. Using both Batch-Normalization and LeakyReLU led to significant increase in overall accuracy. Using dropout, though the increment was less, it helped our model generalize well as training.

5 Results

Figure 3 shows ROC curve for different users with their Equal Error Rate (EER) value and user number where user number corresponds to the user as stated in CMU dataset [Killourhy and Maxion, 2009]. Table 4 shows the comparison of 16 other proposed keystroke timing algorithms with Deep Secure. This comparison is performed on the same dataset and evaluation criteria thus assuring an objective comparison. Deep Secure is able to achieve an average equal error rate (EER) of 0.30 and with a standard deviation (stddev) of 0.024 across 51 subjects. The average equal error rate (EER) shown in the table below are the fractional rates between 0.0 and 1.0, not the percentages. From Table 4 it can be inferred that Deep Secure performs superior than other proposed techniques in comparison.
<table>
<thead>
<tr>
<th>Model/Algorithm</th>
<th>Average EER (stddev)</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep Secure</td>
<td>0.030 (0.024)</td>
<td>[Deng and Zhong, 2013]</td>
</tr>
<tr>
<td>Deep Belief Nets (DBN)</td>
<td>0.035 (0.027)</td>
<td>[Deng and Zhong, 2013]</td>
</tr>
<tr>
<td>GMM-UBM</td>
<td>0.055 (0.052)</td>
<td>[Deng and Zhong, 2013]</td>
</tr>
<tr>
<td>Median Vector Proximity</td>
<td>0.080 (0.055)</td>
<td>[Al-Jarrah, 2012]</td>
</tr>
<tr>
<td>Manhattan-Mahalanobis (No Outlier)</td>
<td>0.084 (0.056)</td>
<td>[Zhong et al., 2012]</td>
</tr>
<tr>
<td>Manhattan-Mahalanobis (Outlier)</td>
<td>0.087 (0.060)</td>
<td>[Zhong et al., 2012]</td>
</tr>
<tr>
<td>Manhattan (scaled)</td>
<td>0.0962 (0.0694)</td>
<td>[Killourhy and Maxion, 2009]</td>
</tr>
<tr>
<td>Nearest Neighbor (Mahalanobis)</td>
<td>0.0996 (0.0642)</td>
<td>[Killourhy and Maxion, 2009]</td>
</tr>
<tr>
<td>Outlier Count (z-score)</td>
<td>0.1022 (0.0767)</td>
<td>[Killourhy and Maxion, 2009]</td>
</tr>
<tr>
<td>SVM (one-class)</td>
<td>0.1025 (0.0650)</td>
<td>[Killourhy and Maxion, 2009]</td>
</tr>
<tr>
<td>Mahalanobis</td>
<td>0.1101 (0.0645)</td>
<td>[Killourhy and Maxion, 2009]</td>
</tr>
<tr>
<td>Manhattan (Filter)</td>
<td>0.1360 (0.0828)</td>
<td>[Killourhy and Maxion, 2009]</td>
</tr>
<tr>
<td>Neural Network (Auto-associ)</td>
<td>0.1614 (0.0797)</td>
<td>[Killourhy and Maxion, 2009]</td>
</tr>
<tr>
<td>Euclidean</td>
<td>0.1706 (0.0952)</td>
<td>[Killourhy and Maxion, 2009]</td>
</tr>
<tr>
<td>Fuzzy Logic</td>
<td>0.2213 (0.1051)</td>
<td>[Killourhy and Maxion, 2009]</td>
</tr>
<tr>
<td>k Means</td>
<td>0.3722 (0.1391)</td>
<td>[Killourhy and Maxion, 2009]</td>
</tr>
<tr>
<td>Neural Network (Standard)</td>
<td>0.8283 (0.1483)</td>
<td>[Killourhy and Maxion, 2009]</td>
</tr>
<tr>
<td>Mahalanobis</td>
<td>0.1101 (0.0645)</td>
<td>[Killourhy and Maxion, 2009]</td>
</tr>
<tr>
<td>Manhattan (Filter)</td>
<td>0.1360 (0.0828)</td>
<td>[Killourhy and Maxion, 2009]</td>
</tr>
<tr>
<td>Neural Network (Auto-associ)</td>
<td>0.1614 (0.0797)</td>
<td>[Killourhy and Maxion, 2009]</td>
</tr>
<tr>
<td>Euclidean</td>
<td>0.1706 (0.0952)</td>
<td>[Killourhy and Maxion, 2009]</td>
</tr>
<tr>
<td>Fuzzy Logic</td>
<td>0.2213 (0.1051)</td>
<td>[Killourhy and Maxion, 2009]</td>
</tr>
<tr>
<td>k Means</td>
<td>0.3722 (0.1391)</td>
<td>[Killourhy and Maxion, 2009]</td>
</tr>
<tr>
<td>Neural Network (Standard)</td>
<td>0.8283 (0.1483)</td>
<td>[Killourhy and Maxion, 2009]</td>
</tr>
</tbody>
</table>

Table 4: Comparison of 16 different keystroke timing pattern algorithms in terms of Average Equal Error Rate (EER)

Figure 5: Train and Validation loss over epochs for our proposed 5-layer (31-100-400-100-51) neural network.

6 Conclusion and Future Work

In this paper we introduced Deep Secure, a neural network based approach for user authentication via keystroke timing patterns. Deep Secure employs the latest advances in deep learning and optimization techniques to automatically learn feature interactions in the keystroke data. We showcase the two different scenarios where our model can be used with a slight change in its architecture. Experiments on the CMU dataset, demonstrate the superiority of our proposed neural network model over all other methods by a big margin in both the evaluation criterion. We plan on extending our models to other available datasets on this domain. We would also like to investigate if transfer learning can help with user authentication and identification for large pool of users when trained from a limited dataset.

References


[Bergadano et al., 2002] Francesco Bergadano, Daniele Gunetti, and Claudia Picardi. User authentication through


How to Pick Your Friends
A Game Theoretic Approach to P2P Overlay Construction

Saar Tochner and Aviv Zohar
The Hebrew University of Jerusalem, Israel
{saart, avivz}@cs.huji.ac.il

Abstract
One major limitation of open P2P networks is the lack of strong identities that allows any agent to attack the system by creating many false personas. Such attacks can be used to disrupt the overlay network’s connectivity and to sabotage its operation. In this paper, we explore practical ways to defend P2P networks from such attacks. To do so, we employ a game theoretic approach to the management of each peer’s list of known nodes and to the overlay construction mechanisms that utilize this list. We consider the interaction between defender and attacker agents as a zero-sum game. We show that the cost of attacks can be driven up substantially if the defender utilizes available information about peers it chooses to connect to such as their IP address. In addition to the theoretical analysis of the underlying game, we apply our approach to the Bitcoin P2P network and derive effective strategies that guarantee a high safety level.

1 Introduction
P2P networks are used as an underlying communication layer in many applications such as Bitcoin [Nakamoto, 2008], BitTorrent [Norberg, 2009], and DHTs [Urdaneta et al., 2011]. Unfortunately, they are extremely susceptible to isolation attacks in which individual nodes that wish to participate in the network connect only to attackers and are effectively quarantined from all other honest participants. Attackers can then distort the view of nodes regarding events in the network, filter their messages, change, or delay them at will.

Attacks of this sort give attackers great power. One prominent example, is the Bitcoin protocol [Nakamoto, 2008], in which isolated nodes can have their computational power subverted to attack the rest of the network, or may be blocked from issuing transactions.

The general mechanisms through which P2P clients choose their connections may vary, but most use the following general technique: once an initial connection is established with a node (usually with the help of some centralized server that helps to bootstrap the process) nodes share the IP addresses of others with their connections, and maintain lists of possible peers to connect to. Such potential connections are stored in buffers and are exchanged often to keep the lists populated with “fresh” addresses. The important decisions to be made are how to select which nodes to evict from a full buffer, and which nodes to connect to.

A naive approach is to form connections to random nodes in the buffer (as random graphs are typically well connected) and to evict nodes uniformly at random from the buffer if it is overflowing. Other approaches that have been used are to remove the oldest IPs assuming they are the ones most likely to be stale. Attackers can take advantage of such policies to replace all IPs in the buffer of the victim with IPs of attacker nodes (or with un-assigned IP addresses). The defender will thus fail to connect to other honest nodes and will be quarantined by the attacker.

Bitcoin itself employs a more sophisticated eviction strategy, sorting IP addresses into buckets by combining the IP of the sending node along with the IP address in the message itself. Recent work has shown this mechanism to be susceptible as well due to the fact that attackers get nodes to evict IP addresses from their buffer [Heilman et al., 2015]. There are two approaches used in isolation attacks: Sybil attacks [Douceur, 2002] in which attackers create many identities and thus increase their chance of getting connections and Eclipse attacks [Singh and others, 2006] wherein the attacker increases the visibility of each of its nodes by advertising his own IP more aggressively.

In this paper we seek to make the attacker’s job more difficult. We wish to suggest good behaviors for the agents in the P2P network that will successfully avoid being quarantined by attackers, unless the attacker invests a great deal of resources. In particular our work models the limited buffer from which agent we can choose connection and suggests ways to manage this buffer. Our work falls in the general scheme of security games: we consider the attacker and defenders as rational agents that seek to maximize their gains (or alternatively minimize costs and penalties). We assume that if costs are sufficiently high, attackers will be discouraged from at-
tacking or unable to corrupt the required resources.

The main idea that we utilize to increase the cost for
the attacker is to take advantage of properties of the
connections to peers to try and pick connections that
are not just Sybils of the attacker, i.e., to pick a set of
nodes that the attacker would be unlikely to control all
at once. We take inspiration from Bitcoin’s P2P
formation, and consider the IP subnet mask of the peers.
We assume that once an attacker has purchased an IP
address in some range of IPs it is easy for him to gain
access to other similar IPs. Our peer selection strategy
in this case will tend to be biased towards selecting peers
from many different IP ranges. This affects the way we
should manage the IP addresses’ buffer. Our main result
(Theorem 4) states that we can implement a safety level
strategy for this game, using only a limited amount of
memory, in which the IPs are stored. Our contribution
is thus a practical one.

It is important to note that our model has been created
to defend from attacks on a single node. This is not the
general case: the attacker may try to isolate a specific
subset of nodes. We thus approximate an honest node as
“safe” if it is connected to other honest nodes in the local
sense. Our empirical analysis suggests that the resulting
network graph is well connected and difficult to attack.

The remainder of the paper is structured as follows: In
the next section we briefly review related work, then in
section 2 we define our model of the game and the agents’
memory buffer. Section 3 shows results for the game
without any restrictions on the buffer size. In section 4
we restrict the size of the memory buffer, and derive the
safety level strategies for the defender. We conclude and
discuss future work in Section 6.

1.1 Related Work

Work by Douceur [2002] was the first to expose the problem
of multiple identities (Sybils) in P2P systems with no
strong identities. Many such systems have indeed been shown to be vulnerable to such attacks [Castro et al., 2002; Urdaneta et al., 2011].

Bitcoin, a P2P currency system [Nakamoto, 2008] was in fact completely designed to work as an open system with no strong identities. Its overlay formation, however, is still susceptible to attacks [Heilman et al., 2015].

Another approach to defense from Sybils was taken up by ‘SybilGuard’ [Yu et al., 2006], where peers utilize a social network to form their connections (Sybils are assumed to have few connections to honest participants in this setting). Unfortunately, most settings do not have this additional network of relations between peers to use for overlay formation.

Modern botnets are also known to be structured as P2P networks and their susceptibility to such attacks has been used to attack the botnets themselves [Andriesse and Bos, ; Rosso et al., 2013].

The approach of modeling the interaction between at-
tackers and defenders using game theoretic tools is well
established in artificial intelligence [Tambe, 2011]. A line
of work on security games deals with several variants of
such problems. These include the ARMOR project for
security at airports [Pita et al., 2008], transportation
networks [Tsai et al., 2009], and Patrolling [Basilio et al., 2009].

2 Model

We assume the attacker wishes to separate the defender
node from all other honest nodes. We model this as a
game between the defender and the attacker. The first
model we examine, assumes unrealistically that defender
knows all nodes in the ‘universe’ V, and is essentially
unrestricted by memory considerations. We later use this as a building block for the second model in which the defender has a limited memory buffer.

2.1 Preliminaries

let $S^1, S^2$ be the strategy sets of players 1, and 2 corresponding.
Let $U : S^1 \times S^2 \to \mathbb{R}$ be the utility of player 1 in the game, and assume player 2 has utility $-U$ (this is a 2-player zero-sum game). Recall that in a 2-player game with strategies $S^1, S^2$, and utility $U$ the strategy profile $(s^1, s^2)$ for $s^1 \in S^1$ and $s^2 \in S^2$ is called a Nash-equilibrium iff $\forall s^1 \in S^1, U(s^1, s^2) \leq U(s^1, s^2)$ and $\forall s^2 \in S^2, U(s^1, s^2) \geq U(s^1, s^2)$.

Similarly, strategy $s^1 \in S^1$ is called an L-safety-level strategy iff $\forall s^2 \in S^2, U(s^1, s^2) \geq L$ (a similar definition can be stated for player 2).

Recall that in a 2-player zero-sum game the Nash equi-
librium (possibly in mixed strategies) is also the max-
imum solution of the game. This implies that it is com-
prised of safety level strategies for both players.

2.2 The Limitless-Buffer Game

Denote V as the set of peers (either honest or those owned by the attacker) in the network. Let $H$ be the number of connections the defender creates. We assume the attacker can corrupt or acquire a subset $S$ of nodes from the universe $V$ at a cost that we denote by $C(S)$. $C : 2^V \to \mathbb{R}$. We further assume that the attacker gains some value $W_{att}$ from a successful attack against the defender (and that the defender suffers this as a loss).

The 2-player zero-sum game between the attacker and
defender is defined as follows: The defender’s strategy space $S^1$ contains possible sets of nodes that it may connect to. $S^1 = \{U \subset V : |U| = H\}$. The attacker’s strategy space $S^2$ contains subsets of the universe $V$ that he chooses to corrupt: $S^2 = 2^V$. The attack is considered successful iff the attacker owns all the nodes the defender selected. The utility function is thus

$$U(s^1, s^2) = \begin{cases} C(s^2) & \text{if } s^1 \not\subseteq s^2 \\ C(s^2) - W_{att} & \text{if } s^1 \subseteq s^2 \end{cases}$$

We will usually consider the game with mixed-strategy
$\sigma^1 \in \Delta_{S^1}, \sigma^2 \in \Delta_{S^2}$ and assume players maximize their expected utility:

$$U(\sigma^1, \sigma^2) = \sum_{U \in S^2} \sigma^1_U C(U) - W_{att} \sum_{U \in S^2, V \in S^1} \sigma^1_U \sigma^2_V \delta_{U \subset V}$$

in the above $\sigma^1_U$ is the probability that player $k$ chooses the subset $U \in S^k$. $\delta_{U \subset V}$ is 1 if $U \subset V$, 0 otherwise.
2.3 The Restricted-Buffer Scenario

We wish to defend P2P networks in realistic settings so we must take buffers and limited knowledge of the world into account. We thus define the following refinement of the game: We assume that at any point in time the defender can only maintain a set of potential peers in his buffer. Let $B$ denote the buffer size (the number of nodes whose information can be saved). The agent receives a stream of announcements about nodes from which it picks which ones are to be stored in the buffer. We assume honest nodes are advertised at least once in each time period of length $T$ (attacker nodes may be updated more frequently, as is often the case during eclipse attacks). If a node’s details are stored in the buffer, and the buffer is full, a different stored record must be evicted first. The node then chooses its connections from the set of nodes whose records are held in the buffer.

Each node should specify an algorithm $\mathcal{R}$ that implements the functionality that decides, only by the peers’ cost and unique identity (the only revealed information), which node records to save or evict from the buffer. As the defender, we try to find $\mathcal{R}$ that maximizes the minimum amount of resources that an attacker should invest to successfully attack.

The cost of corrupting nodes As a main example, we assume that acquiring an IP costs $c_{\text{new}}$ and that every further IP from that range is cheaper and costs $c_{\text{node}}$. Mark $D_U := \# \text{ masks in subset } U$. Then $C(U) = c_{\text{new}} \cdot D_U + c_{\text{node}} \cdot |U|$. 

3 Analysis with an Unrestricted Buffer

In this section we show results for the setting in which the nodes in $V$ are known to all.

The following lemma shows that it is better for the defender to err on the side of over-estimating the damage from an attack:

Theorem 1. Let $U_W$, be the utility of the game in which a successful attack causes $W_i$ damage. If $(\sigma^1, \sigma^2)$ is a Nash equilibrium in the game with $W_1$ and $W_2 \leq W_1$, then $\sigma^1$ is a $U_W(\sigma^1, \sigma^2)$-safety level in $U_{W_2}$, i.e., $\forall \sigma_2 \in U_{W_2}(\sigma^1, \sigma^2) \geq U_{W_1}(\sigma^1, \sigma^2)$

Proof. $U_{W_1}(\sigma^1, \sigma^2) = \sum_{U \in S^1} \sigma_1^U C(U) - W_2 \sum_{U \in S^2, V \in S^1} \sigma_V^U \delta_{V \subset U} \geq \sum_{U \in S^2} \sigma_1^U C(U) - W_1 \sum_{U \in S^2, V \in S^1} \sigma_V^U \delta_{V \subset U} = U_{W_2}(\sigma^1, \sigma^2) \geq U_{W_1}(\sigma^1, \sigma^2)$

The last step is because $(\sigma^1, \sigma^2)$ is an equilibrium. \qed

Next, we observe that Nash equilibria come in one of two forms: either the attacker corrupts no nodes at all or he places some small probability to “cover” every node.

Lemma 1. For any Nash equilibrium $(\sigma^1, \sigma^2)$ in the game it holds that either $\forall K \sigma_K^1 = 0$, or alternatively, $\forall U \in S^1 \exists K \text{ s.t. } U \subseteq K$ and $\sigma_K^1 \neq 0$.

Proof. If exists $U \in S^1$ s.t. for all $K$ with $U \subseteq K$ holds $\sigma_K^1 = 0$, then the attacker never answers the defender’s strategy to play pure $U$ (denote it with $S_U^1$). Therefore, $U(\sigma^1, \sigma^2) \geq U(S_U^1, \sigma^2) \geq 0$ (the first inequality follows by the Nash equilibrium definition). Clearly $0 = U(\sigma^1, 0) \geq U(S_U^1, \sigma^2) = 0$. \qed

We now show that in any Nash equilibrium, the defender places more probability on selecting more expensive sets of nodes (from the attacker’s support).

Lemma 2. Let $(\sigma^1, \sigma^2)$ be Nash equilibrium. Then $\forall A, B \in supp(\sigma^2)$:

$$C(A) \leq C(B) \iff \sum_{U \subseteq A \cap S^1} \sigma_U^1 \leq \sum_{V \subseteq B \cap S^1} \sigma_V^1$$

Proof. Reminder: If $v$ is the value of the game $U$ and $(\sigma^1, \sigma^2)$ is a Nash equilibrium, then any pure strategy from the support can be used to achieve it. I.e. if $\sigma^2_\lambda \neq 0$ (for some $A \subset V$), then $U(\sigma^1, S_\lambda^2) = v$.

$$v = U(\sigma^1, S_\lambda^2) = C(A) - W_{\text{att}} \cdot \sum_{U \subseteq A \cap S^1} \sigma_U^1$$
$$v = U(\sigma^1, S_B^2) = C(B) - W_{\text{att}} \cdot \sum_{V \subseteq B \cap S^1} \sigma_V^1$$

The first step in each row is because $A, B \in supp(\sigma^2)$, the second is directly from the definition of $U$. Therefore

$$C(A) - C(B) = W_{\text{att}} \left( \sum_{U \subseteq A \cap S^1} \sigma_U^1 - \sum_{V \subseteq B \cap S^1} \sigma_V^1 \right)$$

So $C(A) \leq C(B) \rightarrow \sum_{U \subseteq A \cap S^1} \sigma_U^1 \leq \sum_{V \subseteq B \cap S^1} \sigma_V^1$. \qed

The next corollary is simple, but it contains an insight that we will use later:

Corollary 1. The probability of choosing a group of peers is determined by the attacker’s support, i.e., $\forall (\sigma^1, \sigma^2) \in \Delta_{S^1} \times \Delta_{S^2}$ equilibrium, $\forall A, B \in supp(\sigma^2)$,

$$C(A) = C(B) \iff \sum_{U \subseteq A \cap S^1} \sigma_U^1 = \sum_{V \subseteq B \cap S^1} \sigma_V^1$$

Reducing the defender’s strategy space

In this section, our goal will be to reduce the base of the defender’s strategies space so we can decrease the number of nodes we should remember in the buffer, while achieving the same game value. First, we will define an equivalence relation on $S^1$ and show that it preserves the game value. Our equivalence relation is defined over sets of nodes $A, B \in S^1$ by $A \sim B \iff \forall (\sigma^1, \sigma^2)$ Nash equilibrium in $U$, $\sigma_A^1 = \sigma_B^1$. We will denote this equivalence class with $[A]$. \[ \sigma^1_A = \sigma^1_B \]
Using those equivalence classes as the new defender’s strategies space \((\hat{S}_1)\) we can define a new game \((\hat{U})\).

Intuitively, in this game, the defender does not distinguish between different connections’ sets in the same equivalence class, so he chooses one uniformly. This definition can be also related to the cost to corrupt sets of nodes in any attacker’s Nash strategy (as in Corollary 1).

Formally: let \([A]\) be defender’s strategy in \(U\), \(U\) attacker strategy, then define: \(\hat{U}(|A|, U) = \frac{\sum_{B \in |A|} U(B, U)}{||A||}\) \(\Delta S_1 \rightarrow \Delta S_1\) by \(T(\sigma)|_{|A|} = \sum_{b \in |A|} \sigma_B\)

Define a mapping between the defender’s strategies in both games: \(T: \Delta S_1 \rightarrow \Delta S_1\) by \((T(\sigma)|_{|A|}) = \frac{\delta_{|A|}}{||A||}\)

and \(T^{-1}: \hat{U} \rightarrow U\) by \((T^{-1}(\sigma))|_{|A|} = \frac{\delta_{|A|}}{||A||}\)

The following results show the connection between the two games:

**Lemma 3.** \(\forall (\sigma^1, \sigma^2) \in \Delta S_1 \times \Delta S_1\) holds that

\[ \hat{U}(\sigma^1, \sigma^2) = U(T^{-1}(\sigma_1), \sigma^2) \]

Proof. First note that:

\[ \sum_{U \subseteq V} \sum_{|A| \in S_1} \sigma^1_{|A|} \sigma^2_{\bar{|A|}} \sum_{B \subseteq |A|} \delta_{BCU} = \]

\[ \sum_{U \subseteq V} \sum_{|A| \in S_1} \sum_{b \in |A|} \sigma^1_{|A|} \sigma^2_B \delta_{BCU} \frac{||A||}{||A||} = \]

\[ \sum_{U \subseteq V} \sum_{|A| \in S_1} \sum_{b \in |A|} (T^{-1}(\sigma^1)_B \cdot \frac{||A||}{||A||}) \sigma^2_B \delta_{BCU} = \]

\[ \sum_{U \subseteq V} \sum_{|A| \in S_1} \sum_{b \in |A|} T^{-1}(\sigma^1)_B \sigma^2_B \delta_{BCU} = \]

\[ \sum_{U \subseteq V} T^{-1}(\sigma^1)_B \sigma^2_B \delta_{BCU} \]

Where the second step is \(T^{-1}(\sigma^1)_B = \frac{\sigma^1_{|A|}}{||A||}\).

Then: \( \hat{U}(\sigma^1, \sigma^2) = \)

\[ \sum_{U \subseteq V} \sigma^2_B C(U) - W_{att} \sum_{|A|} \sum_{b \in |A|} \sigma^1_{|A|} \sigma^2_B \sum_{B \subseteq |A|} \delta_{BCU} \]

\[ \sum_{U \subseteq V} \sigma^2_B C(U) - W_{att} \sum_{|A|} \sum_{b \in |A|} T^{-1}(\sigma^1)_B \sigma^2_B \delta_{BCU} = \]

\[ U(T^{-1}(\sigma^1), \sigma^2). \]

The next lemmas are easing the proof of Theorem 2:

**Lemma 4.** If \((\sigma^1, \sigma^2)\) is Nash equilibrium in \(U\), then \(U(T(\sigma^1), \sigma^2) = U(\sigma^1, \sigma^2)\).

Proof. Note that \(\sigma^1_B = \frac{T(\sigma^1)|_{|A|}}{||A||}\) for all \(B \in |A|\) (because this is a Nash equilibrium), therefore:

\[ \sum_{U \subseteq V} \sum_{B \subseteq |A|} \sigma^1_B \sigma^2_B \delta_{B \subseteq C} = \]

\[ \sum_{|A| \in S_1} \left( \sum_{B \subseteq |A|} \sigma^1_B \left( \sum_{U \subseteq V} \sigma^2_B \delta_{B \subseteq C} \right) \right) = \]

\[ \sum_{|A| \in S_1} \sum_{B \subseteq |A|} \sigma^2_B \delta_{B \subseteq C} = \]

\[ \sum_{U \subseteq V} \sum_{|A| \in S_1} T(\sigma^1)|_{|A|} \sigma^2_B \sum_{B \subseteq |A|} \delta_{B \subseteq C} \frac{||A||}{||A||} \]

Using this equation in the game utility function as before, and get: \(\hat{U}(T(\sigma^1), \sigma^2) = U(\sigma^1, \sigma^2)\)

**Lemma 5.** It holds that \(\forall (\sigma^1, T(T^{-1}(\sigma^1))) = Id\). Moreover, if \((\sigma^1, \sigma^2)\) Nash equilibrium in \(U\), then \(T^{-1}(T(\sigma^1)) = Id\) too.

Clear proof.

**Theorem 2.** Nash equilibria in both games have the same game value, and if \((\sigma^1, \sigma^2)\) is a Nash equilibrium in \(U\) then \((T(\sigma^1), \sigma^2)\) is a Nash equilibrium in \(\hat{U}\), and holds that \(U(T(\sigma^1), \sigma^2) = U(\sigma^1, \sigma^2)\).

Proof. At first, we will show that the T function saves the property of Nash equilibrium. Let \((\sigma^1, \sigma^2)\) be a Nash equilibrium in \(U\), we want to prove that \((T(\sigma^1), \sigma^2)\) is Nash equilibrium in \(\hat{U}\). On the one hand, \(\forall \sigma^1 \in S_1, \hat{U}(\sigma^1, \sigma^2) = U(T^{-1}(\sigma^1), \sigma^2) \leq U(\sigma^1, \sigma^2) = \hat{U}(T(\sigma^1), \sigma^2)\) where the first equality is Lemma 3, the second is the definition of Nash equilibrium in \(U\), and the last is Lemma 4. On the other hand, \(\forall \sigma^2 \in S_2\)

\[ \hat{U}(\sigma^1, \sigma^2) = U(\sigma^1, \sigma^2) \leq U(\sigma^1, \sigma^2) = \hat{U}(T(\sigma^1), \sigma^2) \]

where the first equality is Lemmas 5 and 3.

Then we proved that this is a Nash equilibrium. And clearly, they have the same value due to Lemma 4: \(U(\sigma^1, \sigma^2) = \hat{U}(T(\sigma^1), \sigma^2)\)

**Corollary 2.** The defender’s safety level \(l\) in \(\hat{U}\) can be translated to safety level \(\geq l\) in \(U\).

Proof. Let \(\sigma^1 \in S_1\) be a safety level \(l\) in \(U\). We should prove that \(T^{-1}(\sigma^1)\) is safety level in \(\hat{U}\); indeed \(\forall \sigma^2 \in S_2, U(T^{-1}(\sigma^1), \sigma^2) = U(\sigma^1, \sigma^2) \geq l\) Where the equality on the left is Lemma 3, and the inequality on the right is the definition of safety level \(l\) in \(U\).

4 **Buffer-Restricted Implementation**

We now turn to the scenario where buffers are restricted.

We begin by defining new equivalence relation on \(V\): two nodes \(u, v\) are equivalent if for any set of other nodes \(A \subseteq V\), adding \(u\) to \(A\) results in a strategically equivalent set to \(A \cup \{v\}\). Formally: \(\forall u, v \in V, u = v \iff \forall A \subseteq \)
\[ V, |A| = H - 1 \] holds \( A \cup \{u\} \sim A \cup \{v\} \). We will denote the equivalence class of node \( v \in V \) with \( [v]_{eq} \), and the equivalence classes space with \( \hat{V} := \{[v]_{eq} | v \in V \} \).

In general \( \{(v_1, \cdots, v_H)\} \neq \{v_1\} \times \cdots \times \{v_H\}_{eq} \) and generally there is no containment in either direction (See lemma 8 for a specific interesting property).

### 4.1 The Buffer Management Algorithm

In this section we propose a concrete way of implementing the buffer management algorithm \( R \), that utilizes a Bloom filter.

A Bloom filter [Bloom, 1970] is a data structure that uses hash-coded information to encode a set of items. It allows for some small fraction of errors in membership tests (false positives). The error rate can be lowered by increasing memory usage. In our paper, we use this method to avoid nodes that we already saw and accepted including those evicted from the buffer. We use the Bloom filter’s deterministic answer to completely avoid known nodes, and accept a small fraction of false-positives on new nodes.

First, let us see the main theorem of this section, that discusses the benefits of well-implemented buffer management algorithms. Let \( Uni(B) \) denote the uniform distribution on some set \( B \).

**Theorem 3.** Assume that we have some buffer management algorithm \( R \) that satisfies the conditions that were presented in subsection 2.3 and in addition \( \forall A \in \hat{S}^1 \), \( Uni([A] \cap R(v_1, \cdots, v_k)) \sim Uni([A] \cap \{v_1, \cdots, v_k\}) \) (i.e. choosing a connection set uniformly in \([A]\) from the set of nodes in the buffer, is the same as choosing uniformly from the entire universe).

Then we can implement any Nash strategy or safety level on a restricted buffer of size \( O(H \cdot |\hat{S}^1|) \) with the same value as the game on limitless buffer.

**Proof.** Store \( H \) nodes for any equivalence class in \( \hat{S}^1 \). Using the additional given property, \( \forall [A] \in \hat{S}^1 \) choosing a connection set uniformly in \([A]\) from the set of nodes in the buffer, is the same as choosing uniformly from the entire universe. Therefore, we can play any strategy \( \sigma^1 \) in the game \( \hat{U} \), by uniformly selecting a group in the buffer that is in the chosen equivalence class.

Therefore, we implement a choice that is equivalent to the defender’s strategy space \( \hat{S}^1 \) in the game \( \hat{U} \). Let \( \sigma^2 \) be an attacker strategy, then the value of the game that was played in this buffer-limited world is exactly \( \hat{U}(\sigma^1, \sigma^2) \). So finally, if this is a Nash equilibrium strategy, then \( \hat{U}(T^{-1}(\sigma^1), \sigma^2) \) is also Nash equilibrium in \( \hat{U} \) (Theorem 2). If \( \sigma^1 \) is a \( \hat{U} \) safety level then it is also a \( U \) safety level (Corollary 2).

We consider the following algorithm: Let \( EG \) be the equivalence classes ("buckets") in our game, and let \( buckets_{size} \) be the number of nodes that each bucket can hold, and \( B \in \mathbb{N} \) the bytes size of the Bloom filter.

**Algorithm 1:** \( R \) algorithm for \( EG \)

\[
\text{Initialize:} \\
\text{BF} := \text{Bloom-filter buffer of size } B. \\
\forall \text{bucket} \in \text{EG: bucket}_\text{history}[\text{bucket}] = 0 \\
\forall \text{bucket} \in \text{EG: bucket}[\text{bucket}] = \Phi \\
\text{for } n := \text{new input node} \text{ do} \\
\quad b := n.\text{bucket} \\
\quad \text{bucket}_\text{history}[b] ++ \\
\quad \text{if not BF.contains(n) and Prob}(\text{bucket}_\text{size} \quad \text{bucket}_\text{history}[b]) \text{ then} \\
\quad \quad \text{if buckets[b].isFull then} \\
\quad \quad \quad \text{buckets[b].uniformlyRemoveOne} \\
\quad \quad \text{end} \\
\quad \text{buckets[b].add(n)} \\
\quad \text{BF.add(n)} \\
\text{end}
\]

**Lemma 6.** The above algorithm with \( EG = \hat{S}^1, B = \infty \) satisfies the conditions of theorem 3.

**Proof.** We should prove that for any equivalence class in the defender’s strategy space ("bucket"), choose uniformly a node from the buffer has an equal probability to choose it uniformly from the entire input. Assume that we make the choice after we saw \( v_1, \cdots, v_l \) nodes from this buffer. We need to prove that there is a chance of \( \frac{1}{l} \) to choose any node. And indeed, \( \forall j \in \{1, \cdots, l\} \), holds that it inserted into the buffer with probability \( \frac{1}{l} \) and it still in the buffer after the next input in probability \( \frac{1}{l+1} \cdot \frac{1}{l+2} \cdot \cdots \frac{1}{T} = \frac{1}{l^T} \), so after the l’th input: \( \frac{1}{l+1} \cdot \frac{1}{l+2} \cdot \cdots \frac{1}{l^T} = \frac{1}{l^T} \). Therefore, the probability of choose any \( v_j \) uniformly from the buffer is \( \frac{1}{l^T} \), which is exactly the same probability as chose it over all the input \( (v_1, \cdots, v_l) \).

**Continuous games: Refreshing the Buffer and the Bloom Filter**

The algorithm above works for a single ‘round’ of choosing the connections, which is not enough for a continuous game, where nodes in the network come and go frequently. To overcome this difficulty, we can define the network protocol to propagate live nodes every \( T \) time units, and store two copies of the buffer and filter that reset alternately every \( 2T \) time units. This method gives us the ability to remember IPs from a window of \( T \), which is the full available information on the network (as we’ve assumed honest nodes send their IP address to others at least once every \( T \)).

Additionally, we can use the above algorithm with \( buckets_{size} = 1 \) because we assume that we need to choose the connection set once, and that all honest nodes will be available to answer. In more realistic scenarios in which churn is an issue, and honest nodes may be offline

\[ a \text{ Bloom filter with buffer size } \infty \text{ is optimal} \]
at times, we suggest using larger bucket sizes to preserve a sufficiently large set of alternative connections.

In the full version of the paper, we prove that even if some IPs in the bucket can not be selected, e.g. if they are stale, selecting uniformly from the remaining nodes in the bucket is equivalent again to a uniform selection.

4.2 IP masks

For the rest of the paper we focus on the IP mask implementation for the restricted buffer case. A similar treatment applies to other cost functions.

Lemma 7. If $v_1, v_2$ are nodes in the same mask then $[v_1]_a = [v_2]_a$, i.e., $\forall A \subset V \text{ with } |A| = H - 1$ and for all $(\sigma^1, \sigma^2)$ Nash equilibrium holds $\sigma_{A \cup \{v_1\}} = \sigma_{A \cup \{v_2\}}$.

Proof. Assume with contradiction that there exist two nodes $v_1, v_2 \in V$ from the same mask that are not in the same equivalence class. I.e. there exists $A \subset V, |A| = H - 1$ where $A_1 := A \cup \{v_1\}$ and $A_2 := A \cup \{v_2\}$ are not in the same equivalence class. Therefore, exists Nash equilibrium where the defender choose (WLOG) $A_1, A_2$ in probabilities $\alpha_1 < \alpha_2$. For the attacker, the cost of corrupting groups $A_1$ and $A_2$ is identical because $v_1$ and $v_2$ are on the same mask. Therefore, an attacker’s BR is to play strategy $\sigma^2$ where $\sum_{A_1 \subset B} \sigma^2_B > \sum_{A_2 \subset B} \sigma^2_B$ (the cost is identical but the value is higher). Therefore, the defender’s best response is to choose $A_2$ in greater probability than $A_1$. I.e. the defender’s best response is to change his strategy, therefore this is not a Nash equilibrium, and our first assumption is false.

The following lemma shows that we can save representatives from each equivalence class.

Lemma 8. For any defender’s strategy $A = \{v_1, \ldots, v_H\} \in \mathcal{S}_H$, holds that: $[v_1]_a \times \cdots \times [v_H]_a \subseteq [A]$ I.e. we can replace any node with a node in the same mask, and still be in the same strategy equivalence class.

Proof. Our cost function does not distinguish between two nodes in the same mask, we may switch any node with another one in the same mask, and it will cost the same. Choosing them unequally will ease the attacker’s game (according to Lemma 7). We will choose those two groups with the same probability.

As a direct consequence of the previous lemma we derive of our main results:

Theorem 4. We can implement the IP mask game on a limited buffer, with the same game value as the limitless-buffer game.

Proof. On one hand, we can implement any defender strategy $\{v_1, \ldots, v_H\} \in \mathcal{S}_H$ using the classes $[v_1]_a, \ldots, [v_H]_a$ (Lemma 8). On the other hand, we can implement any $[v]_a \in \hat{V}$ using a set of masks (Lemma 7). Therefore, by using the masks as buckets, we can implement any defender strategy in $\mathcal{S}_H$. 

4.3 Safety level

In this subsection, we define the game $\hat{U}$ wherein we assume that the defender can choose one node per mask. This restriction on the defender’s strategy space gives us a good safety level for the original game, while making computation of the strategy easier.

Note that the defender should not differentiate between two masks that have the same number of nodes (in any Nash equilibrium, masks with the same number of nodes are selected with the same probability). Therefore, the strategic equivalence classes are defined by mask size: the equivalence class of $\{v_1, \ldots, v_H\} \in V$ is all the set $\{u_1, \ldots, u_H\}$ where for all $i$, the nodes $v_i, u_i$ are in masks with the same size.

Therefore, define the equivalence classes game $\hat{U}$: Allow the defender to choose only one connection in each mask-size. In mask of size $a$, denote with $M_a$ the number of nodes, and with $\text{avg}_a = \frac{\sum_{i=1}^{M_a} x_i}{M_a}$ the average cost of node. Then we can bound from below the attacker’s cost of corrupting $x_a \in N$ nodes with $x_a \cdot \text{avg}_a$ and the fraction of corrupted nodes with $\frac{x_a}{M_a}$.

We determine the defender’s strategy: choose the connections with probability related to the $\text{avg}$ values. I.e. select the mask-size $a$ with probability $p_a := \frac{M_a \cdot \text{avg}_a}{\sum_{a=1}^{M_a} M_a \cdot \text{avg}_a}$, and choose a single node uniformly over all the masks.

Denote $y_a \in \{0, 1\}$ as the defender’s probability of choosing from mask-size $a$ or not, and $x'_a$ as $x_a$ if $x_a \neq 0$ or $\frac{M_a(1-E(y_a))}{E(y_a)}$ otherwise. The game utility is: ²

$$U(\hat{x}, \hat{y}) = \sum_{i} x_i \cdot \text{avg}_i - W_{att} \cdot \prod_{a} \left(\frac{x'_a}{M_a}\right)^{y_a}$$

$$\sum_{i} x_i \cdot \text{avg}_i - W_{att} \cdot \prod_{a} \text{avg}_{y_a} \left(\frac{x'_a}{M_a}\right)$$

Note that $y_a$ are Bernoulli distributed parameters. Let $I = [i_1, \ldots, i_{\ell}]$ to be a list of indexes of mask-sizes, and denote by $p_I$ the probability to choose from mask of size $i_{\ell}|1|$, then $i_{\ell}|2|$, and so on. Then $p_I = \prod_{j=1}^{\ell} \frac{p_{I|j}}{1 - \sum_{l=1}^{\ell} p_{I|l}}$.

Therefore, we get $y_a \sim B \left(\sum_{I|H = H} p_I, p_I \right)$, and then:

$$\mathbb{E}(\hat{U}) = \sum_{i} x_i \cdot \text{avg}_i - W_{att} \cdot \prod_{a} \left(\sum_{I|H = H} p_I \frac{x'_a}{M_a}\right)$$

$$= \sum x_i \cdot \text{avg}_i - W_{att} \cdot \prod_{a} \left(\sum_{I|H = H} p_I \frac{M_a}{x'_a}\right) \prod_{a} x'_a$$

²according to our assumption it’s true because $y_i \in \{0, 1\}$. Otherwise it’s just an upper bound.
Figure 1: Number of subnet masks in the Bitcoin network (data crawled at September 2015).

Figure 2: Fraction of times that this mask-size is chosen. Calculated on Bitcoin’s network, with $H = 8$, $\frac{\text{max}}{\text{node}} = 10$ (sum of all fraction should be 8).

5 Evaluations

5.1 Evaluation of $U$

To retrieve an actual value for the calculation above, we examine the behavior of the Bitcoin network. We collected a snapshot of all the nodes that connected to the network (including the the IPs appearing on site blockchain.info). From this data we collected statistics on the number of masks, and distribution of nodes within each one (see Figures 1).

We calculated the value of the Bernoulli parameters (Figure 2), and concluded the following term, which is multiplied in the expected utility formula by the profit from the attack: $\prod I_{|I|=H, a \in I} \approx 4.357 \cdot e^{-51}$. Therefore only attacks that are highly profitable can derive a negative game value for the defender.

5.2 Evaluating the safety level for Bitcoin

We will use Corollary 2 to deduce the solution for the more general game $U$. Let $m_1, \ldots, m_k$ be the mask sizes and $\hat{y}_a$ the probability to choose a connection from the $a$'th mask. Due to the cost function’s definition, for all $i, j$ holds $m_i = m_j \Rightarrow \hat{y}_i = \hat{y}_j$, therefore $\hat{y}_a = \frac{y_{na}}{|\{i|m_i = 8\}|}$.

From here, we derive the probability of choosing IPs from each mask (Figure 3). When we switch back to the original game formulation:

$$\mathbb{E}(U) = \sum I_{i=1, \ldots, k} \cdot \hat{y}_a \cdot g_i - W_{att} \prod I_{|I|=H} \left( \sum I_{|I|=H, \sum m_i \in I} \frac{p_l}{M_{m_l}} \right) \prod I_{x_l}$$

And finally: $\prod I_{|I|=H} \left( \sum I_{|I|=H, \sum m_i \in I} \frac{p_l}{M_{m_l}} \right) \approx 1.785 \cdot e^{-8807}$

Therefore the attacker needs to invest great amount of resources to gain a negative game value.

5.3 Comparing to a Naive Benchmark

In this section we would like to compare our results to a naive benchmark, where each node chooses its connections uniformly from the buffer, and the buffer is chosen uniformly from the whole network. In the first benchmark (the most naive), we assume that the node may be subject to repeated transmissions of the same IP, which it can not detect unless that IP is already in the buffer. We assume that the buffer size is the current buffer size of the Bitcoin’s nodes (= 20480 unique addresses). For the second benchmark, we assume uniform selection of IPs, but with an accompanying Bloom filter to filter out re-transmissions. We consider a network with the proportion of masks that is derived from Bitcoin’s topology. Considering our strategy, an attacker’s response to our strategy is to create nodes in a way where $p_a$ is equal for any mask $a$. Then, we calculate the probability to be successfully attacked as a function of the attacker’s investment. The results are shown in figure 4 along with the probability that our own algorithm is successfully attacked.$^3$

6 Future Work & Conclusions

In this work we explored a game theoretic model for P2P network formation. Our results indicate that given some model for the cost of nodes for an attacker it is possible to select each peer’s connections so as to reduce the likelihood that it is isolated by an attacker.$^3$

$^3$The ‘spike’ in the graph caused by the number of masks exists (when the attacker corrupted nodes in all the masks). In this case, the probability to successful attack is the same as in the naive approach.
Part IV

Posters
AN INTELLIGENT CONTEXT-AWARE BIOMETRICS SYSTEM BASED ON AGENT TECHNOLOGY: FATINA SHUKUR AND HARIN SELLAHEWA
An Intelligent Context-Aware Biometrics System Based on Agent Technology

Fatina Shukur and Harin Sellahewa
The University of Buckingham, United Kingdom
(fatina.shukur, harin.sellahewa)@buckingham.ac.uk

Abstract
Traditional biometric systems deal with each instance of identification in the same way irrespective of the circumstances in which the biometric samples were captured at different times or for different applications. Our main objective is to enhance the traditional biometric identification process and improve the decision making process. This is by giving biometric systems an intelligent and flexible identification mechanism using agent technology. Our aim is to develop a multiagent based framework to represent a context-aware adaptive biometric system with multimodalities.

1 Introduction
A typical biometric system includes five main steps to identify a person as we will describe them in details in section 4. Each one might be affected by multiple factors and particularly when a number of biometrics modalities work in the same system. Such factors that have to be considered when designing a biometric system include in general: the quality of sensors, illumination, noise and other such environmental conditions; the biometric modalities used; type of user’s interaction with the system, the type of features and classifiers and the image processing techniques applied at each step of its processes. All these factors will affect the final objective of the system which is to accurately identify/verify the claimed identity of a person.

Traditional biometric systems essentially use specific biometric modalities and techniques at each of their five steps of the identification process irrespective of those techniques being the best one at that specific time or not. They use adaptive techniques to consider only one step/factor and ignore the rest such as Sellahewa and Jassim [2010] who proposed adaptive techniques for quality-aware illumination normalisation and fusion for face recognition.

Agents based biometrics have been used to enhance such traditional mechanism and again in partial biometric system process steps such as in a classification module only [Abreu and Fairhurst, 2011]. Therefore, we propose to design a framework based on multiagents that consider all the steps and factors in a cooperative manner to support adaptive and intelligent solutions across all of these steps.

2 What are Context-aware and Adaptation?
Context-aware in biometrics means at any given instance of identification or identity verification, the system has the ability to find the best configuration information (awareness) for the requirements of its three main components (user, application, environment), which have a direct affect to make a final identification decision correct or not. Then, the system has to respond adaptively to achieve these requirements. The system works self-awareness and adaptation to get a set of requirements/attributes for each one of its components by using agent technology as follows

Unidentified user context agent is either a cooperative user that gives his profile data and would either have a preferred choice of biometric data to be collected or does not have depending on the application conditions, or a non-cooperative user that does not have such choice at any application/scenario. This agent will determine what kind of user the system is supposed to work with.

Application context agent is based on the type of the application. Each one may require different level of cost, accuracy, convenience, usability, speed of process, privacy and security issues, error rates, etc. In addition, the type of extracted biometric modal(s) is either optional, compulsory or mixed. This agent determines which application we have, and what are the levels of conditions the system should get.

Environment context agent is concerned with the type of environmental conditions that the system is situated in. These conditions are varied such as illumination, quiet or noisy space, occlusion, indoor or outdoor scene, etc.

All these three components with their particular parameters are incorporated and formed the overall system context at a particular instance of identification.

2.1 Use Case: Remote Authentication Service
There is no one specific combination of requirements that works best every time for every biometric application. For example, ATMs are normally placed in controlled environments with cooperative users, whilst surveillance applications have opposite scenarios. These two applications can use the same framework of biometric system and the agent technology can figure out and adapt all the possible requirements based on the context of the given scenarios.
Figure 1 shows a biometric system that provides authentication services for different applications such as banks, and border control. For example, bank ‘X’ needs to authenticate its customers to offer its services (e.g. ATM, m-payment), with each service is required different levels of authentication accuracy and user/merchant convenience depend on the type, amount and location of the transaction. Accordingly, the system must be able to select the most appropriate biometric modal(s) based on the application context, sensor availability and environmental conditions.

Bank ‘Y’ or any other applications could have completely different set of requirements in authenticating their customers. Thus, the system has to be context-aware and adaptive to give such service to heterogeneous applications.

3 A Novel Framework for Adaptive Biometrics System Based-Multiagents

We propose a multiagent framework to represent a context-aware adaptive biometric system. The agents will help conduct the system’s internal processes so that they are able to utilise the best approaches to identify an individual. The framework will include five sequential modules as follows:

1) Sensing module uses an appropriate sensor according to the required biometric sample. This can be done either with a user's agreement or remotely. Sensors that could capture biometric samples in a good quality will help to minimise the ratio of noise to clean and process these samples. An agent will check the sample’s liveness and quality to decide to pass it on to the next module or to re-capture it, 2) pre-processor module normalises the captured samples to a canonical form. It will remove the undesirable noise from these samples. Traditional biometric systems preprocess samples even when they are of good quality, but in our framework, an agent will decide the actual necessity when to do it, 3) features extractor module processes the biometric data to generate a set of distinct features and either store them as templates database (in the enrolment) or send them forward to the next module (in the identification). Features are extracted based on the type of the modality. An agent will select the best feature(s) representation based on the application context and the sample quality determined by the previous module, 4) classifier and matcher module compares between the extracted template of the new biometric sample during the identification process and all the stored templates in the database. An agent will determine which classifier to select based on the type of features extracted and the level of accuracy is required. This module generates a match score for each extracted feature from the biometric sample, and 5) decision maker module: an agent based score fusions approaches will be used to determine the best score fusion strategy for a given instance. As the framework accommodates multimodalities, therefore, this module will use agents to fuse scores at two sequential levels: 1) an agent based sample score fusion is to receive and fuse all the match scores of different samples ($S_1$, $S_2$ .. $S_n$) which are related to the same modality ($M_i$). There is one agent is concerned with one modality in order to handle all its samples’ scores ($M_1.S_1$, $S_2$ .. $S_n$; $M_2.S_1$, $S_2$ .. $S_n$; $M_n.S_1$, $S_2$ .. $S_n$), and 2) an agent based modality score fusion is to collect a number of results from these multiple agents according to the number of modalities in the system.

At the final step, another agent (i.e. a decision making agent) is used to present the final score to determine the predicted identification decision by the biometric system for a given user. After making the decision, there is a check by using a threshold comparison if template update is required. In addition, some applications (e.g. border control) has an operator who may update the stored templates if necessary.

At every step of the above five modules, there will be several options to select from (e.g. which classification method is most appropriate?) Multiagents will be used to determine the most appropriate solution adaptively based on the given scenario and awareness of the system’s context. Agents will negotiate (e.g. game theory- and auction-based approaches [Wooldridge, 2002]) with each other, use their past experiences, and change opinions if necessary to arrive at an optimal result for the final identification decision.

4 Conclusion and Future Work

This paper proposed a conceptual framework for context-aware adaptive multimodal biometrics system using agents. Agents will be used at each of the key processing steps of a biometric system to determine the most suitable action to take based on the application, user and environment context at the time of identification. Our future work is to implement and evaluate the proposed framework.

References


Goal Recognition Assisted Decision Making in Security Games: A Real-time Attack Graph Interdiction Game

Kaiming Xiao¹, Cheng Zhu¹, Kai Xu², Yun Zhou¹, Xianqiang Zhu¹, Weiming Zhang¹

1. Science and Technology on Information Systems Engineering Laboratory, NUDT, Changsha, 410073, China
2. The Institute of Simulation Engineering, College of Information System and Management, NUDT, Changsha, 410073, China
kmxiao@nudt.edu.cn

Abstract

Security games provide a methodology for making decisions when taking attackers' reactions into account, whereas attack graph is an efficient modelling technique for security risk assessment. In this paper, we proposed a real-time attack graph interdiction game by utilizing inferred knowledge from goal recognition, as well as a strategy to bridge the gap between the observation and decision making. Initial experimental results show the effectiveness and accuracy of the proposed methods.

1 Introduction

Cyber security is an epitome of asymmetric, strategic conflict between defenders and attackers, which is usually modelled as Stackelberg security games [Wilczynski et al., 2016]. Specifically, the attacker launches a series of intrusion actions persistently such as network scanning and vulnerability exploiting to penetrate the targeting network, while the defender deploys countermeasures such as intrusion detection/prevention systems, firewalls and honeypots on selected components to protect the network from cyber attacks [von Solms and van Niekerk, 2013].

Attack graph, on the other hand, is one of the tools for analysing the security landscape of a network; thus can provide both players in the security game with an overview of the battlefield, which contains all possible penetrating paths towards critical goal nodes. Although, abundance of studies have been conducted on generating algorithms and analysis methods of attack graphs [Yi et al., 2013], few studies have utilized knowledge from attack graphs thereby assisting defenders in making better decisions in security games.

Recently, some researchers have studied static security games on attack graphs, such as the game-theoretical approach for honeypot deployments using attack graph information [Durkota et al., 2016], the network interdiction game based on attack graphs [Nandi et al., 2016]. However, real-time decision making in security games on attack graphs, as well as the utilization of knowledge from attack graphs, remains an open question. Therefore, we propose a Model Predictive Control (MPC) strategy to bridge the gap between the goal recognition and decision making in the real-time Attack Graph Interdiction (AGI) game, which is based on a proposed Markov Decision Process (MDP)-based goal recognition and a Bi-level Mixed Integer Programming (BLMIP).

2 MPC Strategy for the Real-time AGI Game

2.1 Problem Definition

In the real-time AGI game, the attack graph is donated by $G(N, A)$, where node set $N$ represents attack states of the networked system and arc set $A$ represents atomic attack-s. Let $c_k$ denote the attack cost on the arc $k = (i, j) \in A, \forall i, j \in N$, whereas $r_k$ and $d_k$ denotes the defence cost and the added attack cost caused by the countermeasures on arc $(i, j)$ respectively. The attacker aims to penetrate to a certain node $g \in N$ from an initial state node $s$ at the lowest cost, while the defender attempts to deploy limited countermeasure resources $R$ on a selected set of arcs in order to maximize the lowest cost of the attacker in real time. That is, both the attacker and the defender adopt an observe-and-response action rather than an once-and-for-all decision. Meanwhile, we assume that the attacker’s exact goal node $g$ is unknown for the defender, which is rife and reasonable in real conflicting games in cyberspace. Hence, we can formulate the real-time AGI game as a multi-stage BLMIP problem:

$$\max_{x_t \in X} \min_{y_t} \sum_{k \in A} (c_k + x_{kt}d_k)y_{kt}$$

s.t. $\sum_{k \in FS(i)} y_{kt} - \sum_{k \in RS(i)} y_{kt} = \begin{cases} 1 & \text{for } i = s_t \\ 0 & \forall i \in N \backslash \{s_t, g_1, \ldots, g_m\} \\ -p(g_j) & \forall i = g_j, j \in \{1, \ldots, m\} \end{cases}$

$x_{kt} \in \{0, 1\}$, $\forall k \in A; y_{kt} \geq 0$, $\forall k \in A$

where $X = \{x_t \in \{0, 1\}^{|A|} | x^T X_t \leq R_t\}$, and $\sum_i R_t \leq R$ is an overall constraint for the whole multi-stage game. $k \in FS(i)(k \in RS(i))$ denotes arcs directed out of (into) node i. $x_{kt}$ and $y_{kt}$ are decision variables, where $x_{kt} = 1$ if arc $k$ is interdicted by the defender; else $x_{kt} = 0$; $y_{kt} = 1$ if arc $k$ is exploited by the attacker; else $y_{kt} = 0$. Besides, $0 \leq p(g_j) < 1, \sum_{j=1, \ldots, m} p(g_j) = 1$, the probabilistic distribution over the possible goals $g_1, \ldots, g_m$.

2.2 MDP-based Goal Recognition

The aim of goal recognition is to provide the defender with probabilistic distribution over the possible goals. The proposed MDP model is a combination of three
parts: a) the standard MDP; b) the agent goal and c) the goal termination variable, which are denoted by a tuple \( s_0, S, G, e, A, P_s(s'|s) \), \( O \geq 0 \) where \( s_0 \) is the initial state, \( S \) denotes the non-empty state space with goal states \( G \subseteq S \). \( e = \{0, 1\} \) denotes the termination states, and \( A, O \) denotes the set of actions and observations respectively. \( P_s(s'|s) \) is the probability for an action \( a \in A, s, s' \in S \). Essentially, the model is a Dynamic Bayesian Network, in which all causalities could be depicted. Thus, the behavior of system evolution can be described using a state transition function \( (P_s = p(s_t|s_{t-1}, a_t)) \) and an observation function \( (P_o = p(o_t|s_t)) \).

Recognizing the evader’s goal is an inference problem trying to find the real goal behind agent actions based on observations online. To achieve this, we use Particle Filter method which is an approximate inference method designed to handle non-Gaussian, nonlinear and high-dimensional problems.

### 2.3 MPC Strategy for Decision Making

The MDP-based goal recognition model serves as the system model in this MPC framework, and the optimizer is defined to solve the [RTAGI-P] as a single-stage static problem in a rolling horizon manner.

In each stage \( t \), we solve the [RTAGI-P] optimally for decisions \( x^*_t \); however, only the decisions relating to \( FS(s_t) \), i.e., the outgoing set of the current source node \( s_t \), are implemented at stage \( t \). That is,

\[
x_{kt} = x^*_{kt}, \forall k \in FS(s_t); \quad x_{kt} = 0, \forall k \notin FS(s_t)
\]

Thus, only a small part of countermeasure resources are deployed in each stage \( t \) as the urgent and necessary deployment, i.e., \( R_t = \sum_k r_{kt} x_{kt} \), \( \forall k \in FS(s_t) \).

The remaining resources are still available for future deployment. That is, using this MPC strategy the defender can adopt an observe-and-response decision adaptively. This helps the defender reduce the decision-making risk due to the uncertainty of its opponent’s intention. Accordingly, the defender can avoid countermeasures resources waste and achieve more robust decisions.

### 3 Experiments

A set of attack graphs are generated according to the method in [Nandi et al., 2016] and are used to illustrate the efficacy and efficiency of the proposed MDP-based goal recognition and the MPC strategy for real-time AGI game.

We run the agent decision model of the attacker repeatedly and collect a test dataset consisting of 100 labeled traces on a simulated 20 \( \times \) 40 attack graph. Our inference method is evaluated and validated in Table 1 by measuring its precision, recall and \( F \)-score, which are frequently used to measure overall accuracy of the recognizer. It can be observed that when the progress rate is bigger than 40\% and 50\%, the values of three measures are over 80\% and 95\% respectively.

The performance of proposed MPC strategy for decision making is then compared with a static interdiction strategy, as shown in Figure 1 (Error-bar). A defender who adopts the MPC strategy achieves more payoff (i.e., the added penetrating cost of its opponent) than those who adopt the static defence strategy when given the same amount of countermeasure resources. Besides, this superiority of MPC strategy increases as the growth of available countermeasure resources (a certain percentage of \( \sum_k r_{kt} \)). As shown in Figure 1, the added penetrating cost under the proposed MPC strategy is nearly 2 times of that under the static strategy, which is an overwhelming improvement for the defender’s decision making.

### 4 Conclusion

We present a real-time attack graph interdiction problem using a game-theoretical approach, and a MPC strategy is proposed to bridge the gap between the observation and decision making. Experimental results show the effectiveness and accuracy of our methods as well as the value of inferred knowledge utilization to decision making in security games.

### Acknowledgments

This work is sponsored by the National Natural Science Foundation of China under Grants No.71571186 and No.71471176.

### References


<table>
<thead>
<tr>
<th>Progress Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>0.282</td>
<td>0.298</td>
<td>0.290</td>
</tr>
<tr>
<td>20%</td>
<td>0.445</td>
<td>0.529</td>
<td>0.484</td>
</tr>
<tr>
<td>30%</td>
<td>0.610</td>
<td>0.616</td>
<td>0.612</td>
</tr>
<tr>
<td>40%</td>
<td>0.843</td>
<td>0.814</td>
<td>0.828</td>
</tr>
<tr>
<td>50% - 100%</td>
<td>&gt; 0.950</td>
<td>&gt; 0.950</td>
<td>&gt; 0.950</td>
</tr>
</tbody>
</table>

Table 1: Inference Evaluation Figure 1: Interdiction Performance
ROCSAFE: Remote Forensics for High Risk Incidents

Brett Drury, Nazli Bagherzadeh and Michael G. Madden
Discipline of Information Technology, National University of Ireland Galway,

1 Introduction
Incidents that involve the dispersal of Chemical, Biological, Radiological or Nuclear material (CBRNe) material, although rare, can cause not only material damage, but also can contaminate natural resources as well as being hazardous to humans for long periods of time. For example, the Chernobyl Disaster caused the displacement of 110,000 people from the immediate area, and was estimated to have affected a further 220,000 people [USNRC, 2009]. The hazardous environments created by these incidents impede and endanger any subsequent forensic investigations. The dangers to first responder and forensic teams was highlighted by a recent investigation that found that there were 70 cancers specific to first responders and related workers who attended the World Trade Centre site in 2001 [Goodman, 2016]. A key principle underlying our research is that lives of forensic teams can be protected if the physical collection of evidence can be automated. The Remotely Operated CBRNe Scene Assessment & Forensic Examination (ROCSAFE) project which is funded by the HORIZON 2020 programme is developing strategies and technologies that will automate the collection of evidence. This paper will provide an overview of ROCSAFE as well as its aims for the future.

2 ROCSAFE Overview

The central premise of the ROCSAFE project is that automated collection of evidence from incidents that involve CBRNe material is possible through the use of autonomous RAV (Robotic Aerial Vehicles) and RGV (Robotic Ground Vehicles). The strategy that the project will use is that the RAVs will survey the scene. These vehicles will be equipped with infra-red, video and still photography as well as sensors for chemical, biological and/or radiation threat detection. The sensor configuration will be chosen by the scene commander.

The information from these sensors and cameras will then be transmitted to a Central Decision Management (CDM). The CDM has as part of its function the ability to use probabilistic reasoning over time to determine the most likely threats, likely locations of hotspots/epicentres, and recommend the forms of forensic evidence that should be sought. The CDM will estimate the best route for the RGVs to collect the previously observed evidence and return it to a mobile laboratory. An overview of the ROCSAFE project is provided in Figure 1.

The diagram in Figure 1 demonstrates the aims of the ROCSAFE project which are to assess the scene, collect evidence, and provide decision support. The scene commander will have access to the infra-red, video and still images as well as sensor information. The graphical user interface (GUI) will be designed to limit the cognitive overload for the scene commander.

A motivation for the development an AI-based decision support system is that the assessment of CBRNe risks and recovery of CBRN-contaminated evidence is not a routine or everyday task. In addition there are extremely large number of variables to consider, which may be beyond an individual or group of experts to evaluate. ROCSAFE’s decision support system will highlight key information, therefore reducing information overload on the scene commander, and who then can retrieve the potentially relevant standard operating procedures.

3 Scene Assessment
The scene assessment strategy is predicated upon dividing the incident scene into predetermined subdivisions. At the start of the scene assessment process each of the subdivisions will have an equal probability of containing evidence. The RAVs will assess the scene quickly and relay back information from sensors as well as infra-red and video cameras. It is unlikely that they will relay any detailed information, however the video and infra-red may detect indicators (objects) of evidence, such as: small craters, debris, and damaged buildings. Although object detection is primarily associated with the visible spectrum, it has been used with infra-red imagery [McClintock et al., 2011]. The individual RAVs will be coordinated with flocking or swarm strategies that have been used in other search based problems [Cimino et al., 2015].
4 Central Decision Management

A key component of the CDM will be a Dynamic Bayesian Network (DBN). A DBN has been chosen because it is able to reason with multiple data streams. The initial structure of the DBN will be created from outside sources such as standard operating procedures from expert end-users. In addition to the DBN, there will be image analysis of the video, still and infra-red images. The proposed method of analysing images is Convolutional Neural Networks that will highlight regions of interest in the aforementioned images. Algorithms that analyse sensor information are also being developed. Finally, intuitive Graphical User Interfaces (GUIs) are being developed to display the relevant information from which the scene commander can make informed decisions.

There are a number of challenges that the DBM will have to face. For example the differing time frames of data-streams. The video and infra-red streams are likely to be constant, data from other sensors may suffer from lag and will transmit data for short periods. In addition the data gathered from video and infra-red cameras mounted on the RA V from differing altitudes will need to be accounted for in the structure of the Bayesian Network. This will build upon the work of [Aleks et al., 2009; Enright et al., 2010; 2013].

5 Conclusion

The ROCSAFE project is designed to advance the automated collection of evidence. In addition to the described advances, there will be advances in sensor technology. It is hoped that this project will develop technology that will protect forensic teams as well as guaranteeing the chain of evidence. This research has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 700264, ROCSAFE (Remotely Operated CBRNe Scene Assessment and Forensic Examination).

References


[Goodman, 2016] Leah McGrath Goodman. 9/11s second wave: Cancer and other diseases linked to the 2001 attacks are surging, 2016.


Open Social Data Crime Analytics

Ihsan Ullah, Caoilfhionn Lane, Brett Drury, Marc Mellotte, and Michael G. Madden
Insight Centre for Data Analytics, National University of Ireland, Galway, Ireland
ihsan.ullah@insight-centre.org

Abstract

Crime is under-reported. Reporting crime requires the victim to complete a number of administrative obligations. These obligations, as well as the nature of the crime, may create an inertia that discourages the reporting of the crime (for example, being defrauded might damage a financial organisation’s reputation). However, there may be information leaks from compromised organisations, via affected customers on social media. A key advantage of using social data is that it is often immediate, and can have indications of the nature of a crime such as (1) named entities, for example, Bitcoin or PayPal; (2) geocoding information; and (3) the affected persons. Our aim in this work is to use social media platforms e.g. Twitter, Reddit, Facebook, etc. to detect signals of cybercrime incidents. Such signaling is arguably a better indicator of the extent and effect of cybercrime than traditional reporting methods.

1 Introduction

Computers and the Internet have become part of daily life for almost all people and organizations. The daily usage of smartphones, phenomena related to the internet of things, and a move to cashless societies etc. have increased the risk of cybercrime. The definition of cybercrime encompasses any crime committed with the help of computers or the internet. However, the majority of these crimes are under-reported due to many reasons e.g. because of lack of knowledge or due to potential damage to reputation [Levi et al., 2017; Calnan and Denise, 2016].

According to the IBM X-Force threat intelligence index 2017, the five most targeted areas are financial institutions, information and communications, manufacturing, retail, and healthcare [Alvarez et al., 2017]. On 12th May 2017, ransomware WannaCry attack hit around 150 countries and affected about 200K computers [Karla and Adam, 2017]. In January 2015, Facebook & Google confirmed a phishing scam carried out between 2013 and 2015 in which both the companies lost US $100 m. The vendor management team were asked to transfer money based on fake documents. In December 2016, a CEO fraud was perpetrated on Meath County Council, Ireland, in which a cybercriminal impersonated the county council CEO and secured the transfer of €4.3m to an account in Hong Kong.

In a recent survey [Levi et al., 2017], three months of crime-related data, provided by Action Fraud UK (the national centre for reporting fraud and cybercrime) were analyzed. Out of 106,681 reported incidents, 4% of the incidents were related to cybercrime. The banking and credit industries face the highest number of fraud incidents [Levi et al., 2017; Alvarez et al., 2017].

Table 1 shows our analysis of attacks on financial institutions since 2014. When the end customer is affected directly, information may leaks. For example, we reviewed tweets about recent compromise of Tesco Bank. Tesco customers started to complain on Twitter from the 31st July 2016 whereas Tesco only instructed their customers to take extra security precautions on 6th November 2016 (Newspapers reported the breach on 8th November 2016). There are indications that financial institutions do not always report due to potential concerns over reputational damage e.g. SWIFT fraud in Ecuadorian Banco del Austro [Spier, 2016].

2 Indicators of Cybercrime on Social Media and Web Forums

There are a number of sources of social media, social news, and web forums such as Twitter, Facebook, Instagram, Google+, Reddit, IRC, devRant, etc. that potentially contain indicators of cybercrime. One of the most frequently quoted sources in the literature is Twitter (users tweet around 500 million times per day). A number of studies have examined the role of Twitter in relation to criminal acts. The best channels to reach the public during a health crisis are Twitter, Facebook, and Instagram. It is also a good platform for collecting and analyzing data [Burnap and Williams, 2016]. Using Twitter to detect criminal acts is a recent area of study. Tweets have been used to detect both offline and online criminal acts, for example, to predict hit-and-run crimes from traffic alerts[Wang et al., 2012], detect cyber hate [Burnap and Williams, 2016], and to study online rumors in terms of offline harm [Webb et al., 2015]. Social media is also used in research for health surveillance. The research on detection and analysis of phishing, spam, rumour, riot, etc. may be applicable in the detection of cybercrime.
Table 1: Major cyber attacks on financial institutions since 2014

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
<th>Year</th>
<th>Target</th>
<th>Damage</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>WannaCry</td>
<td>~ 150 Countries</td>
<td>2017-05</td>
<td>~ 200K Computers</td>
<td>NA</td>
<td>Ransomware</td>
</tr>
<tr>
<td>Lloyd Banking Group</td>
<td>UK</td>
<td>2017-01</td>
<td>Online Services</td>
<td>NA</td>
<td>DDoS</td>
</tr>
<tr>
<td>Tesco Bank</td>
<td>UK</td>
<td>2016-11</td>
<td>~ 9K Customers</td>
<td>US $3.10m</td>
<td>Breached</td>
</tr>
<tr>
<td>Taiwan FirstBank</td>
<td>Taiwan</td>
<td>2016-07</td>
<td>Banks ATMs</td>
<td>US $2.2m</td>
<td>Breached</td>
</tr>
<tr>
<td>Qatar National Bank</td>
<td>Qatar</td>
<td>2016-04</td>
<td>Data 1.4GB</td>
<td>NA</td>
<td>Breached</td>
</tr>
<tr>
<td>Bangladesh Central Bank</td>
<td>Bangladesh</td>
<td>2016-02</td>
<td>Bank</td>
<td>US $101m</td>
<td>SWIFT</td>
</tr>
<tr>
<td>HSBC</td>
<td>UK</td>
<td>2016-01</td>
<td>Online Services</td>
<td>NA</td>
<td>DDoS</td>
</tr>
<tr>
<td>RBS &amp; Natwest</td>
<td>UK</td>
<td>2015-07</td>
<td>Online Services</td>
<td>NA</td>
<td>DDoS</td>
</tr>
<tr>
<td>Banco del Austro</td>
<td>Ecuadores</td>
<td>2015-01</td>
<td>Bank</td>
<td>US $12m</td>
<td>SWIFT</td>
</tr>
<tr>
<td>HSBC</td>
<td>Turkey</td>
<td>2014-11</td>
<td>Credit Card Details</td>
<td>NA</td>
<td>Breached</td>
</tr>
<tr>
<td>European Central Bank</td>
<td>Germany</td>
<td>2014-07</td>
<td>20K Records</td>
<td>NA</td>
<td>Breached</td>
</tr>
</tbody>
</table>

3 Methodologies, Data Sources, Tools, and Supplementary Material

A number of methodologies from other domains can be adopted to detect cybercrime in open social data e.g. Text classification, Spam classification, River/Riot modeling, Pandemic disease detection, Event detection/alignment.

Data Sources: We have identified over 30 general cybercrime related open datasets. The majority of them are related to networks e.g. intrusion detection. The second most common type of dataset was related to reviews for spam detection or opinion fraud. Only four of them were related to social media i.e. Twitter. Currently, there is no suitable dataset available that can be used for our specific objectives.

Tools: Although there is no known dataset that can be directly used for this project, there are tools with the help of which we can collect the required data. e.g. Twitter, Reddit, In-House Insight API, IBM X-Force Exchange. As a starting point, we can take hints from previously committed crimes, search data related to that. And then use it for financial institutions in a single jurisdiction.

Supplementary Material: In addition to social media data, digital currency data may also be a helpful tool in detecting suspicious activities. e.g. in recent ransomware attack, the ransom typically must be paid in Bitcoins, so an increase in Bitcoin purchases relative to the usual baseline number of transactions in a specific jurisdiction should provide an indicator of how many individuals paid the ransom to ransomware attack.

4 Conclusion

Open social data can be helpful in detecting cybercrime in financial institutions. Analyzing digital currency stock may supplement open social data. Existing methodologies for studying riots and rumours, sentiment analysis, or the analysis of pandemic disease etc. can be adopted. Currently, there is no suitable dataset available, but a number of suitable APIs may be used to collect relevant data. There is also the possibility of using a combination of two or more platforms to confirm a cyberattack e.g. any tweet on Twitter asking/discussing a compromised account of a specific bank and any discussion in the same time period on a blog or forum e.g. on Reddit.

Acknowledgment

This project is funded by IBM and Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289.

References


Part V

Appendix
Bibliography


