Penetration Testing == POMDP Solving?

Conference Paper · July 2011

Source: arXiv

CITATIONS
7

READS
243

3 authors, including:

Carlos Sarraute
Grandata
84 PUBLICATIONS **292** CITATIONS
SEE PROFILE

Olivier Buffet
National Institute for Research in Computer Science and Control
105 PUBLICATIONS **656** CITATIONS
SEE PROFILE

Working Notes for the 2011 IJCAI Workshop on Intelligent Security (SecArt)

Mark Boddy
Adventium Labs
Minneapolis, MN, USA
mark.boddy@adventiumlabs.org

Stefan Edelkamp
TZI,Universität Bremen
Germany
edelkamp@tzi.de

Yacine Zemali
ENSI de Bourges, LIFO
France
yacine.zemali@ensi-bourges.fr

18 July 2011

Contents

Preface

This is the third in a series of workshops exploring issues at the intersection of Computer Security and Artificial Intelligence. This is a fertile area for research, and has been attracting an increasing amount of interest in both communities. Previous workshops on Intelligent Security have been held at ICAPS 2009, and AAAI 2010. In addition, and more from the Security community than from the AI community, there is a series of workshops, now on their third iteration, being held in conjunction with the ACM Conference on Computer and Communications Security (CCS), and so organized primarily from the Computer Security community. Clearly, this is an active area of research in both communities.

The aim of these workshops is to encourage ongoing dialogue and collaboration, both between the AI and Security communities, and among researchers on both sides, working on similar problems. AI and security is a large and growing area, both for research and for applications. Our increasingly networked world continues to provide new opportunities for security breaches that have severe consequences at the personal level (identity theft, and resulting financial losses), for businesses (theft of intellectual property, or business plans, or costly responses to the theft of customer data), and for governments. Computing and the internet have become crucial parts of the infrastructure of almost every significant commercial or governmental enterprise. Turning off the computers or disconnecting from the network has become tantamount to turning off the power.

The current workshop includes research papers describing the application to computer security of a wide range of techniques in Artificial Intelligence. We have also included a brief description of a live demonstration of the vulnerabilities of current smart-phones and PDAs, which we hope will be of interest to workshop attendees.

A Comparative study of Computational Intelligence in Computer Security and Forensics

Sulaiman Al amro[1], Khalid Aldrawiesh[1] and Ajlan Al-Ajlan[2]

[1]De Montfort University, Software Technology Research Laboratory (STRL)

Leicester, LE1 9BH, UK

{salamro, Khalid}@dmu.ac.uk

[2]Qassim University, Information System Management (ISM)

Al-Melaidah, Buraydah, Qassim, 51452, KSA

aajlan@qu.edu.sa

Abstract

Because of the growing reliance that corporations and government agencies place on their computer networks, the significance of defending these systems from attack cannot be underestimated. Computer security and forensics are crucial in protecting systems from attack or from unauthorized access that might have undesirable consequences, and can help to ensure that a network infrastructure will remain integrated and survive. The use of high quality techniques within a system will provide a better protected system, and will offer clues that might help in any criminal investigation. Techniques such as artificial neural networks, fuzzy logic, and evolutionary computing can be used to help deal with issues associated with computer security and forensics. This paper concentrates on the use of neural networks, fuzzy logic, and evolutionary computation in computer security and forensics. It also examines the hybrids of these three techniques, and considers how they can be applied to computer security and forensics.

Keywords computer security, forensics, neural networks, fuzzy logic, and evolutionary computation.

Introduction

In the years since it was established, Artificial Intelligence (AI) has provided a rich topic of debate for writers and researchers who have produced a large number of articles, books and studies on the subject. While AI is similar to many other systems, it uses unique methods and is remarkable for providing intelligent systems that simulate the nature of life, human beings, etc., in order to achieve improved simulations. Computational Intelligence (CI) includes techniques such as neural networks, fuzzy logic, and evolutionary computation. It also includes other methods that use swarm intelligence, artificial immune systems, etc. However, with the ever-increasing use of computer networks, computer security has become an important issue in modern systems [Ryan et al, 1997]. A single infringement of a network system can cause much damage to an organisation. For this reason, computer security has become vital for protecting systems from attack and from its undesirable outcomes. Lippmann et al (2000) lists four main types of attack which can be harmful to the system:

Denial-of-service (DoS) attack: This type makes the network unable to operate because users are not allowed to access a resource. It does this by making the network full or busy.

Remote to local (R2L) attack: This is an attack in which an attacker who has no access to a resource removes a file or modifies data by acquiring access to the resource.

User to root (U2R) attack: Here, the assaulter begins by hacking the system as a normal user and later tries to obtain root privileges.

Scanning or Probing: In this type, the attacker attempts to find any weakness or susceptibility to attack by scanning a particular resource. Data mining is a common form of this technique.

Eavesdropping, Data Modification, Identity Spoofing, Password-based attacks, Man-in-the-Middle attack, Comprised-key Attack, Sniffer attacks, and Application Layer attacks are also common types of attacks that harm network systems. However, computer and intrusion forensics have been rapidly growing in recent years because of the presence of computers and computer networks everywhere [Mohay et al, 2003]. Computer and intrusion forensics represent an important approach in helping criminal investigations, not only to track crimes that have occurred and recover what has been stolen or

corrupted, but also to bring those criminals to justice. The use of high quality techniques within a system should help to protect a system, and should be able to offer some direction in any criminal investigation. Techniques such as Artificial Neural Networks (ANNs), Fuzzy Logic (FL), and Evolutionary Computing (EC) have been shown to have considerable ability in dealing with issues associated with computer security and forensics.

The paper is organized as follows: Section II states the problem and contribution. Section 3 draws a comparative study in the use of Artificial Neural Networks, Fuzzy Logic, and Evolutionary Computation in computer security and forensics. Section 4 will start with a brief overview of what hybrid computational intelligence is. It will then examine the hybrids of ANN, FL and EC and how they can be used to help computer security and forensics. Section 5 will provide the results of this paper.

Statement of problem and contribution

There are several publications which have studied the use of ANN, FL, and CE in computer security and forensics, but the problem is that these publications have examined them separately. That is, there is no previous publication which has examined the use of the three paradigms and drawn a comparison between them.

The contribution of this paper is to provide and highlight a comparative study of the use of ANN, FL, and CE in computer security and forensics. It also examines the hybrids of these three techniques and how they can be applied to computer security and forensics.

Comparative Study

Artificial Neural Networks

A neural network is an enormous parallel distributed processor composed of uncomplicated processing units, which has an inborn tendency to retain knowledge gained through experience and to be ready for use [Haykin, 2008]. ANNs, also known simply as neural networks, are constructed on the model of biological neural networks and are similar to it, but different terms are used, as shown in Table 1. ANNs work in ways similar to that of the human brain, and the purpose of following the brain in this way is to attempt to replicate its ability to learn.

Neural networks have recently been applied to computer security and are seen as an improvement over expert systems. It is differ from expert systems that use a set of security rules acquired from the knowledge of human experience by its learning ability [Bonifacio et al, 1998]. ANNs have been applied in anomaly detection systems as a substitute for statistical analysis [Ryan et al, 1998]. They can also be used in misused detection in which its learning ability enables it to detect dangerous attacks, in cases where many attackers strike the network at the same time.

Biological	Artificial
Soma	Neuron
Dendrite	Input
Axon	Output
Synapse	Weight

Table 1: Correspondences between biological and artificial neural networks

Neural networks are capable of solving problems related to patterns, using several techniques such as clustering, classification and generalising; they are also able to predict events in the future on the basis of history [Mena, 2003]. These abilities may be useful for forensics, where they can be used to collect evidence after a crime has been committed. However, ANNs have four algorithms which can be helpful for forensics [8]; first, Classification which is a helpful algorithm for investigators by which to determine illegal activities that have been conducted within the system. Next, Clustering may have benefits for analysts who wish to group similar attacks on a particular system or systems. Grouping crimes in this way makes it easier to deal with a new attack which is similar to earlier ones, because these have been investigated and analysed. The third algorithm is generalising which can be used to identify data that have not been seen in the system. As a result, a new type of attack can be identified by the neural generalising algorithm, and this ability can help in the investigation of crimes. Finally, there are forecasting algorithms which can be useful for investigators by providing a list of suspicious people.

Fuzzy Logic

The theory of classical sets depends on the basic idea of a set in which any element is either a member of this set or not [Chen and Pham, 2001]. In most cases of daily life, such precision is inappropriate. By contrast, the fundamental concept of fuzzy sets is that an element has a certain degree of membership of a given fuzzy set, as shown in figure 1. Therefore, the result is not absolutely true or false, but can be true or false to some degree [Negnevitsky, 2005].

One of the most important capabilities that a network system needs to have in its security system is a fast response, and fuzzy logic has this ability when applied to a security system [Gomez and Dasgupta, 2002]. Fuzzy inference is the process of making decisions using fuzzy sets and it has four fundamental steps, namely, fuzzification, knowledge base, decision making and defuzzification [Chen and Pham, 2001].

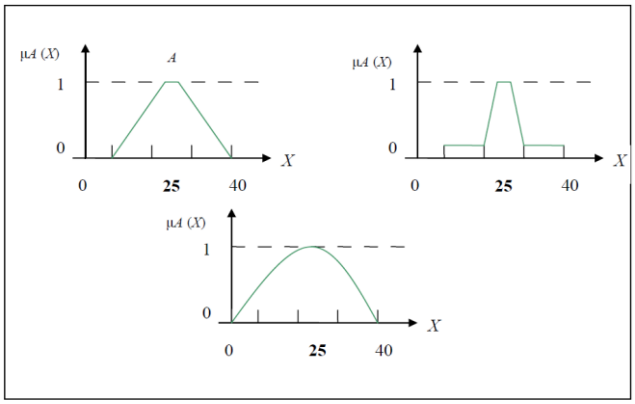


Fig 1: Some ways to represent the fuzzy set of mild temperatures

There are four steps in fuzzy inference [Negnevitsky, 2005]: first, fuzzification is used to take the inputs, to decide the degree of these inputs and determine which fuzzy set they should belong to. Next, knowledge base is used to take the inputs which have been fuzzed, and to apply a set of fuzzy rules to them. Decision making is then used to unify the output from the second step into a single set. Finally, defuzzification is used to remove the fuzziness and transform the output to numerical values so that it can be easily understood by computers and other machines. Figure 2 shows the mechanism of fuzzy inference.

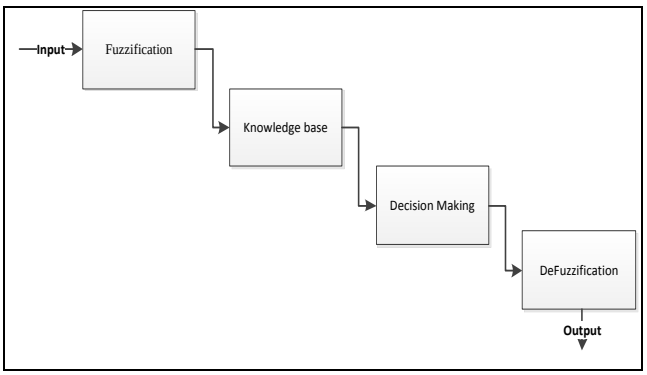


Fig 2: The mechanism of fuzzy inference

Moreover, fuzzy systems have a number of aspects which make them suitable for intrusion detection [Dickerson et al, 2006]. First, they have the ability to combine inputs taken from different sources. Secondly, some encroachments cannot be clearly determined; thus, fuzzy systems are best able to deal with these intrusions. Thirdly, fuzzy systems have the ability to recognise a degree of membership, and this can make alarms sound differently, depending on the attack (low-medium-high). Thus, if the attack is very dangerous, the alarm will sound high, etc. In addition, fuzzy rule-based detection systems provide a perfect method to capture the imprecise tendency of occurrences in the computer network system [Yao et al, 2006]. Indeed, fuzzy logic is perfect if the differences between normal and irregular classes are not specified, because the use of fuzzy spaces enables an object to be a member of different classes at the same time [Gomez and Dasgupta, 2002].

Further, fuzzy logic tools have the ability to help by giving a useful report on how a computer has been utilized [Meyers, 2005]. Furthermore, fuzzy logic is a good tool for forensics investigators when they are uncertain about the file type or the text string they are seeking [Shinder, 2002]. Fuzzy logic, which allows linguistic variables to be presented in mathematical formulas, can help in making the evidence provided by witnesses more significant [Verma, 1997]. Using a fuzzy engine will solve the problem of misspelled words, or even of the use of the wrong word, by relying on the selection of the degree of fuzziness [Johansson, 2003]. Filter I is a fuzzy logic filter that is used to analyse data in files, which can be very useful for forensics [Wai, 2002]. However, it is very difficult to avoid integrity leakage while conducting a forensic investigation, but it can be achieved by using the concept of fuzzy measures, which differ from traditional data analysis by having great robustness and scalability [Pan et al, 2008].

Evolutionary Computation

Two main approaches to EC have been identified by [Fogel, 2006]: theoretical and empirical. The theoretical approach is used to search through algorithms to seek the mathematical truth about them, while the empirical approach is used to examine evolutionary algorithms by statistical means. Alternatively, by creating a population of individuals, appraising their fitness, producing a new population via genetic procedures and repeating this process several times, all EC methods imitate the natural evolutionary processes of selection, mutation and reproduction [Negnevitsky, 2005]. There are three main methods of evolutionary computation: genetic algorithms (GAs), evolutionary strategies (GSs) and genetic programming (GB) [Mitchell, 1998]. Genetic algorithms provide an efficient way to solve problems associated with computer security, especially in detecting intrusions and malicious attacks [Sinclai et al, 1999]. Also, GP has the ability to learn new events in the network and to use computer programs to detect malicious attacks; these properties make it more powerful in the area of computer security and intrusion detection [Crosbie and Spafford, 1995].

Evolutionary computation algorithms have been used efficiently within a forensics system to gather network forensics data and to track novel attacks [Merkle, 2008]. One of the most useful approaches here is to use the capability of genetic algorithms to create a set of optimised rules that recognise unauthorised processes and perform function-based process verification [Bradford and Hu,

2005]. However, a problem has arisen in computer forensics concerning how to determine the type of a file fragment. This problem can be solved by using a genetic algorithm, because it can provide a better classification solution than traditional techniques [Calhoun and Coles, 2008]. Furthermore, genetic algorithms can also be used in other areas of forensics and can save time for investigators.

Hybrids Model of CI techniques

In previous sections, the three types of computational Intelligence, their weaknesses, strengths and abilities have been discussed. This section will consider the combination of these three types for use in computer security and forensics.

A neuro-fuzzy system (NFS) or fuzzy neural network (FNN) is a combination of fuzzy logic and an artificial neural network that has the advantages of both. As shown in Figure 3, an NFS is a multilayered system which can be described as follows [10]. In layer 1, external crisp input data are transmitted to the next layer by each neuron. Layer 2 is called the fuzzification layer, which receives the crisp inputs and determines which fuzzy sets they belong to. Layer 3, which is called the fuzzy rule layer, receives inputs from the fuzzification layer and applies them to a fuzzy rule neuron. Layer 4, the output membership layer, receives inputs from analogous fuzzy rule neurons and combines them with an output membership neuron by applying the fuzzy operation union. Finally, in the defuzzification layer the outputs are defuzzified in order to make them understandable by the computer.

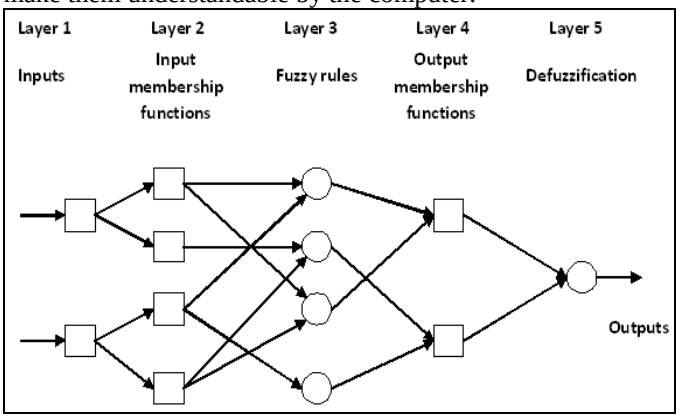


Fig 3: The structure of a neuro-fuzzy system

An NFS is an appropriate approach to security systems, combining the learning ability of ANNs with the human-like ability of FL in order to determine whether activity in the system is normal [Maghooli and Moghadam, 2006]. However, neuro-fuzzy agents can be used in network intrusion detection systems (NIDSs) in order to determine known and unknown behaviours by making use of fuzzy and neural networks [Abouzakhar and Manson, 2006], as shown in figure 5. When the agent's system logic is known, if-then fuzzy rules are used (figure 4a), whereas when the incoming and outgoing network traffics are unknown, a neural network is used (figure 4b). In computer forensics, NFS can be trained differently and tailored to be appropriate for the required response time. They can also be used with a quick search through large databases for fingerprint images [Quek et al, 2001]. They also can be used to help investigators in emotion recognition and speaker verification [Yarmey, 1995].

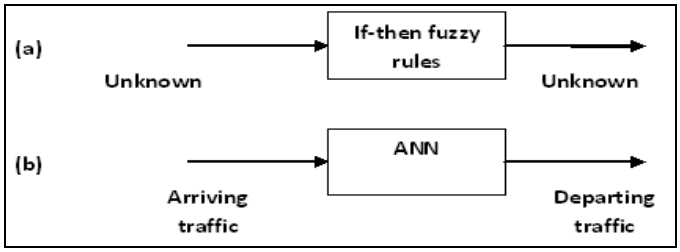


Fig 4 a: Known, if-then fuzzy rules **b**: Unknown, neural network

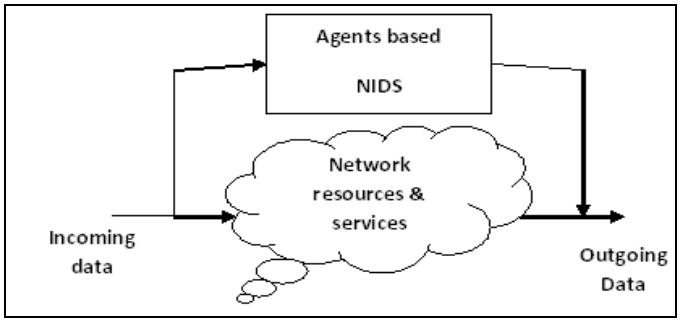


Fig 5: The structure of neuro-fuzzy agents in a NIDS

There are two problems associated with ANNs when used by themselves: these can be solved by combining them with EC. First, they cannot provide optimal solutions; i.e. they are unable to provide a desired solution to a given problem. Secondly, optimal ANN topology cannot be systematically identified: "The 'right' network architecture for a particular problem is often chosen by means of heuristics, and designing a neural network topology is still more art than engineering" [Negnevitsky, 2005, p.285]. As shown in Figure 6, the mechanism for combining an ANN and a GA can be achieved in several steps. First, the problem domain is represented as chromosomes. Secondly, within small intervals, a set of initial weights is randomly selected; this can be represented as a set of gene chromosomes in which each gene matches a weighted link in the network. Thirdly, a fitness function is defined to appraise the performance of the chromosome. The fitness function must evaluate the performance of the neural network. The evaluation of chromosomes is achieved by allocating each weight included in the chromosome to a corresponding link in the network. Next, the genetic operators (crossover and mutation) are selected. The

crossover operator generates a single child with genetic material taken from the two parents by taking the chromosomes of two parents. The genes in the child's chromosomes are represented by the analogous parent's genes. On the other hand, a gene in the chromosome is randomly selected by the mutation operator and a small value is added to every weight in this gene. Finally, the population size is defined; here, the number of weights, the mutation and crossover probabilities and the generation numbers are defined. Several neurons and their interconnections then evaluate the success of the application [Negnevitsky, 2005].

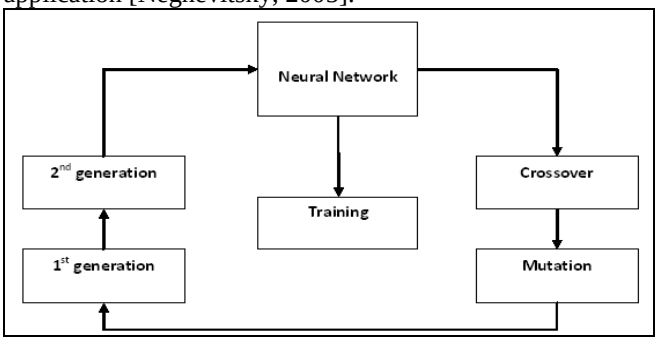


Fig 6: The mechanism of EAANs

The combination of GA/ANN is appropriate for computer security. In particular, this approach is fitting for intrusion detection because it is robust, parallel and non-linear, making it suitable for real-time applications with high precision requirements [Wang et al, 2001]. In addition, this technique can search automatically, and has a robust adaptation and great flexibility. Such an evolutionary neural network (ENN) can be used in an IDS. In computer forensics, ENN can be used to recognise unconstrained fingerprints and also to avoid the trapping of neural networks in a local minimum [Reddy and Reddy, 2004].

Fuzzy systems can also be combined with evolutionary algorithms to produce fuzzy evolutionary systems (FESs). In addition, both computer research and practical applications have shown that fuzzy evolutionary algorithms have the ability to search for optimal solutions more rapidly and efficiently than standard GAs [Xu and Vukovich, 1994]. Indeed, many evolutionary algorithms can be combined with fuzzy systems to provide excellent IDSs [Haghighat et al, 2007]. These evolutionary algorithms can be used separately, or can be combined with each other to achieve the same purpose with higher performance. However, the use of fuzzy genetics in IDSs can be explained in two steps [Fries, 2008]: the first is to use a genetic algorithm to create a subset of optimal communication features; the second step is to use the genetic algorithm to optimise a set of rules which are then used to distinguish abnormal from normal communication behaviours. In forensics, a genetic-fuzzy system can be used to identify types of attack, which is very important in recovering from such attacks. This can be achieved by incorporating the identification of attack types within the system.

Fuzzy systems lack the learning ability of neural networks and the optimal solutions of evolutionary computation, but do have the advantage of providing knowledge in linguistic form [Castellano et al, 2007]. As a result, the combination of evolutionary, neural and fuzzy systems crossbreeds the learning abilities of neural networks with the reasoning mechanism of fuzzy systems and the ability to search for optimal solutions of evolutionary computing. Such networks can be used to construct an efficient intrusion detection system, and each of its components plays a role in making the system more successful [Toosi and Kahani, 2007]. First, features are extracted from the audit data by a number of neuro-fuzzy classifiers in order to group activities in the network. Each input to the system is assigned a degree of membership which determines the dataset to which it belongs. Secondly, a fuzzy inference system is used to decide whether an activity is normal or not. The outputs of the neuro-fuzzy classifiers are mapped to the final output space in order to recognize them. Finally, the structure of the fuzzy sets is optimized by means of genetic algorithms, which are used in order to achieve the best result.

Evolutionary fuzzy neural systems can be applied to help investigators find evidence when a crime has occurred. However, the identification of glass is a significant task in forensic science, where good evidence has a fundamental role in investigations. The abilities of a neuro-fuzzy-genetic system make it an appropriate approach for the identification of glass types. The first step is to construct the neuro-fuzzy classifier that assigns each input to an associated fuzzy set. Next, an initial rule base for the neuro-fuzzy classifier is generated. The final stage is to apply genetic algorithms in order to provide optimal solutions and improve the result. Such a system has the ability to identify the type of glass more accurately than by using just a neuro-fuzzy classifier system [Gorzalczany, and Gradzki, 2000].

The result

In the comparative study of ANN, FL, and EC, and their concepts, abilities, advantages, drawbacks, etc., this paper has produced a result that indicates which technique can be applied to which field of security and forensics. They have been shown to be most efficacious in dealing with issues associated with computer security and forensics because they have the following advantages: first, they are sufficiently intelligent (like humans) to deal with sudden

changes in circumstances. Secondly, they have considerable ability to deal with complex problems when they appear in network systems. Thirdly, they are flexible systems which can acquire the need of specialized applications. Fourthly, they can learn and adjust to changing environments. Last, but not least, they can be hybridised to provide better performance.

ANN takes its idea from the biological neural networks in the human brain, and has the great ability of learning and adjusting itself to new circumstances. In computer security, ANN has shown significant ability in detecting novel attacks and learning them through training. ANN has three particular abilities that make it an appropriate approach for computer forensics, viz, classification, clustering, and generalising. These help investigators by identifying a group of crimes that are similar to each other, distinguishing between legal and illegal behaviour in the system, and recognising attacks that have not been detected in the system.

Fuzzy logic is a human-like reasoning approach that resembles the ambiguity of human daily life. It has the ability to describe core aspects that may not be apparent: i.e. it can provide precise outcomes by its ability to assess degrees of membership and linguistic variables. However, security in itself is imprecise, and therefore a fuzzy system is a proper approach to computer security. There are a great many advantages to using fuzzy systems in security, such as defining attacks by applying a set of fuzzy rules, and exploiting the speed of its response. Many cases in forensics, such as unclear file types, bad handwriting, etc., are uncertain, but fuzzy systems have shown considerable ability in dealing with these kinds of ambiguous cases.

Evolutionary computation derives its concept from the evolution of nature, which takes the advantages gained from one generation and passes them on to the next. It can also choose the best or optimal solution to a problem that has appeared in a system because genetic algorithms and programming are used, and these can be adjusted to restart from a particular point when a problem arises. EC algorithms can also be used to search and gather network forensics from a large amount of network data, and can provide valuable reports on malicious behaviour in such networks. Taking into account computer security and forensics, Table 2 shows the abilities of each EC type and the degree to which they can be applied. For instance, Fuzzy logic is the best type of EC for constructing security rules because it has the if-then fuzzy rules that can specify the normal and abnormal activities of a system. On the other hand, EC and ANN, has less ability to create security rules.

Type / Abitlity	Learning	Classification	Clustering	Generalising	Security rules	Ambiguous Situations	Optimal solution	Recovery
ANN	√√√	√√√	√√√	√√√	√	X	√	X
FL	X	√√	√√	√	√√√	√√√	√	X
EC	X	√	√√	√	X	√	√√√	√√√

Table 2: Shows how ANN, FL, and EC can be applicable to abilities and this table is stated as: √√√ strongly applicable, √√ applicable, √ can be applicable, and X not applicable.

In order to obtain improved performance, hybrid computational intelligence has come to light, as shown in table 3. There are four hybrid CI techniques that have been scrutinised in this paper: Neuro-Fuzzy Systems (NFS), Evolutionary Neural Systems (ENS), Fuzzy Evolutionary Systems (FES) and Evolutionary Neural Fuzzy Systems (ENFS). An NFS is a hybrid of ANN and FL, which has the ability of learning within a neural network, and has the human-like reasoning and thinking design of fuzzy systems. In security, this system uses FL to detect known attacks, and, if certain behaviour is unknown, then ANN is invoked. It can be trained and tailored to be appropriate to the required response time, making it useful for forensic investigation. On the other hand, ENS provides a system that is capable of learning, adjusting, and providing optimal solutions. It also provides an automatic search and has robust adaptation and great flexibility, making it applicable to computer security. In addition to these abilities, this system can provide forensic investigation evidence extremely quickly. Alternatively, FES has the ability to search for optimal solutions more quickly and more efficiently than traditional techniques. It can combine various EC algorithms with FL to achieve a high quality security system. It can also be quickly used to identify the type of attack when a forensic investigation has been initiated. Finally, ENF systems have all the abilities of ANN, FL, and EC in computer security and forensics. It provides a system that has the capability of learning, adjusting to and searching for optimal solutions, and has human-like reasoning and thinking.

Depending on the outcome of this paper, it is apparent that each one of the three types of EC has its own power. However, when choosing the best type to be used to enhance computer security and forensics, one can decide depending on the security polices within the organisation; i.e. one cannot decide which is the best because they all have shown excellent ability when they are used in computer security. For example, when a fast response is extremely important within an organization, a fuzzy logic is the best EC type to be used.

System/ Type	Computer Security	Computer Forensics
Neuro-fuzzy systems	Extract information from multiple sources	Can be differently trained and tailored to be appropriate for the required response time
	Use fuzzy rules to detect known attacks	Can be used with a quick search through large databases for fingerprint images
	Use neural network for unknown attacks	Used to help investigators in emotion recognition
	ANFIS can be used to train and during training can be used to classify patterns	Used efficiently in speaker verification
Evolutionary Neural Systems	Can automatically search for patterns	Can be used to recognise unconstrained fingerprints
	Detect novel attacks because of ability to adapt	Avoiding the trapping of neural networks in a local minimum
	Flexible system	Can be used for face recognition
	Can provide a better classifier in a short time	Provide evidence as fast as possible
Fuzzy Evolutionary Systems	Can combine more than EC algorithms with FL to achieve a higher performance	Can identify the type of attack very rapidly
	Can be used to detect changeable attacks	Used for fingerprint recognition
	Can be retrained to detect novel attacks	Can be used to identify speakers
Evolutionary Neural Fuzzy Systems	Have all the above abilities of ANN, FL, and EC in computer security	Has all the above abilities of ANN, FL, and EC in computer forensics

Table 3: overview of hybrid CI systems in computer security and forensics.

Conclusion and Future Research

It is evident that computer security is a critical issue within organizations, government agencies, etc. It helps to maintain the integrity, secrecy and availability of the network system. However, computer forensics have shown their significant ability to make network infrastructures more integrated and capable of surviving attack. Techniques such as artificial neural networks, fuzzy logic, and evolutionary computing can be used to help deal with issues associated with computer security and forensics. This paper has shown the use of these three techniques in computer security and forensics, their applications, abilities, advantages, drawbacks, etc. We also discovered that the hybrids of the three techniques provide better solutions to the deficiencies than when only one type of CI is used.

Depending on the particular circumstances, fuzzy logic has shown considerable ability in both computer security and forensics. Several questions arise in this research proposal, but one is of particular significance: Is it possible to extend and improve fuzzy logic to meet the changing requirements of computer security and forensics? The most encouraging area for future research would be to investigate how to improve the performance of fuzzy logic to meet the requirements of computer security and forensics. Several studies and articles have shown that the future of fuzzy logic is not determined, i.e. there are no boundaries for searching in this area. Therefore, our future research will focus deeply on how to use fuzzy logic in computer security and forensics.

References

Engelbrecht, A.P. 2007. *Computational intelligence: an introduction*, 2nd edition, Sussex: John Wiley & Sons Ltd.

Ryan, J., Lin, M. and Miikkulainen, R. 1998. Intrusion Detection with Neural Networks. *Advances in Neural Information Processing Systems* 10:72-77.

Lippmann, R., Haines, J., Fried D., Korba, J. and Das, K. 2000. The 1999 DARPA Off-Line Intrusion Detection Evaluation. *Computer Networks*, 34:579-595.

Mohay, G., Anderson, A., Collie, B., Vel, O. and McKemmish, R. 2003. *Computer and Intrusion Forensics*, Massachusetts.: Artech house, Norwood.

Haykin, S. 2008. Neural Networks and Learning Machines, 3rd edition, New Jersey.: Prentice Hall, Inc.

Bonifacio, J., Cansian, A., De Carvalho, A. and Moreira, E. 1998. Neural networks applied in intrusion detection systems. In *Proceedings of the International Joint Conference on Neural Networks*, 205-210. Anchorage, AK: Science Publications.

Ryan, J., Lin, M. and Miikkulainen, R. 1998. Intrusion Detection with Neural Networks. *Advances in Neural Information Processing Systems* 10: 72-77.

Mena, J. 2003. *Investigative Data Mining for Security and Criminal Detection.* New York.: Butterworth Heinemann.

Chen, G. and Pham, T. 2001. *Introduction to fuzzy sets, fuzzy logic, and fuzzy control systems*. Washington, D.C.: CRC Press.

Dubois, D., and Prade, H. 1980. *Fuzzy Sets and Systems Theory and Applications*. Chestnut Hill, MA.: Academic Press.

Negnevitsky, M. 2005. *Artificial Intelligence: A guide to intelligent systems*. Essex.: Pearson Education Limited.

Gomez, J. and Dasgupta, D. 2002. Evolving Fuzzy Classifiers for Intrusion Detection. In *Proceedings of 2002 IEEE Workshop on Information Assurance*, 68-75. West Point, NY.: IEEE.

Dickerson, J.E., Juslin, J., Koukousoula, O. and Dickerson, J.A 2001. Fuzzy Intrusion Detection. In*; IFSA World Congress and 20th NAFIPS International Conference*, 1506-1510. Piscataway, NJ.: IEEE.

Yao, L., ZhiTang, L. and Shuyu, L. 2006. A Fuzzy Anomaly Detection Algorithm for IPv6. Semantics, Knowledge, and Grid, pp. 67-70.

Meyers, M. 2005. Computer Forensics: Towards Creating A Certification Framework. M.Sc. Diss., Centre for Education and Research in Information Assurance and Security, Purdue University, West Lafayette, IN.

Shinder, D. 2002. Scene of the Cybercrime, Rockland, MA.: Syngress Publishing, Inc.

Verma, A. 1997. Construction of offender profiles using fuzzy logic. *An International Journal of Police Strategies & Management* 20: 408-418.

Johansson, C. 2003. Computer Forensic Text Analysis with Open Source Software. M.Sc. Diss., Department of Software Engineering and Computer Science, Blekinge Institute of Technology. Ronneby, Sweden.

Wai, H. 2002. Research in Computer Forensics M.Sc. Diss., Department of Computer Science, Naval Postgraduate School, Monterey, CA.

Pan, L., Savoldi, A., Gubian, B. and Batten, L. 2008. Measure of Integrity Leakage in Live Forensic Context. *Intelligent Information Hiding and Multimedia Signal Processing*: 534-537.

Fogel, D. 2006. Evolutionary Computation Toward a New Philosophy of Machine Intelligence, 3rd edition, Piscataway, NJ.: IEEE Press.

Mitchell, M. 1998. An Introduction to Genetic Algorithms, Cambridge MA.: MIT Press.

Sinclair, C., Pierce, L. and Matzner, S. 1999. An Application of Machine Learning to Network Intrusion Detection. In Proceedings of the 15th Annual Computer Security Applications Conference, 371-377. Washington, DC.: IEEE Computer Society.

Crosbie, M. and Spafford, G 1995. Applying Genetic Programming to Intrusion Detection. In Proceedings of the AAAI Fall Symposium series on Genetic Programming. Menlo Park, CA: AAAI Press.

Merkle, L. 2008. Automated Network Forensics. *In Proceedings of the 2008 GECCO conference companion on genetic and evolutionary computation*, 1929-1932. New York: ACM.

Bradford, P. and Hu, N. 2005. A layered approach to insider threat detection and proactive forensics. In *Annual Computer Security Applications Conference (Technology Blitz)*, Tucson, AZ.: Silver Spring.

Calhoun, W. and Coles, D. 2008. Predicting the types of file fragments. *The Journal of Digital Investigation* 5:14-20.

Maghooli, K. and Moghadam, A.M. 2006. Development of Neuro-fuzzy System for Image Mining. *Fuzzy logic and applications* 3849: 32-39.

Abouzakhar, N. and Manson, J. 2003. Networks security measures using neuro-fuzzy agents. *Journal of Information Management & Computer Security*, 11: 33-38.

Quek, C., Tan, K.B. and Sagar, V.K. 2001. Pseudo-outer product based fuzzy neural network fingerprint verification system. *The official Journal of Neural Networks Society*, 14: 305-323.

Yarmey, A. 1995. Earwitness speaker identification. *The Journal of Psychology 1*: 792-816.

Wang, L., Yu, G., Wang, G. and Wang, D. 2001. Method of evolutionary neural network-based intrusion detection. In *International Conferences on Info-tech and Info-net Proceedings*, 13-18. Menlo Park, CA: IEEE Press.

Reddy, H.R. and Reddy, N.V. 2004. Development of Genetic Algorithm Embedded KNN for Fingerprint Recognition. In: *Asian Applied Computing Conference*, 9-16. Kathmandu, Nepal.: Springer.

Xu, X.Y. and Vukovich, G. 1994. Fuzzy Evolutionary Algorithms and Automatic Robot Trajectory Generation. In: Proceedings of the First IEEE Conference on Evolutionary Computation, 595-600. Florida.: IEEE Computational Intelligence Society.

Haghighat, A.T., Esmaeih, M. and Mousavi, V.R. 2007. Intrusion Detection via Fuzzy-Genetic Algorithm Combination with Evolutionary Algorithms. In: the 6thIEEE/ACIS International Conference on Computer and Information Science, 587-591. Melbourne, Australia.: IEEE Computer Society.

Al amro, S. 2009. Computational Intelligence towards Computer Security and Forensics. M.Sc. Diss., Department of Computer Science, De Montfort University, Leicester, UK.

Fries, T.P. 2008. A fuzzy-genetic approach to network intrusion detection. In: *Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, Atlanta, GA.: ACM.

Castellano, G., Castiello, C., Fanelli, A.M. and Jain, L. 2007. Evolutionary Neuro-Fuzzy Systems and Applications. *Advances in Evolutionary Computing for System Design*, 66: 11-45.

Toosi, A.N. and Kahani, M. 2007. A new approach to intrusion detection based on an evolutionary soft computing model using neuro-fuzzy classifiers. *Journal of Computer Communications*, 30: 2201-2212.

Gorzalczany, M.B. and Gradzki, P. 2000. A neuro-fuzzy-genetic classifier for technical applications. In Proceedings of IEEE International Conference on Industrial Technology 503-508. Goa, India.: IEEE Industrial Electronics Society.

Current Approaches in Algorithmic Intelligence: Efficient Tolerant Pattern Matching with Constraint Abstractions in Description Logic *

Carsten Elfers, Stefan Edelkamp and Otthein Herzog
Center for Communication and Computing Technologies (TZI)
Bremen, Germany
{celfers;edelkamp;herzog}@tzi.de

Abstract

In this paper we consider efficiently matching logical constraint compositions (called patterns) to noisy observations or to ones which are not well described by existing patterns. The major advantage of our approach to tolerant pattern matching is to exploit existing domain knowledge from an ontological knowledge base represented in description logic in order to handle imprecision in the data and to overcome the problem of an insufficient number of patterns. The matching is defined in a probabilistic framework to ensure that post-processing with probabilistic models is feasible.

Additionally, we propose an efficient complete (and optionally approximate) algorithm for this kind of pattern matching. The presented algorithm reduces the number of inference calls to a description logic reasoner. We analyze its worst-case complexity and compare it to a simple algorithm and to a theoretical optimal algorithm.

Introduction

Conventional *pattern matching* methods determine if a given pattern is fully satisfied or not. In real-world domains these pattern matching approaches suffer heavily from uncertainty in the environment in form of typical noise or imprecision in the data. This leads to the natural conclusion that matching patterns must become a matter of degree (Dubois and Prade 1993). Early work on this topic has been done in the context of fuzzy pattern matching in (Cayrol, Farreny, and Prade 1980) and, more recently, in the context of neuro-fuzzy systems, i.e., a combination of fuzzy logic and artificial neural networks (Oentaryo and Pasquier 2009).

Our application area is *tolerant pattern matching* for improved network security within *security information and event management* (SIEM) systems. SIEM systems are monitoring systems, which collect events from several sensors distributed in a computer network. These sensors are known as *intrusion detection systems*, which analyze, e.g., firewall logs, network streams or system health status to detect illegal intruders. The typical huge amount of events collected by a SIEM system is hard to accomplish by users. Therefore, correlation techniques based on pattern matching are typically used to reduce these huge amount of events to the most relevant ones. However, these patterns must be created or adapted regarding the conditions of the network. Due to constantly varying attacks and varying network configurations these patterns must deal with these changing conditions. Nowadays, SIEM systems are essential in most huge business computer networks. Therefore, the following approach can be seen as part of the *Algorithmic Intelligence* initiative of creating intelligent algorithms for real world problems.

In this paper an alternative to fuzzy pattern matching is proposed. A semantically well-defined and intuitive method is presented to calculate the *degree of matching* from patterns and data using a probabilistic composition of the patterns' partially matching constraints. Furthermore, an efficient complete (and optionally approximate) algorithm is presented to calculate the degree of matching. The proposed method differs to fuzzy pattern matching in the way of how the degree of matching is calculated. We use ontological knowledge from *description logic* (and the inference of subsumption) without the need of specifying a characteristic function as usual in fuzzy pattern matching. This decreases the design efforts of appropriate fuzzy patterns, since the proposed method automatically derives the "fuzzyness" from an ontology by still supporting most of the expressivity of description logic.

To our knowledge the most similar approach was proposed by (He et al. 2004) which also uses a similarity measurement in an ontology to handle noise in the matchmaking process. However, in contrast, our approach does not need the specification of similarity weights. Additionally, this approach also describes how to handle disjunctions and negations of conditions (or constraints).

Another approach to use logical background-knowledge has been proposed by (Morin et al. 2009). However, they used first order logic without any abstractions or approximate matching techniques. Instead, this paper is focused on using description logic (and ontologies) to model security background-knowledge (e.g. (Undercoffer, Joshi, and Pinkston 2003), (Li and Tian 2010)).

The paper is structured as follows. First, a brief overview of description logic in the context of constraint satisfaction is given and the tolerant matching problem and the abstraction principle is introduced. Then, we turn to the divide-and-conquer algorithm for tolerant matching and analyze its

*This work was supported by the German Federal Ministry of Education and Research (BMBF) under the grant 01IS08022A.

correctness and complexity. The paper closes with experiments and some concluding remarks.

Tolerant Pattern Matching in Description Logic

The proposed tolerant pattern matching approach is based on constraint satisfaction in description logic. Therefore, a brief introduction of Ontologies in DL is given in the following.

Ontologies and Description Logic

An ontological knowledge representation in description logic addresses the investigation "that knowledge should be represented by characterizing classes of objects and the relationships between them" (Baader and Nutt 2007, p. XV). While convenient databases focus on data persistancy, description logic focuses on modeling generic descriptions, building a formal framework for validation and categorization of data. Description logic is, therefore, applied to model knowledge rather than making data persistent. In other words, it is a formal language which is designed to represent knowledge in a more expressive way than propositional logic by still being decidable (in contrast to first order logic)[1].

Ontologies in description logics consist of concepts, roles and individuals. Concepts represent classes of objects. Roles are binary relations between concepts. Individuals represent instances of classes (Gomez-Perez, Fernandez-Lopez, and Corcho 2004, p. 17). The semantic is defined by the interpretation (an interpretation can be regarded as the corresponding domain) of the set of all concept names, the set of all role names and the set of all names of the individuals (cf. (Baader, Horrocks, and Sattler 2008, p. 140)). A concept C is subsumed by a concept D if for all interpretations $\mathcal{I}$ we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.

Tolerant pattern matching

In this work tolerant pattern matching is realized by successively generalizing the pattern and determining a remaining degree of satisfaction.

Definition 1 (Entity, Constraint, Satisfaction)**.** *An* entity E *is either (a) an individual, (b) a concept, or (c) a variable. A* constraint $\gamma \in \Gamma$ *is defined as* $\gamma = eRe'$ *of a left hand side entity* e*, a right hand side entity* e' *and a relation* R *between these entities. It is assumed that either* e *or* e' *is an individual (fixed by an observation), or a variable. A constraint* γ *is* satisfied *if there exists an interpretation such that relation* R *holds for* e *and* e'*.*

Definition 2 (Partially Matching Pattern, Degree of Matching)**.** *A* pattern p *consists of a set of constraints and logical compositions among them. A* partially matching pattern p *– given the data* $\boldsymbol{x}$ *– is a real valued function with range* $[0, 1]$*. The value of such a function is called* degree of matching *or* matching degree.[2]

$$p(\boldsymbol{x}) = \begin{cases} 1, & \text{if the pattern fully matches} \\ \alpha \in]0,1[& \text{if the pattern matches to degree } \alpha \\ 0, & \text{otherwise} \end{cases}$$

Each constraint in a pattern can be expressed as a query triple in description logic (DL). This allows an easy transformation of patterns into a query language like SPARQL, which can be interpreted by DL reasoners.

Example 1 Let γ_1 be a constraint saying that an observation with the attribute value of `product` is the same as the individual `apple` and γ_2 be a constraint saying that the attribute value of `product` must be the same as the individual `pear`. Furthermore, the pattern p is specified as $\gamma_1 \vee \gamma_2$. Considering a query "Is pattern p satisfied for `product = banana`?" the pattern is transformed to SPARQL as

```
{ns:banana owl:sameAs ns:apple} UNION {
   ns:banana owl:sameAs ns:pear}
```

which is obviously not satisfied since a banana is not the same as an apple or a pear.

However, the pattern from the example describes fruits and if no other pattern is satisfied this may give us a good hint of the kind of data presented. This can be easily achieved by using subsumption, i.e. each pear, banana and apple may be subsumed by a concept called "fruit". E.g. γ_1, γ_2 or both could be abstracted to the condition that the banana must be a fruit instead of being an apple or pear. Each of these abstractions of the pattern will be satisfied. However, the smallest abstraction is desired to maintain as most semantic of the original pattern, i.e., either γ_1 or equivalently γ_2 should be abstracted but not both.

Next, a relation $\geq_g$ (adapted from (Defourneaux and Peltier 1997)) describes that a constraint is an abstraction of another constraint. We say γ_1 is *more general or equal than* γ_2 (noted $\gamma_1 \geq_g \gamma_2$) if for all interpretations $\gamma_2^{\mathcal{I}}$ of γ_2 there exists an interpretation $\gamma_1^{\mathcal{I}}$ of γ_1 such that $\gamma_2^{\mathcal{I}} \subseteq \gamma_1^{\mathcal{I}}$.

In the following we use superscripts to enumerate different levels of specialization. A zero denotes the most abstracted case, while a larger number indicates an increasing specialization, e.g., p^0 is the direct abstraction of p^1. Fig. 1 shows an example of a pattern p^1 with three constraints $\gamma_1^1, \gamma_2^1, \gamma_3^1$ and their direct abstractions $\gamma_1^0, \gamma_2^0, \gamma_3^0$.

To abstract a pattern it is first neccessary to propagate all negations to the leaves (i.e., the constraint triples) by applying De Morgan's rules. The construction of this negational normal form can be done before abstracting a pattern or be applied to all patterns in advance (even to the most specific ones). Moreover, each constraint must be abstracted by the following rules:

- A negated constraint is abstracted to a tautology, since the current model of such a constraint includes all individuals except of the negated one, e.g. γ_3^0 in the example has changed to a tautology. In other words, all individuals

[1] An overview over several knowledge representations can be found in (Kubat, Bratko, and Michalski 1996).

[2] As we will see later, the degree of matching is determined by a fusion function F.

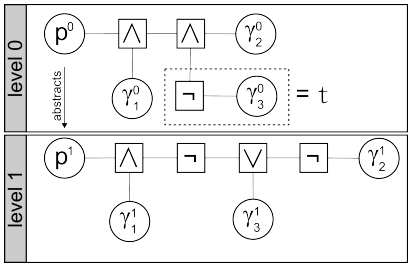


Figure 1: Example of a pattern with constraints and their abstractions.

are valid results except of the negated one. Abstracting this (to extend the set of valid results) must include the negated individual and, therefore, is an tautology.

- If the entity of the abstracted constraint is a concept or an individual this is replaced by a more general concept due to the defintion of $\geq_g$.
- The relation of the abstracted constraint might have to be replaced to ensure that the set of interpretations of the constraint increases, e.g., the identity relation must be exchanged to an appropriate (transitive) subclass relation.

Measuring Abstraction

In the following, a measure $\theta(\gamma^j, \gamma^k)$ for constraints γ^j and γ^k) is assumed to quantify the similarity of an abstracted constraint γ from the original level j to an abstracted level k. A simple example of such a measure is $\theta(\gamma^j, \gamma^k) = 1/(|j-k|+1)$. We write $\gamma^\perp$ for the original constraint on the most specific level $\perp$, and $\theta(\gamma^i)$ for $\theta(\gamma^i, \gamma^\perp)$. Independent of a concrete measurement, such a measurement is assumed to be 1, if the constraint is not abstracted, and decreases, if the constraint is getting more abstract; by still being greater than or equal to 0.

This measurement can also be regarded as a *similarity function*, which says how exactly γ^j describes γ^k, or how similar they are. The properties of a similarity function are assumed to hold for the computation of the degree of matching of a pattern.

Definition 3 (Similarity, extended from (Fanizzi and d'Amato 2006) and (Batagelj and Bren 1995))**.** *A* similarity measure θ *is a real-valued function into* $[0,1]$ *defined by the following properties:*

- $\forall \gamma^j, \gamma^k$*:* $\theta(\gamma^j, \gamma^k) \geq 0$ *(positive definiteness)*
- $\forall \gamma^j, \gamma^k$*:* $\theta(\gamma^j, \gamma^k) = \theta(\gamma^k, \gamma^j)$ *(symmetry)*
- $\forall \gamma^j, \gamma^k$*:* $\theta(\gamma^j, \gamma^k) \leq \theta(\gamma^j, \gamma^j)$ *(identity)*
- $\forall j < k : \theta(\gamma^j, \gamma^{k+1}) < \theta(\gamma^j, \gamma^k)$ *(monotonicity)*

Such similarity function values of the constraints are combined to a matching degree of the pattern by applying some *fusion operator* $F(\theta_1, \ldots, \theta_n)$ similar to fuzzy pattern matching (cf. (Cadenas, Garrido, and Hernndez 2005)). This is necessary to consider the semantics of the logical operators while abstracting the pattern. Due to the absence of a continuous membership function, a different fusion operator (as the proposed multiplication, minimum, average or fuzzy integral for fuzzy pattern matching) is used. Therefore, a probabilistic fusion approach is suggested by using a Bayesian interpretation of the tree of logical operators in each pattern as follows.[3]

Definition 4 (Fusion Function)**.** *The* fusion function F *of pattern* p *is recursively defined with respect to some similarity function* θ *of constraints* γ *composed by logical operators.*

$$
\begin{aligned}
F(\gamma_1^i \wedge \gamma_2^j) &= F(\gamma_1^i) \cdot F(\gamma_2^j) \\
F(\gamma_1^i \vee \gamma_2^j) &= 1 - (1 - F(\gamma_1^i)) \cdot (1 - F(\gamma_2^j)) \\
F(\neg\gamma^i) &= \begin{cases} 1 - F(\gamma^i), & \text{for } i = \perp \\ \beta \cdot F(\gamma^i), & otherwise \end{cases} \\
F(\gamma^i) &= \theta(\gamma^i),
\end{aligned}
$$

where β *is penalty factor to additionally penalize the abstraction of negations, since these may have a greater impact on the result.*

The factor β may depend on the used similarity function and the depth of the ontology.

The fusion function's conjunction/disjunction can be regarded as deterministic AND/OR node. Therefore, the conditional probability table is fully specified. By the assumption of independent constraints the result can efficiently be calculated.

As an example the pattern p^1 from Fig. 1 is represented as $F(p^1) = F(\gamma_1^1) \cdot (1 - (1 - ((1 - F(\gamma_2^1))) \cdot (1 - F(\gamma_2^1))))$

The negation is differently interpreted than in a corresponding Bayesian conditional probability table to ensure that an increasing abstraction leads to a decreasing fusion function. This different interpretation results from the negation of an abstraction being a specialization. However, due to our interpretation of abstraction that the set of solutions (models) of a constraint always increases (which is the complement of negation/specialization) we have to use the complement of the Bayesian interpretation for the negation as well. This leads to the monotonicity of F with respect to θ, which is very useful for finding the most specific abstraction as we will see later.

With these properties a partial order of patterns with respect to the generality of their containing constraints is defined. From this basis it is necessary to find the best matching pattern, i.e., the pattern with the biggest F. This problem can be postulated for a pattern p containing d constraints $\gamma^{x_1}, \ldots, \gamma^{x_d}$ to find a combination of $x_1, \ldots, x_d$ which satisfies the pattern and maximizes F. The solution of interest is in the *Pareto front* of maximum x due to the monotonicity of F with respect to θ and the monotoncity of θ itself (cf. Def. 3). If the level of specialization increases, F increases as well, or – in other words – if any constraint of the pattern is abstracted, F decreases. The next section presents an efficient algorithm to find this Pareto front that includes the most specific solution.

[3] These equations naturally result from a Bayesian network (except of the negation) with conditional probability tables equal to the truth table of the corresponding logical operators.

Example 2 For the case of network security, in the example from Fig. 2 γ_1 might be a constraint indicating that a port-scan event has been received by the SIEM, γ_2 that this event's source IP is extracted from the internal network and γ_3 that the event's source IP belongs to the administrators PC. Regarding the logical compositions, therefore, this pattern matches on a port-scan from a host of the internal network, which is not an administrator PC. Or in other words this pattern should ensure that no PC of the internal network should perform port-scans except of the administrators PC. This pattern is not intuitive due to the multiple negations, however, it is a good example to show the influence of the fusion function. Now considering a ping event instead of a port-scan event, which relates to a very similar situation, since this is also a reconnaissance event from a possible intruder and, therefore, may be structurally very close in the ontology, e.g., ping and port-scan might be summarized by the concept reconnaissance event. The described pattern is not fulfilled, since this pattern was only designed to match port-scans. However, with tolerant pattern matching the pattern can be abstracted with respect to γ_1 in order to include the ping event, e.g., $\theta(\gamma_1) = 0.5$. Therefore, the pattern is assumed to match by 0.5, since all other constraints are satisfied. Now considering that the pinging PC is the administrator's PC, this constraint must be abstracted too, e.g., $\theta(\gamma_3) = 0.1$ (due to $\beta = 0.2$). This results in an overall rating of just 0.05, indicating that the pattern is far from being satisfied. If one of these constraints must further be abstracted, the degree of matching decreases.

This measurement is of course highly dependent to the used similarity function and to an appropriate ontology. There are several alternatives of similarity functions in ontologies, but, unfortunately, tackling this issue appropriately is out of scope of this paper. The reader is encouraged to make further experiments with the fusion function to get an impression of the intuitive measurement of logically composed constraints in a Bayesian way.

Divide-and-Conquer Algorithm

In this section a divide and conquer algorithm is proposed to efficiently search for most specific satisfied patterns, which correspond to the Pareto front of the constraint abstractions. Each level of abstraction of a constraint is represented as one dimension of the search space. The search space $\mathbf{X} = \{0, \ldots, n-1\}^d$ is divided into satisfied elements (satisfied constraint combinations) $\mathbf{X}^+ \subseteq \mathbf{X}$ and unsatisfied elements $\mathbf{X}^- \subseteq \mathbf{X}$ with $\mathbf{X}^+ \cap \mathbf{X}^- = \emptyset$.

Example 3 Fig. 2 gives an example how the algorithm works for the two dimensional case (i.e. for γ_1 and γ_2). At first the middle of the search space is determined, i.e. point $(4, 4)$. Around this point the search space is divided into (in the two dimensional case) four equal sized areas each including the middle and the corresponding border elements. Two of these areas are marked with a gray background the others as area A and area B. The minus sign at $(4, 4)$ indicates that the pattern with γ_1^4 and γ_2^4 is unsatisfied, the circle indicates an inference call to test this satisfaction. Therefore all more specific pattern combinations are omitted in the further recursion, i. e. area A. This method is continued for the gray areas but at first for area B. Area B is divided into four equal sized areas around the middle $(2, 2)$. This is a satisfied match therefore we know that each more abstract pattern than γ_1^2, γ_2^2 is also satisfied, marked as area C which can be omitted in the following. The recursion is continued for the new middle $(3, 3)$. At this point an unsatisfied area can be determined which also affects the currently not investigated gray areas due to we know that from $(3, 3)$ to $(8, 8)$ every solution must be unsatisfied because they are more or equal specific. These temporary results are stored in a list and checked before investigating the gray areas in subsequent recursion steps to omit inference calls for these points.[4]

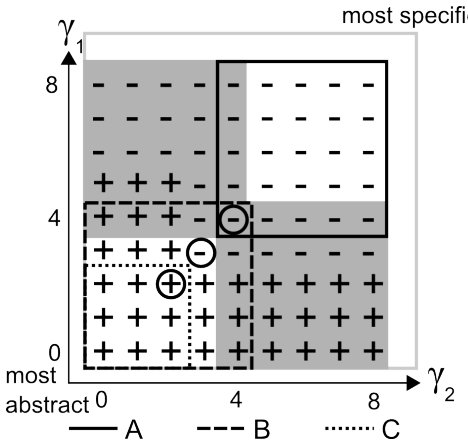


Figure 2: Example of the *Pareto* algorithm.

Please note that the algorithm may be limited in the search space to give approximate results. By increasing the search depth the solution is more and more appropriately approximated.

Algorithm 1 shows the full implementation of the approach. This algorithm is initialized with an empty set of solutions (representing the most specific satisfied patterns) $\mathbf{S}^+$ and $\mathbf{S}^-$ (representing the most abstract unsatisfied patterns). The individual search spaces are specified by a most specific bound (**msb**) and a most abstract bound (**mab**), where **msb** and **mab** are coordinates of the search space. Initially, for all i we have $\mathbf{msb}_i = 0$ and $\mathbf{mab}_i = n$ (to ensure completeness **mab** is located outside of the actual search space). For reasons of simplicity, each constraint is assumed to have an equal amount of specializations/abstractions, however the algorithm is also capable of differing amounts.

In Algorithm 1 we find *Eval*, the call to the reasoner. The other method that enumerates the sublattice structure is called *Hypercubenodes*(**msb**, **mab**) (without **msb**, **mab**

[4] An example of the two dimensional case can be watched at `http://www.informatik.uni-bremen.de/~celfers/programs/FSAT.html`

themselves), formally defined as

$$\bigcup_{i=1}^{2^d-2} \mathbf{msb} \otimes bin(i) + \mathbf{mab} \otimes \overline{bin(i)},$$

where $bin(i)$ denotes the binary representation of a number i, $\overline{bin(i)}$ denotes its (first) complement, and $\otimes$ the component-wise multiplication of two vectors.

Algorithm 1 Pareto

Input: Most specific bound $\mathbf{msb} \in \mathbf{X} = \{x_1, \cdots, x_d\}^d$
Most abstract bound $\mathbf{mab} \in \mathbf{X} = \{x_1, \cdots, x_d\}^d$

1: $\mathbf{m} \leftarrow \lfloor(\mathbf{mab} + \mathbf{msb})/2\rfloor$
2: *// CHECK IF RESULT IS ALREADY KNOWN*
3: **if** $\exists \mathbf{x} \in \mathbf{S}^+$ **with** $(\mathbf{m} \geq_g \mathbf{x})$ **then**
4: $s = 1$
5: **else**
6: **if** $\exists \mathbf{x} \in \mathbf{S}^-$ **with** $(\mathbf{m} \leq_g \mathbf{x})$ **then**
7: $s = 0$
8: **else**
9: $s = Eval(\mathbf{m})$
10: **if** $s = 1$ **then**
11: $\mathbf{S}^+ = \{\mathbf{x} \in \mathbf{S}^+ \cup \{\mathbf{m}\} \mid \forall \mathbf{x}' \in \mathbf{S}^+ \cup \{\mathbf{m}\} : \mathbf{x}' \geq_g \mathbf{x}\}$
12: **else**
13: $\mathbf{S}^- = \{\mathbf{x} \in \mathbf{S}^- \cup \{\mathbf{m}\} \mid \forall \mathbf{x}' \in \mathbf{S}^- \cup \{\mathbf{m}\} : \mathbf{x}' \leq_g \mathbf{x}\}$
14: **end if**
15: **end if**
16: **end if**
17: *// TERMINATION*
18: **if** $\mathbf{mab} = \mathbf{m}$ **then**
19: **return**
20: **end if**
21: **if** $s = 1$ **then**
22: *// EXPLORATION IN MORE SPECIFIC DIRECTION*
23: *Pareto*(**msb**, **m**)
24: **else**
25: *// EXPLORATION IN MORE ABSTRACT DIRECTION*
26: *Pareto*(**m**, **mab**)
27: **end if**
28: *// EXPLORATION IN REMAINING DIRECTIONS*
29: **for each** $\mathbf{h} \in Hypercubenodes(\mathbf{msb}, \mathbf{mab})$ **do**
30: **for** $i = 1$ **to** d **do**
31: $\mathbf{msb}'_i = \max\{\mathbf{h}_i, \mathbf{m}_i\}$
32: $\mathbf{mab}'_i = \min\{\mathbf{h}_i, \mathbf{m}_i\}$
33: **end for**
34: *Pareto*(**msb**′,**mab**′)
35: **end for**

The following definitions express the previous considerations transfered to the d dimensional search space $\mathbf{X}$ which can be interpreted as a coordinate system:

Definition 5 (Domination)**.** *Let*

$$\geq_g = \{(\boldsymbol{x}, \boldsymbol{x}') \in \boldsymbol{X}^2 \mid \forall i\, (\boldsymbol{x}_i \leq \boldsymbol{x}'_i)\}.$$

We say that $\boldsymbol{x} \in \boldsymbol{X}^-$ dominates $\boldsymbol{x}' \in \boldsymbol{X}$ *if* $\boldsymbol{x}' \leq_g \boldsymbol{x}$ *and* $\boldsymbol{x} \in \boldsymbol{X}^+$ dominates $\boldsymbol{x}' \in \boldsymbol{X}$ *if* $\boldsymbol{x}' \geq_g \boldsymbol{x}$.

All more specific patterns than an unsatisfied one are still unsatisfied and all more general patterns than a satisfied one are still satisfied. In other words, we have

$$\forall \mathbf{x} \in \mathbf{X}^-, \mathbf{x}' \in \mathbf{X}.\, (\mathbf{x}' \leq_g \mathbf{x}) \Rightarrow \mathbf{x}' \in \mathbf{X}^-$$

and

$$\forall \mathbf{x} \in \mathbf{X}^+, \mathbf{x}' \in \mathbf{X}.\, (\mathbf{x}' \geq_g \mathbf{x}) \Rightarrow \mathbf{x}' \in \mathbf{X}^+.$$

The algorithm computes the Pareto frontier:

Definition 6 (Pareto Frontier)**.** *The Pareto frontier is the set of extreme points* $\boldsymbol{E} = \boldsymbol{E}^+ \cup \boldsymbol{E}^-$ *with* $\boldsymbol{E}^+ \cap \boldsymbol{E}^- = \emptyset$ *containing each element of* $\boldsymbol{X}^+$ *with no element in* $\boldsymbol{X}^+$ *being more general*

$$\boldsymbol{E}^+ = \{\boldsymbol{x} \in \boldsymbol{X}^+ \mid \forall \boldsymbol{x}' \in \boldsymbol{X}^+ (\boldsymbol{x}' \geq_g \boldsymbol{x})\} \tag{1}$$

and each element of $\boldsymbol{X}^-$ *with no element in* $\boldsymbol{X}^-$ *being more specific*

$$\boldsymbol{E}^- = \{\boldsymbol{x} \in \boldsymbol{X}^- \mid \forall \boldsymbol{x}' \in \boldsymbol{X}^- (\boldsymbol{x}' \leq_g \boldsymbol{x})\}. \tag{2}$$

No element in $\mathbf{E}$ is dominated by another element in this set, i.e., the compactest representation of the set of satisfied / unsatisfied solutions.

Next, we show that Algorithm 1 computes E^+.

Theorem 1 (Correctness and Completeness of Algorithm 1)**.** *The algorithm determines the whole set of satisfied constraints, i.e.,* $\boldsymbol{E}^+ = \boldsymbol{S}^+$.

Proof. **(Correctness)** To show the correctness of the algorithm we ensure that each element of the expected result set $\mathbf{E}^+$ is in the solution set $\mathbf{S}^+$ of the algorithm and, vice versa, i.e., $\mathbf{e}^+ \in \mathbf{E}^+ \Rightarrow \mathbf{e}^+ \in \mathbf{S}^+$ and $\mathbf{s}^+ \in \mathbf{S}^+ \Rightarrow \mathbf{s}^+ \in \mathbf{E}^+$.

Lemma 1. *(*$\boldsymbol{s}^+ \in \boldsymbol{S}^+ \Rightarrow \boldsymbol{s}^+ \in \boldsymbol{E}^+$*)*

If the search is exhaustive (this is shown later) Line 11 implies that $\boldsymbol{s}^+ \in \boldsymbol{S}^+ \Rightarrow \boldsymbol{s}^+ \in \boldsymbol{E}^+$*, since it computes* $\boldsymbol{S}^+$ *as* $\{\boldsymbol{x} \in \left(\boldsymbol{S}^+ \cup \{\boldsymbol{m}\}\right) \mid \forall \boldsymbol{x}' \in \left(\boldsymbol{S}^+ \cup \{\boldsymbol{m}\}\right) : \boldsymbol{x}' \geq_g \boldsymbol{x}\}$*, which is the same as the expected result* $\boldsymbol{E}^+$ *with* $\boldsymbol{S}^+ \cup \{\boldsymbol{m}\} \subseteq \boldsymbol{X}^+$.

Lemma 2. *(*$\boldsymbol{e}^+ \in \boldsymbol{E}^+ \Rightarrow \boldsymbol{e}^+ \in \boldsymbol{S}^+$*)*

We investigate four conditions under which an element is inserted into (and kept in) the solution set of the algorithm $\boldsymbol{S}^+$ *in Line 11. These conditions, directly derived from the algorithm, are as follows*

1. *Line 10 implies that each element from* $\boldsymbol{S}^+$ *must be contained in* $\boldsymbol{X}^+$ *which is exactly the same condition as in definition of* $\boldsymbol{E}^+$.
2. *The following assumption, derived from Lines 3 and 5, must hold for* $\boldsymbol{e}^+$ *to be inserted into* $\boldsymbol{S}^+$

$$\neg \exists \boldsymbol{x}' \in \boldsymbol{S}^+.\, \left(\boldsymbol{e}^+ \geq_g \boldsymbol{x}'\right).$$

This condition is not fulfilled if an equivalent solution $\boldsymbol{e}^+$ *is already in the set* $\boldsymbol{S}^+$ *or if* $\boldsymbol{e}^+$ *dominates another element from* $\boldsymbol{S}^+$*. In both cases* $\boldsymbol{e}^+$ *is not inserted into the result set* $\boldsymbol{S}^+$.

3. *The next statement, derived from Line 6 and 8, is*

$$\neg \exists \boldsymbol{x}' \in \boldsymbol{S}^-.\, \left(\boldsymbol{e}^+ \leq_g \boldsymbol{x}'\right). \tag{3}$$

This condition is always fulfilled, since we consider the case that $\boldsymbol{e}^+ \in \boldsymbol{X}^+$ *and from eqn. we know that this implies that* $\boldsymbol{x}' \in \boldsymbol{X}^+$ *which cannot be the case since* $\boldsymbol{x}' \in \boldsymbol{S}^- \subseteq \boldsymbol{X}^-$.

4. *Line 11 is does not drop solutions because for all* $\boldsymbol{m} \in \boldsymbol{X}^+$ *we have eqn. 1.*

Analogically, the proof can be made for $\mathbf{E}^-$. □

Proof. **(Completeness)**
From Line 22 to 39 we obtain that the recursion is omitted for

- $\{\mathbf{x} \in \mathbf{X} \mid \mathbf{msb} \leq_g \mathbf{x} \leq_g \mathbf{m}\}$ if $\mathbf{m} \in \mathbf{X}^-$ and for
- $\{\mathbf{x} \in \mathbf{X} \mid \mathbf{m} \leq_g \mathbf{x} \leq_g \mathbf{mab}\}$ if $\mathbf{m} \in \mathbf{X}^+$.

This does not affect the set of solutions due to the definition of domination and the definition of E that there should not be any value in the result set that is dominated by another element. Note that **m** has already been checked by the algorithm at this point.

The remaining space under investigation is getting smaller in each recursion path until **m** is getting equal to **mab** (the termination criteria from Line 18). This is only the case if each edge of the space under investigation is smaller or equal one. This can be derived from Line 1 of the algorithm. At some time in the recursion the space of possible solutions is divided into a set of spaces with edges of the length one or less by still covering the whole space of possible solutions as previously shown. Further if any point of such a smallest area is a possible solution (these are the corners), this point is under investigation in another space due to the recursive call with overlapping borders (cf. Lines 24, 27 and 35) except of the borders of the whole search space at the specific borders due to there is no **mab** of any area including these specific border elements, e.g., there is no **mab** for the one element area $(8, 8)$ in the example from Fig. 2. For this border case the algorithm is called with a lifted **msb** to ensure that the unlifted specific bound is included in some smallest (one element) area as **mab** visualized as a light gray border in Fig. 2. Therefore, each element of the search space which is a possible solution is investigated as a **mab** in some recursive path.

□

After computing the pareto front, the fusion function F is used on the remaining set of candidates to identify the most specific matching pattern abstraction.

Complexity Considerations

The worst-case running time is dominated by the number of calls to the reasoner. So we distinguish between the number of recursive calls $T(n)$ and the number of inference calls $C(n)$ (for the sake of simplicity, we assume $n_1 = \ldots = n_d$ and $n = 2^k$). Of course, a trivial algorithm testing all possible elements in S induces $C(n) = T(n) = O(n^d)$. We will see that the algorithm *Pareto* is considerably faster.

With $\lg n$ we refer to the dual logarithm $\log_2 n$, while $\ln n$ refers to the natural logarithm $\log_e n$.

Recursive Calls

Let us first consider the 2D case. The number of calls of the divide-and-conquer algorithm in a 0/1 $(n \times n)$ matrix is bounded by

$$T(k) = \sum_{i=0}^{k} 3^i = \left(3^{k+1} - 1\right)/2$$

Assuming $n = 2^k$ we have

$$T(n) = \left(3^{\lg n} - 1\right)/2 = \left(n^{\lg 3} - 1\right)/2 = O(n^{1.5849})$$

Let us now consider the 3D case. The number of calls of the divide-and-conquer algorithm in a 0/1 $(n \times n \times n)$ (hyper-) cube is bounded by

$$T(n) = \left(7^{\lg n} - 1\right)/2 = \left(n^{\lg 7} - 1\right)/2 = O(n^{2.8073})$$

We also see that for larger dimensions d the complexities $O(n^{\lg(2^d-1)})$ rise. In the limit for large d the exponent to n converges to d.

Inference Calls

Again, we first consider the 2D case. We observe that the structure of the recursion corresponds to find a binary search to the SAT/UNSAT boundary. The recursion depth is bounded by $\lg n$. Therefore, the worst-case number of calls to the reasoner of the divide-and-conquer algorithm in a 0/1 $(n \times n)$ matrix is defined by

$$C(n) = 2C(n/2) + O(\lg n).$$

The $O(\lg n)$ term is due to the binary search. In the worst case the boundary between SAT and UNSAT cells is in the middle, where one quarter of SAT and one quarter of UNSAT elements are omitted.

Using the Akra-Bazzi theorem (Akra and Bazzi 1998) (a generalization to the well-known master theorem), the above recursion can be shown to reduce to $C(n) = O(n)$ as follows.

The Akra-Bazzi theorem (for $k = 0$) states that for recurence equation $T(n) = g(n) + aT(n/b)$ with $a = b^p$ we have the following closed form

$$T(n) = O\left(n^p \cdot \left(1 + \int_1^n g(u)/u^{p+1}\, du\right)\right).$$

Here, $g(n) = \lg n = \ln n / \ln 2$ and $a = b = 2$ so that $p = 1$ and

$$\begin{aligned} T(n) &= O\left(n + n \cdot \int_1^n \ln(u)/u^2 du\right) \\ &= O\left(n + n \cdot [-\ln u/u]_1^n\right) = O(n). \end{aligned}$$

Let us now consider the 3D case. The number of calls to the reasoner in the divide-and-conquer algorithm in a 0/1 $(n \times n \times n)$ (hyper-) cube is bounded by

$$C(n) = 6C(n/2) + O(\lg n).$$

Again the $O(\lg n)$ term is due to the binary search. In the worst case the boundary between SAT and UNSAT cells is in

the middle, where one eigth of SAT and one eigth of UNSAT elements are neglected.

The recursion yield non-linear time complexity. Here, $g(n) = \lg n$ and $a = 6, b = 2$ so that $p = 2.5849$ and

$$\begin{aligned} T(n) &= O\left(n^{2.5849} \cdot \left(1 + \int_1^n \ln(u)/u^{3.5849}\, du\right)\right) \\ &= O\left(n^{2.5849} \cdot (1 + 0.3868 \ln 1 + 0.1496 - \right. \\ &\qquad \left. (0.3868 \ln n + 0.1496/n^{2.5849}))\right) \\ &= O(n^{2.5849}). \end{aligned}$$

In the general case for d dimensions we have $g(n) = \lg n$ and $a = 2^d - 2, b = 2$ so that $p = \lg(2^d - 2)$ and

$$\begin{aligned} T(n) &= O\left(n^p \cdot \left(1 + \int_1^n \ln(u)/u^{p+1}\, du\right)\right) \\ &= O\left(n^p \cdot (1 + p \ln 1 + 1 - (p \ln n + 1/(p^2 n^p)))\right) \\ &= O(n^p) = n^{\lg(2^d - 2)}. \end{aligned}$$

We see that for larger dimensions d the complexities $O(n^{\lg(2^d-2)})$ rise. In the limit for large d the exponent to n converges to d.

Evaluation

We have evaluated the efficiency of the algorithm with respect to the number of inference calls.

In Fig. 3 two tolerant matching algorithms and the result of a perfect guessing algorithm (a lower bound) are visualized. It is assumed that the lower bound algorithm checks exactly the Pareto border of satisfied and unsatisfied elements. Therefore, the best possible algorithm needs at least $|\mathbf{S}^+| + |\mathbf{S}^-|$ inference calls.

The proposed divide-and-conquer algorithm *Pareto* with pruning the recursive calls as in Algorithm 1 – but without using the lists S^+ and S^- – is called **Pareto-0**. This is the first efficient algorithm one might think of. The proposed algorithm is visualized as **Pareto** and the lower bound as **LOWERBOUND** for the two dimensional case in Fig. 3 (no log-scale) and for the four dimensional case in Fig. 4 (log-scale).

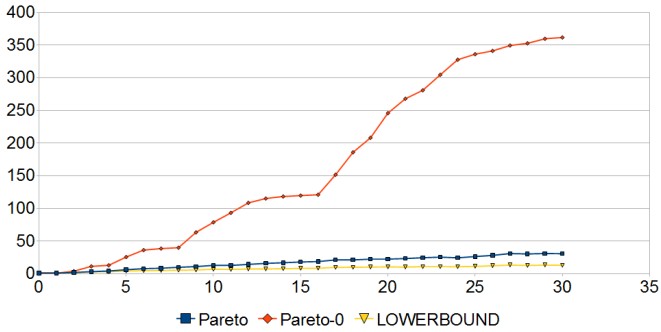


Figure 3: DL Reasoner calls in the two dimensional case. The x-axis represents the amount of possible abstractions and the y-axis the amount of reasoner calls.

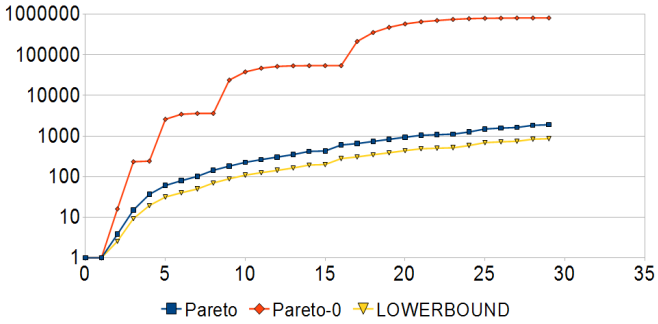


Figure 4: DL Reasoner calls in the four dimensional case with logarithmic scaling. The x-axis represents the amount of possible abstractions and the y-axis the amount of reasoner calls.

Both figures show that the inference calls of **Pareto** is near to the optimal lower bound **LOWERBOUND** and considerably better than the typical divide-and-conquer algorithm **Pareto-0** in both 2D and 4D. These results are reasonable for the proposed pattern matching algorithm due to the depth of an ontology being typically smaller than 30 and the patterns having typically a small amount of constraints. Furthermore, not all constraints share variables, so that some of the constraints can be checked independently to the full pattern by using a simple binary search.

It can be seen that the amount of pareto results, which is around the half of the **LOWERBOUND** value, is very small. For this set the degree of matching must be computed with respect to the fusion function to find the most optimal solution out of the set of Pareto-optimal solutions. This search can be done without to call the DL reasoner, since we already know that these solutions are satisfied.

Conclusion and Outlook

In this paper we have shown how to use ontological background knowledge in the form of description logic to overcome the problem of noisy and imprecise data in the method of pattern matching (of logical terms). The proposed tolerant pattern matching can even address erroneous or missing patterns by successively abstracting them.

This kind of pattern matching suffers from the necessity to call a description logic reasoner several times, which might be - depending on the size and structure of the knowledge base - very time consuming. Therefore, this paper considered an efficient algorithm to reduce the amount of these inference calls. It has been shown that the presented algorithm is correct and complete. Moreover, it can be parameterized to have the ability to infere approximate results, yielding a number of inference calls near to an optimal lower bound.

By maintaining temporary result lists in form of a dictionary to prune the search, the algorithm has been shown to outperform a divide-and-conquer algorithm.

References

[Akra and Bazzi 1998] Akra, M., and Bazzi, L. 1998. On the solution of linear recurrence equations. *Computational Optimization and Applications* 10(2):195–210.

[Baader and Nutt 2007] Baader, F., and Nutt, W. 2007. *The Description Logic Handbook*. Cambridge University Press.

[Baader, Horrocks, and Sattler 2008] Baader, F.; Horrocks, I.; and Sattler, U. 2008. *Handbook of Knowledge Representation*. Elsevier.

[Batagelj and Bren 1995] Batagelj, V., and Bren, M. 1995. Comparing Resemblance Measures. *Journal of Classification* 12(1):73–90.

[Cadenas, Garrido, and Hernndez 2005] Cadenas, J. M.; Garrido, M. C.; and Hernndez, J. J. 2005. Heuristics to model the dependencies between features in fuzzy pattern matching. In *Proceedings of the Joint 4th Conference of the European Society for Fuzzy Logic and Technology*.

[Cayrol, Farreny, and Prade 1980] Cayrol, M.; Farreny, H.; and Prade, H. 1980. Possibility and necessity in a pattern-matching process. In *Proceedings of the 9th International Congress on Cybernetics*, 53–65.

[Defourneaux and Peltier 1997] Defourneaux, G., and Peltier, N. 1997. Analogy and abduction in automated deduction. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 216–221.

[Dubois and Prade 1993] Dubois, D., and Prade, H. 1993. Tolerant fuzzy pattern matching: An introduction.

[Fanizzi and d'Amato 2006] Fanizzi, N., and d'Amato, C. 2006. A similarity measure for the ALN description logic. In *Italian Conference on Computational Logic (CILC)*, 26–27.

[Gomez-Perez, Fernandez-Lopez, and Corcho 2004] Gomez-Perez, A.; Fernandez-Lopez, M.; and Corcho, O. 2004. *Ontological Engineering*. Springer.

[He et al. 2004] He, Y.; Chen, W.; Yang, M.; and Peng, W. 2004. Ontology based cooperative intrusion detection system. In Jin, H.; Gao, G.; Xu, Z.; and Chen, H., eds., *Network and Parallel Computing*, volume 3222 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 419–426.

[Kubat, Bratko, and Michalski 1996] Kubat, M.; Bratko, I.; and Michalski, R. 1996. *A Review of Machine Learning Methods*. John Wiley & Sons.

[Li and Tian 2010] Li, W., and Tian, S. 2010. An ontology-based intrusion alerts correlation system. In *Expert Systems with Applications*, volume 37, 7138 – 7146.

[Morin et al. 2009] Morin, B.; M, L.; Debar, H.; and Ducass, M. 2009. A logic-based model to support alert correlation in intrusion detection. *Information Fusion* 10(4):285 – 299. Special Issue on Information Fusion in Computer Security.

[Oentaryo and Pasquier 2009] Oentaryo, R. J., and Pasquier, M. 2009. A novel dual neuro-fuzzy system approach for large-scale knowledge consolidation. In *Proceedings of the 18th international conference on Fuzzy Systems (FUZZ-IEEE)*, 53–58.

[Undercoffer, Joshi, and Pinkston 2003] Undercoffer, J.; Joshi, A.; and Pinkston, J. 2003. Modeling computer attacks: An ontology for intrusion detection. In *In: 6th International Symposium on Recent Advances in Intrusion Detection*, 113–135. Springer.

Hypergraph Clustering for Better Network Traffic Inspection

Li Pu, Boi Faltings
Artificial Intelligence Laboratory, EPFL/IC/LIA, Lausanne, Switzerland
{li.pu,boi.faltings}@epfl.ch

Abstract

Networked computing environments are subject to configuration errors, unauthorized users, undesired activities and attacks by malicious software. These can be detected by monitoring network traffic, but network administrators are overwhelmed by the amount of data that needs to be inspected. In this paper, we describe how clustering can be used for this application to reduce the amount of data that has to be inspected. Rather than a system that attempts to directly detect malicious software and user, we propose a data-mining component to group the open ports and users in the network and let a human system administrator analyze the results. With empirical study, we show that the behaviors of softwares and users are very different. They should be clustered by the appropriate clustering algorithm accordingly.

Introduction

Networked computing environments are subject to configuration errors, undesired activities and attacks by malicious software. These can be detected by monitoring network traffic, but network administrators are overwhelmed by the amount of data that needs to be inspected. Filtering techniques are widely used by administrators to pick out the unusual traffic (Chandola, Banerjee, and Kumar 2009). But this approach requires the pre-defined filters, so it might miss some unknown anomalies. To deal with this problem, more intelligent techniques need to be adopted. Our work aims to develop a clustering strategy based on hypergraph model for the administrator so that the network traffic can be efficiently inspected with a concise list.

The real-time enterprise desktop monitoring software developed by Nexthink (http://www.nexthink.com) continuously monitors network traffic and application activities (processes) on all desktops in an enterprise network (Figure 1). Unlike traditional system logs, the monitoring software called Nexthink Collector is installed on the client side rather than the server side. This feature allows complete record of TCP/UDP connections, which includes not only the common 5-tuple (source IP, destination IP, source port, destination port, and protocol) but also the information of the application that initialized the connection (e.g. application name, application version, the user who is using the application, etc.). In this work, we empirically study two problems with the nexthink dataset, namely network service identification and user affiliation identification. Both problems are modeled with hypergraph where the goal is a partition of vertices.

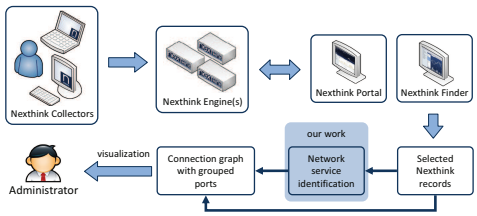


Figure 1: Nexthink tool architecture.

With Nexthink Finder the administrator can select the part of records of particular interest (by smart filters) and generate a connection graph for inspection. In the current version, connections are grouped by the port number range (e.g. 1-1000, 1001-2000, etc.), which is obviously not a good grouping strategy because in each port number range there could be ports of different functionalities. Our work aims to provide a better grouping strategy of the ports based on network service, i.e. a set of ports that serves for the same functionality. In the network service identification problem, clusters of special interest are usually small (e.g. malware), so the clustering approach that is able to isolate small clusters is more preferable than the approach that generates good average performance.

The similar idea is applied to identify the affiliation of a user. Users from the same department tend to use the same set of applications to access similar destinations since they do similar works. The affiliation information can be directly obtained from the profile of the user, but there might exist latent "affiliations" that reflect some unknown work profiles. Thus an unsupervised learning result of user groups is particularly interesting from the perspective of network management. The identified user groups can be used for generating concise connection graph in Nexthink Finder, as well as profiling work flows and improving security rules.

The above two problems in fact have different natures. Although they take the same type of evidence which consists of the connections from application to destination, the

structures of the induced hypergraph model are very different. The structure of hypergraph that involves software behaviors is less randomized than that of a hypergraph which involves human behaviors. Depending on the structure of the hypergraph, we should choose the appropriate clustering algorithm accordingly. A measure called *hyperedge affinity* is proposed in the paper for studying the structure of the hypergraph.

Our previous work has isolated malware in the network traffic (Pu et al. 2010), while in this paper we compare a broader range of state-of-the-art algorithms with the applications of clustering network traffic, including a new task that identifies the affiliation of users. We also study the strategy of choosing suitable clustering algorithm with hyperedge affinity before any partition result is produced.

Related work

There are several published works that address similar problems in the context of networks. Karagiannis et al. developed an approach called BLINC to classify network flows according to the traffic patterns at three different levels (Karagiannis, Papagiannaki, and Faloutsos 2005). This work is similar to our work in the sense that it tries to associate hosts with applications rather than solely classify applications. But in our work the information about applications is in a finer granularity and more complete. Bayer et al. developed a scalable clustering algorithm for grouping malware execution traces by computing the approximated nearest neighbors with a technique called locality sensitive hashing (Bayer et al. 2009). Perdisci et al. tested different clustering algorithms (single-linkage hierarchical clustering, complete-linkage hierarchical clustering, and x-means) on the malware signature generation problem (Perdisci, Lee, and Feamster 2010). The single-linkage hierarchical clustering algorithm produces the best result.

Problem Statement

From the Nexthink records, five types of entities are considered: *connections* that represent TCP/UDP sessions; *applications* that denote processes of the same executable filename; *destinations* are servers of an unique IP address that open some ports to the applications; *ports* are combinations of port type (TCP or UDP) and port numbers; and *users* who are using the applications to access destinations. The source IP address and source port information in the original records are omitted because we treat the same application on different client machines equally. Then a *network service* is defined as a cluster that consists of one or more ports. The goal of the network service identification problem is to extract network services in the dataset and identify the machines/applications which provide/use those services.

The only evidence we extract from the original records states that if more than one connection is associated with the same application-destination pair, the services of the ports of the connections are likely to be the same. This simple evidence generating rule helps to merge similar ports into one cluster and actually defines a group of subsets where each subset can be named by the application-destination pair. In the end the whole dataset is transformed into subsets of ports. These subsets are called *set evidence*. It can be observed that the ports are not always used for the registered purpose as shown in IANA (Internet Assigned Numbers Authority), but we find that the functionality of one port is quite stable within the enterprise network. So it is reasonable to assume that one port only belongs to one service.

The above process of extracting set evidence from raw data assumes the correlation between the service and a collection of distinguishable application-destination pairs. Although this assumption is quite tenable in our dataset, it could be circumvented by malware that mimics the behavior of normal software such that malicious ports are identified as the same service as normal ports. But in order to hide itself, the malware has to know the ports of other services. Thus we can offset the impact of this adversarial behavior by excluding the well known ports to make it harder to hide in a known service. As a transport layer approach, the assumption is also vulnerable to the attacks that directly use well known static ports. But this may cause port conflicts and easy-identifiable anomaly.

The evidence used for user affiliation identification is similar to that of the network service problem. Two or more users are likely to be in the same affiliation if they use the same application to access the same destination. Since network users usually spend a lot of time on destinations outside the enterprise network, all the IP addresses (not only local IP addresses) are included as destinations in the set evidence. We also reduce the number of destinations by replacing the IP addresses with its corresponding net-names and/or domain names. This procedure usually reduces the raw IP addresses to less than 10% destinations.

Hypergraph model

Every port and every user in the processed data can be expressed as a feature vector whose length equals the number of set evidence. The entry in the feature vector is set to 1 if the port or user is contained in the set evidence. By computing the distances between the feature vectors of ports or users, one can apply any distance-based clustering algorithm (e.g. k-means) to the processed data. In this work, however, we use the hypergraph model to represent the co-occurrence relationship among the ports/users.

A hypergraph $\mathcal{H} = \{\mathcal{V}, \mathcal{E}\}$ consists of a finite *vertex set* $\mathcal{V}$ and a finite *hyperedge set* $\mathcal{E}$. Each hyperedge $e \in \mathcal{E}$ is a subset of $\mathcal{V}$, i.e. $e \subseteq \mathcal{V}$. Each hyperedge e is associated with a positive real-valued weight $w(e)$. Let W denote the set of weights. We call $\mathcal{H} = \{\mathcal{V}, \mathcal{E}, W\}$ a *weighted hypergraph*. A hyperedge e is *incident* with a vertex v if $v \in e$. The weighted degree of a vertex is $deg(v) = \sum_{v \in e, e \in \mathcal{E}} w(e)$. The degree of a hyperedge is $deg(e) = |e|$. For the network service problem, the set of all ports is taken as the vertex set, and each piece of set evidence is translated to a hyperedge. For the user affiliation problem, the set of all users is taken as the vertex set, and each piece of set evidence forms a hyperedge. In this paper, the weights of all hyperedges are set to 1 for simplicity.

From a weighted hypergraph $\mathcal{H}$, we can build a simple graph $\bar{\mathcal{H}} = \{\mathcal{V}, \bar{\mathcal{E}}\}$ with edge set $\bar{\mathcal{E}}$ where an edge is added

between $v_i, v_j \in \mathcal{V}$ if v_i and v_j belong to the same hyperedge in $\mathcal{E}$. Each edge $\bar{e} = (v_i, v_j) \in \bar{\mathcal{E}}$ has weight $w(\bar{e}) = \sum_{v_i,v_j \in e} w(e)/deg(e)$. This simple graph $\bar{\mathcal{H}}$ is called the *induced graph* of $\mathcal{H}$.

A *k-way partition* $\mathcal{T}$ consists of k clusters where each cluster $t \subseteq \mathcal{V}$, $\bigcup_{t\in\mathcal{T}} t = \mathcal{V}$, and $\forall t_1, t_2 \in \mathcal{T}$, $t_1 \neq t_2$, $t_1 \cap t_2 = \emptyset$. For cluster t, let t^c denote the compliment of t. The volume of t is $Vol(t) = \sum_{v\in t} deg(v)$. A hyperedge e is said to be incident with a cluster t if $e \cap t \neq \emptyset$.

Extracting Clusters with Hypergraph Min-Cut

Unsupervised learning on simple graphs or hypergraphs is a well-studied topic with many applications. Various techniques can be used to form the partition according to the clustering fitness measures, e.g. graph cut (Schaeffer 2007), hypergraph cut (Karypis and Kumar 1999), graph modularity (Brandes et al. 2007), kernel-defined measure (Kulis et al. 2009) , etc. The idea of cut-based approaches is to search for the partition that minimizes the cost of (hyper)edges on the partition boundary. There are various definitions of the cost of (hyper)graph cut, which lead to different min-cut approaches. We first introduce two existing definitions of hypergraph cut and then propose a hierarchical clustering algorithm based on a non-pairwise hypergraph cut. The hypergraph modularity is used to determine the optimal number of clusters.

Hypergraph cut

Let $D(e) = \{t|e\cap t \neq \emptyset, t \in \mathcal{T}\}$ denote the incident clusters of e. For each cluster t, let $A(t) = \{e|e \cap t \neq \emptyset, \forall t' \neq t, e \cap t' = \emptyset\}$ denote the set of hyperedges that are only incident with t, $C(t) = \{e|e \cap t \neq \emptyset\}$ denote the set of hyperedges that are incident with t, and $B(t) = C(t) \setminus A(t)$ denote the set of hyperedges that are not only incident with t but also with some other cluster.

By (Karypis and Kumar 1999; Xiao 2010), the k-cut of a hypergraph is the weighted sum of hyperedges that have to be removed to produce k disconnected parts, i.e.

$$kCut(\mathcal{H}, \mathcal{T}) = \sum_{e\in\cup_t B(t)} w(e). \quad (1)$$

By (Zhou, Huang, and Scholkopf 2007), the cut volume of t is $Vol_2(\partial t) = \sum_{e\in B(t)} w(e)|e\cap t||e\cap t^c|/deg(e)$ where we use subscript 2 because $Vol_2(\partial t)$ actually computes the pairwise connections of all vertices in the cut. The *normalized hypergraph cut* is then defined as

$$NHC_2(\mathcal{H}, \mathcal{T}) = \sum_{t\in\mathcal{T}} Vol_2(\partial t)/Vol(t), \quad (2)$$

where the normalizer $Vol(t)$ introduces a penalty on extremely small clusters and encourages more balanced cluster sizes. One can easily show that the normalized hypergraph cut equals to the normalized simple graph cut on the induced graph $\bar{\mathcal{H}}$ from the original hypergraph $\mathcal{H}$. Our previous work has shown the connection between NHC_2 cut and the objective of Markov logic in a network service identification problem (Pu et al. 2010).

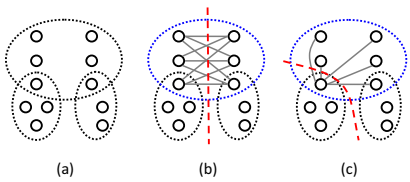


Figure 2: Different definitions of hypergraph cut.

On the other hand, from that fact that $\cup_{t\in\mathcal{T}} B(t) = \{e|D(e) > 1\}$, it is reasonable to directly count the number of pieces of the cut hyperedge as the cost of the cut. Because each hyperedge must be incident with at least one cluster, it only adds a constant (the sum of hyperedge weights) to the total cost if we count the hyperedge pieces in $C(t)$ instead of $B(t)$. For cluster t, we define another cluster cut volume $Vol_0(\partial t) = \sum_{e\in C(t)} w(e)$, which leads to the following definition of hypergraph cut,

$$HC_0(\mathcal{H}, \mathcal{T}) = \sum_{t\in\mathcal{T}} Vol_0(\partial t). \quad (3)$$

HC_0 counts the weighted pieces of cut hyperedges instead of the weighted sum of cut edges in the induced graph. It can be shown that $HC_0 = \sum_{e\in\mathcal{E}} w(e)|D(e)|$. This means we make the same penalty regardless the distribution of vertices across the partition boundary. As shown in Figure 2 (a), we want to divide the hypergraph with 3 hyperedges into 2 parts. The NHC_2 cut by the partition in Figure 2 (b) is 0.46 and by the partition in Figure 2 (c) is 0.27. This pairwise counting approach would prefer an uneven partition of the hyperedge rather than an even partition. It also means that large clusters will dominate the partition while the small clusters have smaller influence on the cut. With the non-pairwise cut HC_0 we can avoid this problem and keep the cuts in both Figure 2 (b) and (c) the same value. The k-cut has the same cost as HC_0 if $k = 2$, but we will show in the experiments that k-cut is not appropriate for clustering tasks when $k > 2$.

For NHC_2 and HC_0 the following property would help to group vertices in the preprocessing stage.

Proposition 1. *For any* $F \subseteq \mathcal{E}$, $K = (\bigcap_{e\in F} e) \setminus (\bigcup_{e\in\mathcal{E}\setminus F} e)$, *there exists a minimum* HC_0 *partition* $\hat{\mathcal{T}}$ *such that* $\exists t \in \hat{\mathcal{T}}$ *and* $K \subseteq t$. *(Proof omitted.)*

The vertices in K can be seen as "flat areas" in the hypergraph. The partition can only be placed on the boundary of "flat areas", so we can assign the vertices in K to the same cluster in the preprocessing stage. But the number of vertices in K depends on the choice of F.

Hypergraph modularity

If no prior knowledge or constraint about the number of clusters is given, minimizing the hypergraph cut would produce a trivial solution with a single cluster. To tackle this problem, a more sophisticated clustering fitness measure called modularity is proposed to consider both non-cut and cut edges in a single real-value number. Modularity is defined as the actual edge weights in the cluster minus the

expected edge weights in the cluster as if the edges are randomly placed between vertices with an expected probability (Newman and Girvan 2004; Brandes et al. 2007). Inspired by the simple graph modularity, for a weighted hypergraph $\mathcal{H} = \{\mathcal{V}, \mathcal{E}, W\}$ and partition $\mathcal{T}$ we extend it to *hypergraph modularity* based on the distribution of hyperedge weights

$$Q(\mathcal{H}, \mathcal{T}) = \sum_{t \in \mathcal{T}} \left(\frac{w(A(t))}{w(\mathcal{E})} - \left(\frac{\sum_{e \in C(t)} \frac{w(e)}{|D(e)|}}{w(\mathcal{E})} \right)^2 \right), \quad (4)$$

where $w(A(t)) = \sum_{e \in A(t)} w(e)$ and $w(\mathcal{E}) = \sum_{e \in \mathcal{E}} w(e)$. If all the hyperedges in $\mathcal{E}$ contain exact two vertices, we get the modularity definition for a simple graph.

One can show that hypergraph modularity has similar bounds as modularity for simple graph, which is from -1 to 1. It suggests a good cluster structure if $Q(\mathcal{H}, \mathcal{T})$ is close to 1 and totally non-cluster structure if it is close to -1.

Algorithm

The combinatorial problems of finding the partition that minimizes NHC_2, HC_0, and maximizes (hyper)graph modularity are proven to be NP-hard (Shi and Malik 2002; Brandes et al. 2006). In this work we use a simple agglomerative hierarchical clustering algorithm that minimizes the hypergraph cuts P mentioned in previous section (P could be NHC_2 or HC_0), and determines the optimal number of clusters with hypergraph modularity. The algorithm works as following,

Algorithm 1 : $\mathcal{T}$ = **AgglomerativeClustering**($\mathcal{H}$)

1: Build an initial partition $\mathcal{T}$ in which each cluster contains exact one vertex. Group the must-link vertices according to Proposition 1.
2: Choose $t_1, t_2 \in \mathcal{T}$ and merge them to get a new partition $\mathcal{T}'$ such that the improvement of clustering fitness measure $\Delta P(\mathcal{H}, \mathcal{T}) = P(\mathcal{H}, \mathcal{T}) - P(\mathcal{H}, \mathcal{T}')$ is maximized.
3: Set the output to $\mathcal{T}'$ if $Q(\mathcal{H}, \mathcal{T}')$ is larger than that of any previous iterations. Let $\mathcal{T} = \mathcal{T}'$ and repeat step 2 until $|\mathcal{T}| = 1$.

Usually the above algorithm gives some partition with $|\mathcal{T}| > 1$ because the modularity reaches a local maximum. In step 2 not all the entries of $\Delta P(\mathcal{H}, \mathcal{T})$ need to be recomputed after merging clusters t_1 and t_2 because the clusters other than t_1, t_2 remain the same.

The complexity of the algorithm is following: in step 1 each vertex is chosen in turns as the seed of finding F in Lemma 1. One can first sort the vertices by the adjacency matrix to make the process faster, so the complexity of step 1 is $O(|\mathcal{V}| \log |\mathcal{V}|)$. In step 2 one has to update the improvement of P for the newly combined cluster, which takes time $|\mathcal{E}|$. Then each newly computed improvement is inserted into a sorted list for choosing the next best combination, which takes time $\log(|\mathcal{V}|)$. Finally step 2 could be repeated at most $|\mathcal{V}| - 1$ times, so the worst case complexity of the algorithm is $O(|\mathcal{E}||\mathcal{V}| \log |\mathcal{V}|)$, and the space complexity is $O(|\mathcal{V}|)$.

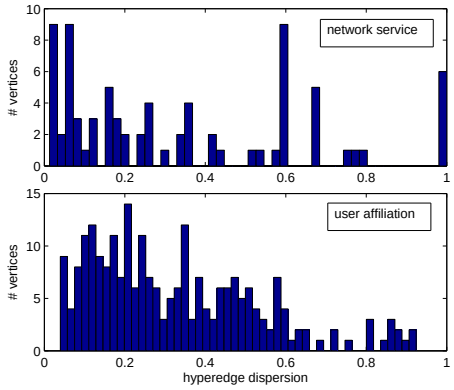


Figure 3: The hyperedge affinity distribution of two hypergraphs.

Hyperedge Affinity

The purpose of hypergraph modularity is to estimate the degree of existence of cluster structure in a hypergraph, but it requires a partition to compute the result. Can we make similar estimation without knowing any partition? The idea is to examine the degree of overlapping between the hyperedges that are incident with a chosen vertex. If these hyperedges cover the same set of vertices, it suggests a strong cluster structure. The hyperedge affinity of a hypergraph $\mathcal{H}$ is a mapping $d : \mathcal{H} \to \mathbb{R}^{|\mathcal{V}|}$ where each dimension d_i corresponds to a vertex v

$$d_i = \frac{2 \sum_{e_1, e_2 \in C(v)} |e_1 \cap e_2|}{|\cup C(v)||C(v)|(|C(v)| - 1)|}, \quad (5)$$

where $C(v)$ is the set of hyperedges that are incident with v. The denominator is just a normalizer that takes the largest possible value of the numerator. It is easy to verify that $0 < d_i \leq 1$. If most vertices have large d_i, it is likely to make a good partition for the hypergraph.

The histograms of $d(\mathcal{H})$ for network service and user affiliation are shown in Figure 3. For the hypergraph of network service, although there exists vertices of low hyperedge affinity, some vertices have very high d_i. Those vertices of high d_i could form several "cores" of the clusters, which allows good partitions. For the hypergraph of user affiliation, more vertices stay in the small hyperedge affinity range. It indicates that the behavior of users is more dispersive than the behavior of softwares. In the histogram of network service there are several peaks generated by some fixed routines in the software, while the histogram of user affiliation is smoother.

We have studied the hyperedge affinity with other datasets. Usually vertices of $d_i > 0.9$ indicate very strong co-occurrence relations and these relations are normally generated by non-human mechanism (e.g. software). On the other

hand, a concentration of vertices in the interval $d_i < 0.2$ suggests that the hypergraph does not carry cluster structure. The experimental results in the next section show that certain algorithms have better performances on hypergraph generated by non-human mechanism, and vice versa. The hyperedge affinity can be used to choose a suitable clustering algorithm.

Experimental Results

Hypergraph clustering approaches

Besides Algorithm 1 based on NHC_2 and HC_0, we also use several other popular clustering algorithms. First k-means is taken as the baseline in the experiments which directly obtains a feature vector of length $|\mathcal{E}|$ for each vertex where the entry is 1 if the vertex is included in the hyperedge, otherwise 0. If all the feature vectors are written in a row, it forms a matrix H of size $|\mathcal{E}| \times |\mathcal{V}|$ which is called the incident matrix. The single-linkage algorithm is reported to be the best clustering approach for network traffic or other signatures generated by softwares (Bayer et al. 2009; Perdisci, Lee, and Feamster 2010). We use the cosine distance for computing the pairwise distance in the single-linkage algorithm. The third algorithm called hMETIS is developed by Electronic Design Automation researchers (http://glaros.dtc.umn.edu/gkhome/views/metis). In hMETIS a multi-level clustering algorithm is implemented to minimize the k-cut.

The same feature vectors are also processed by the algorithm that adopts the spectral clustering (SC) approach described in (Zhou, Huang, and Scholkopf 2007). This approach is operated on the induced simple graph of the original hypergraph. The hypergraph Laplacian is computed from the normalized Laplacian of the induced graph. All the vertices are then mapped into points in a k dimension Euclidian space by the eigenvectors of the hypergraph Laplacian. For $k > 2$ a distance-based clustering algorithm has to be applied to read the cluster information from the mapped points.

We also take the non-negative matrix factorization (NMF) approach as an extension of k-means (Ding, He, and Simon 2005). The idea is to decompose the incident matrix H into two non-negative matrices W and G of low rank such that the difference $\parallel H - W^T G \parallel$ is minimized. If W and G have k rows, the rows of W are the k cluster centroids and columns of G are the cluster indicators for the vertices. We simply choose the index of maximum entry of each column in G as the cluster number of the corresponding vertex.

Finally the theory of centrality and PageRank (Bonacich and Lloyd 2001) is extended to fill in the zero entries of the feature vectors such that the distance between vertices in the same cluster becomes smaller. For each vertex v, we assign a non-negative importance score $s_v(e)$ for every hyperedge e. The score of hyperedges that are directly incident with v is set to 1 as in the feature vector. $s_v(e)$ of other hyperedges is determined by $s_v(e) = \sum_{e_1 \in N(e)} w(e_1) s_v(e_1)$, where $N(e)$ denotes the set of hyperedges that share at least one vertex with e. This computation is similar to a random walk on hyperedges with teleportation, and helps to identify other important hyperedges which is not directly incident with v in the original hypergraph due to noises. The "improved" features filled by the importance scores are then processed by another distance-based clustering approach. We call this approach hyperedge centrality (HCe).

In the experiments, if the algorithm requires the number of desired clusters as input, we set it to the number of clusters in ground-truth.

Network service identification

Two classes of datasets are used for network service identification, i.e., synthetic data and real data collected by Nexthink. The process of generating a synthetic dataset simulates the software and service behaviors in a real network. Each port is first associated with a service. The ratio of maximum service size over minimum service size is denoted by α, which indicates the degree of unbalance among services (clusters). Samples of application-destination pairs using the same service are uniformly drawn from all possible pairs, and a service is assigned to each pair. Then some connections are generated by associating ports in the corresponding services to application-destination pairs. To test the performance of the algorithm under noise, the ports of a fraction β of connections are chosen from a random service, which brings incorrect hyperedges into the dataset. In each synthetic dataset, there are 8000 connections, 80 applications, 40 destinations, and 180 ports which form 40 services.

The dataset collected by Nexthink contains records from July 2008 to September 2008 in a real enterprise network. There are 13774 connections, 320 applications, 840 destinations, and 474 ports. All connections are recorded in Windows systems. The connections with destinations outside the enterprise network are omitted, because we are only concerned about the services inside the local network. For the real data, we find many ports are not used as the registered functionality in IANA, so it is hard to label all the ports.

The algorithms are first tested on the synthetic data. We use the micro and macro average F-score to evaluate the results. Another two internal validation indices are also adopted to assess the goodness of the clustering results. The first index *Dunn index* is designed to identify dense and well-separated clusters and defined as the ratio between the minimal intra-cluster distance to maximal inter-cluster distance (Brun et al. 2007). A partition is better if it has larger Dunn index. The second index *Davies-Bouldin index* is measured by the average distance of points in a cluster to the center of the cluster and the distance between cluster centers (Davies and Bouldin 1979). A smaller value of Davies-Bouldin index indicates a better cluster structure.

Figure 4 shows the average performance on datasets of different degrees of noise and different degrees of unbalanced cluster sizes. We could confirm the fact that the single-linkage algorithm gives the best result in all cases. Since single-linkage finds the nearest neighbor clusters in each step and combines them into a new cluster, the result only depends on the local distances between a vertex and its neighbors. A vertex of high hyperedge affinity value indi-

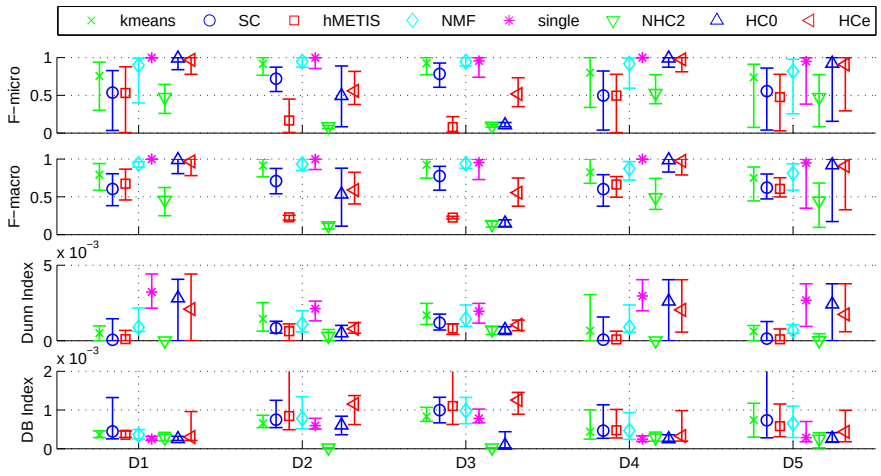


Figure 4: Performance of different algorithms on synthetic datasets. The parameters of the datasets are $D1$: $\alpha = 2$, $\beta = 0$, $D2$: $\alpha = 2$, $\beta = 0.05$, $D3$: $\alpha = 2$, $\beta = 0.1$, $D4$: $\alpha = 5$, $\beta = 0$, $D5$: $\alpha = 10$, $\beta = 0$. For each dataset the performance is averaged on 50 trails. The maximal and minimal values are also shown in the figure.

cates that its neighbors are very close. Thus single-linkage algorithm could easily identify the clusters from the local connectivity. If we known that the dataset is generated by non-human mechanism (e.g. from hyperedge affinity analysis), single-linkage would be the first choice.

One can observe that the HC_0-based algorithm produces good average results (better than that of NHC_2-based algorithm) when the data is noiseless, but it becomes not so good when the data is noisy. Essentially the HC_0-based algorithm is similar to the centroid linkage algorithm, so the result varies due to the change of centroids in the presence of noises. The performance of hyperedge centrality approach also decreases a lot with noisy data, because the connectivity between hyperedges used for computing importance scores is not reliable anymore.

k-means, spectral clustering, and non-negative matrix factorization produce relatively good results in all cases. The latter two methods can be seen as modifications of k-means. What is interesting is that the performance of these three methods becomes better when the data is noisy. This follows the fact that when the data manifold is fractured, some points might be near to the centroid of their true cluster.

The results of hMETIS are not as good as other cut-based algorithms even when the data is noiseless mainly due to the k-cut criterion. One can imagine that with the k-cut criterion big hyperedges are always first removed such that more disconnected clusters can be produced. But with HC_0 or NHC_2, removing big hyperedge would introduce more penalties because big hyperedge usually intersects with more clusters.

In Table 1, the individual F-scores of the smallest 2 clusters and the biggest 2 clusters are computed for all the algorithms. If we compare the unbalanced case ($D5$, $\alpha = 8$) with the more balanced case ($D1$, $\alpha = 2$), one can observe that single-linkage and HC_0 always produce the same good results on small and big clusters, while the other algorithms usually show better F-scores on big clusters and worse F-scores on small clusters. In the network service identification problem, the services of special interest are usually small clusters (rare behaviors, malware, etc.), so the single-linkage and HC_0 algorithm are more capable of isolating small interested services from a huge amount of traffic.

kmeans	TCP2967, UDP2967, UDP38293 rtvscan.exe, savroam.exe 10.0.0.8, 10.100.0.5, 10.104.0.8, ...
SP	TCP2967, UDP1281, UDP2967, ... rtvscan.exe, savroam.exe 10.0.0.8, 10.100.0.5, 10.104.0.8, ...
hMETIS	TCP2967, UDP1153, UDP1179, ... outlook.exe, rtvscan.exe, savroam.exe 10.0.0.8, 10.10.3.21, 10.100.0.5, ...
NMF	TCP2967, TCP8080, UDP8080 rtvscan.exe, yahoomessenger.exe 10.0.0.8, 10.100.0.5, 10.104.0.8, ...
single	TCP2967, UDP2967, UDP38293 rtvscan.exe, savroam.exe 10.0.0.8, 10.100.0.5, 10.104.0.8, ...
NHC2	TCP2967, UDP1281, UDP2967, UDP38293 rtvscan.exe, savroam.exe 10.0.0.8, 10.100.0.5, 10.104.0.8, ...
HC0	TCP2967, UDP1281, UDP2967, UDP38293 rtvscan.exe, savroam.exe 10.0.0.8, 10.100.0.5, 10.104.0.8, ...
HCe	TCP2967 rtvscan.exe, 10.0.0.8, 10.100.0.5, 10.104.0.8, ...

Table 2: Example of network service detected from Nexthink dataset by different algorithms.

The algorithms usually output about 70 to 85 services (clusters) on Nexthink dataset when the maximum modularity is reached. We identified some network services in the dataset, where one example is shown in in Table 2. Usually the clusters generated by single-linkage, NHC_2 and HC_0 coincide with each other, which indicate that the local connectivity in Nexthink data is very strong. The normalized

	pre.	rec.	F-score	pre.	rec.	F-score	pre.	rec.	F-score	pre.	rec.	F-score
cluster $D1$		S1			S2			L1			L2	
k-means	0.877	0.993	0.906	0.901	0.987	0.918	0.936	0.970	0.941	0.877	0.960	0.897
SP	0.754	0.947	0.769	0.723	0.953	0.752	0.762	0.977	0.817	0.845	0.987	0.883
hMETIS	0.695	1.000	0.773	0.656	1.000	0.749	0.929	0.953	0.929	0.926	0.942	0.919
NMF	0.861	1.000	0.902	0.909	1.000	0.936	0.976	0.963	0.963	0.992	0.955	0.967
single	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
NHC_2	0.490	0.947	0.502	0.490	0.920	0.496	0.546	0.917	0.550	0.681	0.876	0.635
HC_0	0.988	1.000	0.991	0.988	1.000	0.991	1.000	1.000	1.000	1.000	1.000	1.000
HCe	1.000	0.993	0.996	0.990	0.987	0.985	0.989	0.990	0.988	1.000	0.990	0.994
cluster $D5$		S1			S2			L1			L2	
k-means	0.915	1.000	0.937	0.901	0.990	0.925	0.984	0.910	0.933	0.942	0.914	0.910
SP	0.691	1.000	0.745	0.664	0.940	0.681	0.824	0.960	0.859	0.851	0.989	0.897
hMETIS	0.517	1.000	0.616	0.648	1.000	0.716	0.967	0.907	0.926	0.972	0.931	0.943
NMF	0.619	1.000	0.693	0.718	1.000	0.782	0.989	0.965	0.971	0.988	0.959	0.968
single	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
NHC_2	0.816	1.000	0.829	0.575	0.940	0.573	0.706	0.911	0.701	0.701	0.915	0.690
HC_0	0.983	1.000	0.986	0.968	1.000	0.973	1.000	0.997	0.998	0.995	0.998	0.996
HCe	0.894	1.000	0.917	0.910	1.000	0.931	0.994	1.000	0.996	1.000	0.998	0.999

Table 1: The precision, recall, and F-score of the 2 smallest clusters (S1, S2) and the 2 biggest clusters (L1, L2) with datasets $D1$ ($\alpha = 2$) and $D5$ ($\alpha = 8$). The performance is averaged on 50 trails.

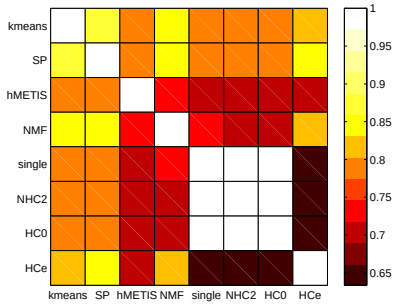


Figure 5: The normalized mutual information between the partitions by different algorithms on Nexthink dataset.

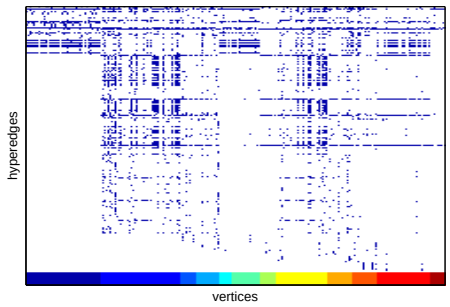


Figure 6: A spy plot of the hypergraph for user affiliation identification. Each vertex represents a user, and each hyperedge denotes a set evidence. There is a dot if the user is included in the set evidence. The vertices are sorted by labels and different colors on the bottom indicate the labels.

mutual information (NMI) between the partitions by different algorithms is shown in Figure 5. We follow the definition of NMI in (Strehl and Ghosh 2003).

User affiliation identification

Another dataset used for user affiliation identification is collected by Nexthink from computer rooms where the users from different departments could access local and Internet contents. With the data collected in a period of two weeks, we get 237 users and 1604 hyperedges after removing those hyperedges which consist of only one user. The affiliation of users is directly taken from the user profile and used as labels. Although the labels may not reflect the underlying group structure of users, we use them for evaluating clustering algorithms. In the label list, there are 12 affiliations. The smallest and largest affiliation contains 7 users and 45 users respectively. A cleaned spy plot of the hypergraph is shown in Figure 6.

The F-scores and modularity are shown in Table 3. What is different from the data generated by a non-human mechanism, we can observe an almost reversed ranking of algorithms. The algorithms that are similar to k-means show better performances than the hierarchical clustering methods. This is mainly due to the fact that the assumed manifold structure behind the nearest neighbor rule becomes unreliable when the user behavior is more dispersive. The F-scores indicate that the partitions of all algorithms are very different from labels. But a partial reason of the low F-score is that the labels from user profiles are not reflecting the real underlying cluster structure. A maximum modularity partition of HCe algorithm consists of only 3 clusters instead of 12 clusters in the labels (Figure 7). The modularity of the labels is actually the lowest among all algorithms.

Conclusion

In this work, the network service identification problem and user affiliation identification problem are studied. The results help to group the open ports and users in the network such that a human system administrator could efficiently an-

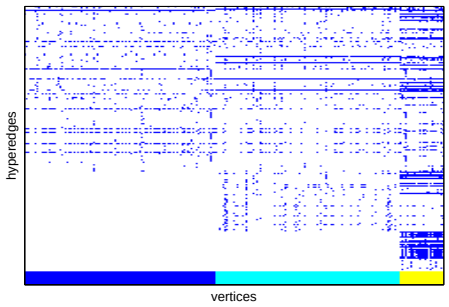


Figure 7: A spy plot of user affiliation with maximum modularity partition from HCe algorithm.

	F-micro	modularity
k-means	0.262 ± 0.085	0.037 ± 0.005
SP	0.225 ± 0.092	0.086 ± 0.012
NMF	0.267 ± 0.064	0.033 ± 0.039
single	0.198 ± 0.000	0.011 ± 0.000
NHC_2	0.203 ± 0.000	0.012 ± 0.000
HC_0	0.205 ± 0.002	0.012 ± 0.007
HCe	0.268 ± 0.000	0.058 ± 0.000
labels	–	-0.121

Table 3: The performance of different algorithms on the user affiliation identification. hMETIS is not listed in the table because it usually fails to give a valid partition.

alyze the records. By adopting the hypergraph model, we propose an agglomerative hierarchical clustering algorithm based on a non-pairwise hypergraph cut HC_0. Experimental results on both synthetic and real data show that hierarchical clustering algorithms based on the nearest neighbor rule performs better on the data generated by a non-human mechanism. With the user generated records, k-means-like algorithms usually produces better results because the assumed manifold structure behind the nearest neighbor rule is not reliable in such cases. The experimental result also confirms the indication suggested by the hyperedge affinity analysis, which allows us to identify the nature of dataset before applying any clustering algorithm.

References

[Bayer et al. 2009] Bayer, U.; Comparetti, P.; Hlauschek, C.; Kruegel, C.; and Kirda, E. 2009. Scalable, behavior-based malware clustering. In *Network and Distributed System Security Symposium (NDSS)*.

[Bonacich and Lloyd 2001] Bonacich, P., and Lloyd, P. 2001. Eigenvector-like measures of centrality for asymmetric relations. *Social Networks* 23(3):191–201.

[Brandes et al. 2006] Brandes, U.; Delling, D.; Gaertler, M.; Goerke, R.; Hoefer, M.; Nikoloski, Z.; and Wagner, D. 2006. Maximizing modularity is hard. *Arxiv preprint*.

[Brandes et al. 2007] Brandes, U.; Delling, D.; Gaertler, M.; et al. 2007. On modularity clustering. *IEEE Transactions on Knowledge and Data Engineering* 20(2):172–188.

[Brun et al. 2007] Brun, M.; Sima, C.; Hua, J.; Lowey, J.; Carroll, B.; Suh, E.; and Dougherty, E. 2007. Model-based evaluation of clustering validation measures. *Pattern Recognition* 40(3):807–824.

[Chandola, Banerjee, and Kumar 2009] Chandola, V.; Banerjee, A.; and Kumar, V. 2009. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)* 41(3):15.

[Davies and Bouldin 1979] Davies, D., and Bouldin, D. 1979. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2):224–227.

[Ding, He, and Simon 2005] Ding, C.; He, X.; and Simon, H. 2005. On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proc. SIAM Data Mining Conf*, 606–610.

[Karagiannis, Papagiannaki, and Faloutsos 2005] Karagiannis, T.; Papagiannaki, K.; and Faloutsos, M. 2005. BLINC: multilevel traffic classification in the dark. *ACM SIGCOMM Computer Communication Review* 35(4):240.

[Karypis and Kumar 1999] Karypis, G., and Kumar, V. 1999. Multilevel k-way hypergraph partitioning. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, 343–348.

[Kulis et al. 2009] Kulis, B.; Basu, S.; Dhillon, I.; and Mooney, R. 2009. Semi-supervised graph clustering: a kernel approach. *Machine Learning* 74(1):1–22.

[Newman and Girvan 2004] Newman, M., and Girvan, M. 2004. Finding and evaluating community structure in networks. *Physical review E* 69(2).

[Perdisci, Lee, and Feamster 2010] Perdisci, R.; Lee, W.; and Feamster, N. 2010. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *USENIX conference on NSDI*.

[Pu et al. 2010] Pu, L.; Faltings, B.; Yang, Q.; and Hu, D. 2010. Relational network-service clustering analysis with set evidences. In *Proceedings of the 3rd ACM workshop on Artificial intelligence and security*, 35–44.

[Schaeffer 2007] Schaeffer, S. 2007. Graph clustering. *Computer Science Review* 1(1):27–64.

[Shi and Malik 2002] Shi, J., and Malik, J. 2002. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22(8):888–905.

[Strehl and Ghosh 2003] Strehl, A., and Ghosh, J. 2003. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *The Journal of Machine Learning Research* 3:583–617.

[Xiao 2010] Xiao, M. 2010. Finding minimum 3-way cuts in hypergraphs. *Information Processing Letters* 110(14-15):554–558.

[Zhou, Huang, and Scholkopf 2007] Zhou, D.; Huang, J.; and Scholkopf, B. 2007. Learning with hypergraphs: Clustering, classification, and embedding. *Advances in Neural Information Processing Systems*.

IPv6 Traffic Flow Security Threats and Trend Analysis: Towards Combined Intelligent Detection Approach

Muhammad Fermi Pasha[1], Mustafa Abdat[2], Mandava Rajeswari[1]

[1]School of Computer Sciences, Universiti Sains Malaysia, 11800 Minden, Pulau Pinang, Malaysia
[2]National Advanced IPv6 Center, Universiti Sains Malaysia, 11800 Minden, Pulau Pinang, Malaysia
{fermi, mandava}@cs.usm.my, mustafa@nav6.usm.my

Abstract

The advent of IPv6 as a replacement of IPv4 has spurred new challenges in intelligent network security research. With IPv4 address allocation predicted to run out not long from now, it is eminent to start migrating to IPv6 network and examine how the threats might evolve from one form to another. This paper presents a proposal towards combined intelligent approach on detecting IPv6 network traffic security threats and trend analysis. The proposed approach combines a fuzzy hybrid network discovery, online two-tier traffic aggregation, fuzzy alarms generation scheme, and network trend analysis as one detection approach. Some preliminary results are also reported and discussed in addition to the work future direction.

Introduction

Since the ARPANET research network first gained popularity in the early eighties and then started to evolve into the commercial Internet in early nineties, we have seen an exponential growth of computer network usage. This exponential growth combined with the limited 32 bits design of IPv4 addresses has been the primary reason for the birth of the new IP version 6 (IPv6). Due to this reason, clearly the first and the foremost spotted feature of IPv6 is its large address space (128 bits).

Even though there are quite a number of improvements are made into IPv6 designs, in many ways, IPv6 is still works pretty much the same as IPv4. Therefore when it comes to security issues, a large portion of IPv4 security threats can be also migrated and occurred in IPv6. With the allocation of IPv4 address predicted to be assigned in the near future, the need to intensify IPv6 security threat detection research has arise.

In this research work, we aim to start research on IPv6 security threats detection focusing on intelligent network traffic flow analysis. Network monitoring and traffic measurement has been the primary focus of network administrator to detect network threats (Mukherjee et al. 1994). Numerous works have been devoted to propose a new intelligent scheme and technique in detecting threats inside computer network (Pasha et al. 2006) (Xia, Lu, and Li 2010) (Li, Li, and Wang 2006) (Wu and Banzhaf 2010). Nevertheless, work that is trying to focus on IPv6 traffic flow based security threats detection is still can be considered lacking.

We propose a combined intelligent detection approach to detect the most common IPv6-based attacks. We combine a fuzzy hybrid network discovery, online two-tier traffic aggregation, fuzzy alarms generation scheme, and network trend analysis to form one intelligent detection approach. Combining these four approaches provide more information details in which when combined, it will increase the accuracy of threat detection.

The reminder of this paper is organized as follows: The IPv6 Traffic Flow Security Threats Section reviews the current and most common attacks that might exist in IPv6, then followed by a Section introducing our proposed combined approach. After that the Preliminary Results and Discussion Section presents some preliminary results obtained from our experiments. Lastly, a Conclusion and Future Works Section ends this paper.

IPv6 Traffic Flow Security Threats

There are hundreds, if not thousands, network security threats exist since the Internet gains popularity. At the very high level, they can be divided into to category: a host based attack and a traffic flow based attack. A host-based attack involves utilizing the network to hack into

individual host/computers to gain unauthorized access. Whereas a traffic flow based attack involves leveraging the traffic flow to disrupt specific services in the network.

As the Internet today started off using IPv4, obviously all attacks have some form of relation to IPv4 characteristics (Postel, 1981). However, as IPv4 address is getting depleted soon, hackers and the Internet villains have started their attack migration by taking into accounts IPv6 specifications (Deering and Hinden 1998). Even though at the core level, the packet transportation protocol across the network and the upper-layer protocol are similar between IPv4 and IPv6, IPv6 has often being considered more secure than IPv4 due to the inclusion of IPsec (Kent and Atkinson 1998) in IPv6 specification. This may be true in ideal environments, however in reality, the same problems that plague IPv4 IPsec will still exist.

Therefore, IPv6 security threat detection research area has slowly launched and becoming more challenging. Here we reviewed some of known common threats that have been evolved as an IPv6 security threats and how it differs from IPv4. The focus of IPv6 security threat analyzed in this paper is only on the traffic flow based attack. Specifically, four types of IPv6 security threats are reviewed: reconnaissance attack, man-in-the-middle attack, distributed denial of service attack, as well as network worms and viruses.

Reconnaissance Attack

Reconnaissance attack is an act of probing active hosts over their open ports (services). Today, it is among the most commonly occurred attacks in almost any network. This method allows intruders to listen to specific services, which could be associated to widely known vulnerabilities (Ford 2005). The address space of IPv4 is small thus probing a whole class C network will take only less than 5 minutes. However, unlike in IPv4, the IPv6 networks have very large address space (default subnets of IPv6 contains up to 2^{64} addresses). Therefore it is almost seems impossible for an intruder to perform simple reconnaissance attack in IPv6 network. Due to this, port-scanning method is likely to change in IPv6 network. Instead of guessing what IP addresses are available through ICMP queries, intruders are prone to use new methods to obtain IPv6 addresses such as from DNS servers. A compromise DNS server can yield a large list of legitimate IPv6 addresses to intruders, which gives them enough information to launch a reconnaissance attack.

Man-in-the-middle Attack

The main purpose of these attacks is to gain information from victim communication messages. Man-in-the-middle attacks might cause communication messages sent by victims to be sniffed, altered or even stopped by intruders. There are several types of DHCP attacks, the most common ones would be DHCP starvation and rogue DHCP server attacks. The first type of attack is normally triggered when intruders send crafted DHCP queries with nonexistent MAC addresses. This causes a DHCP server to dry up its IP address pool size very quickly and becomes unable to release new IP addresses for legitimate hosts, which are leasing for an IP address. The second type of attack is normally launched to disable public servers thus it becomes unresponsive to accept valid requests from network users. Even though IPv6 comes with an enhanced ICMPv6 (Deering and Hinden 1998) with its stateless autoconfiguration, a similar DHCPv6 server may still be used during the foreseeable long transition period. Therefore, this kind of attack will still exist in IPv6 network.

Distributed-denial-of-service Attack

In distributed-denial-of-service (DDoS) attack, an intruder generates large amount of computer network traffic on a victim network. This attack floods a target system via rogue broadcast ping messages. It sends crafted echo-request (ICMP type 8) packets with the victim's IP source address. All hosts on the network segment respond to the crafted packets and flood the victim with echo-reply messages. Even though this kind of attack has become less common, it can still be used to launch an effective service disruption attack. The effectiveness of this method might vary among different computer networks since the configuration applied by network administrators play dominant role for this method to work. Since DDoS attack general mechanism involves flooding the network with traffic targeting specific service, it will still possible to occurred in IPv6 network.

Network worm Attack

Network viruses and worms (or malware) remain one of the most significant problems in IP networking today. Computers with malware underlying beneath them are often linked together to form botnets, and large number of attacks are launched using these malicious, attacker-controlled networks. Today, millions of compromised web sites launch drive-by download exploits against vulnerable hosts (Provos et al. 2008). As part of the exploit, the victim machine is typically used to download and execute malware programs. These programs are often bots that join forces and turn into a botnet. Botnets are often used by intruders to launch DOS attacks, send spam mails, or phishing pages. The IPv6 protocol implementation does not greatly affect the impact of malware attacks. Malware propagation in IPv6 is believed has been reduced due to the difficulty of enumerating the hosts on a subnet. Although, the current mitigation techniques for identifying malware in IPv4 is still valid in IPv6, network administrators will need to be wary of possible new malware proliferation methods.

Combined Intelligent Detection Approach

We propose a combined intelligent detection approach that integrates few techniques and analysis to detect IPv6 security threats. Figure 1 shows the general architecture of our proposed combined approach. Currently, we propose to combine 4 different intelligent techniques focusing on different aspect of traffic analysis: Fuzzy Hybrid Network Device Discovery, Online Two-tier Traffic Aggregation using ECM, Fuzzy Alarms Generation Scheme, and Network Trend Analysis using Triple Exponential Smoothing. As an integrated approach, the output from each analysis is used to improve the accuracy and quality of IPv6 security threats detection.

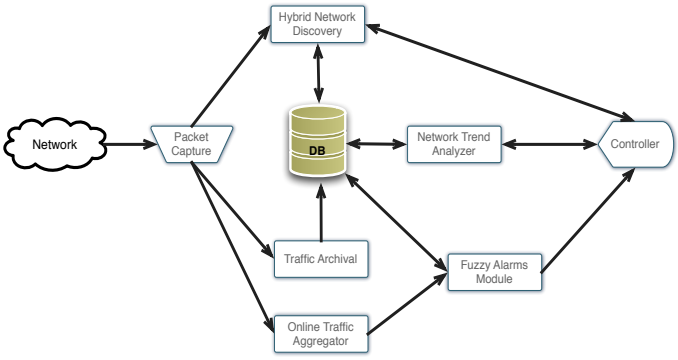


Figure 1: Combined Approach General Architecture

Fuzzy Hybrid Network Device Discovery

We have started the work on proposing a fuzzy-based hybrid technique (Abdat et al. 2010) for network device discovery that covers the detection of personal computers (PC's), switches, routers, and printers in the network. The detection does not entirely rely on SNMP technique, instead it also uses passive packet sniffing technique. Using a fuzzy approach to hybridize the combination of the two techniques, the system has the option to efficiently alternate which technique to be used and improve the performance while minimizing additional generated network traffic.

There are three parts of the proposed work. The first part is the passive discovery analysis module. The second part is the normal active approach using SNMP-based technique. And the last part is the fuzzy hybrid discovery module as the main component to control and combine the first two modules efficiently.

As the main component, the fuzzy hybrid discovery module controls the overall operation of the proposed work. A simple fuzzy membership function creating a fuzzy value is used to improve performance of hybridizing the passive and active device discovery approaches. The following is the summary of the fuzzy hybrid discovery module algorithm.

1. Begin
2. While *prd* > 0
3. Start *prd* count down counter
4. Initialize the membership function μ_a:
5. If $\mu_a > 0.3$
    a. While network packet available
    b. Take *p* as input,
    c. Decode and analyze *p*
        i. Perform passive device discovery analysis.
        ii. Add the discovered IP / MAC address information from *p* to *dal*.
6. If $\mu_a < 0.3$ || If *prd* == 0
    a. Based on the network subnetting, generate possible address list denoted as *pal*
    b. Use SNMP to query addresses in *pal* but not in *dal*
    c. Add SNMP returned IP / MAC address to *dal*.
7. Adjust *prd* and μ_a based on *dal* and *d_val*
8. End while loop
9. Finish

As part of the combined detection approach, an accurate and online network discovery will help providing the required information on the affected device either due to being attacked, faulty, or the one attacking. The discovered list of network devices will be used later by the Fuzzy Alarms Generation Strategy to include information details of the pinpointed sources of problems in the alarms generated.

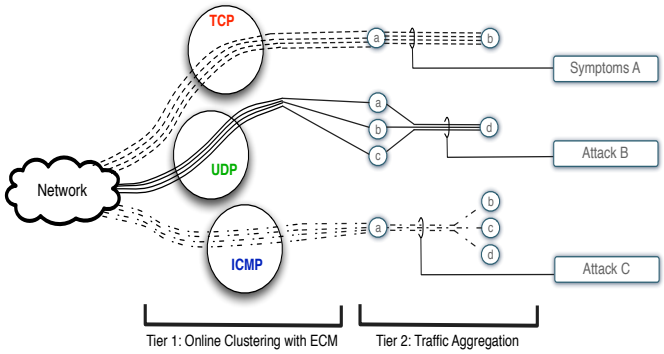


Figure 2: Online Two-tier Traffic Aggregation using ECM

Online Two-tier Traffic Aggregation using ECM

The idea of an online two-tier traffic aggregation is to combine the powerful online clustering techniques ECM with statistical traffic aggregation approach. This

combination will offer a fast traffic aggregation process, which will in turn improve the detection quality of an attack or network fault. Figure 2 depicts the proposed Online Two-tier Traffic Aggregation architecture.

Evolving Clustering Method (ECM) is a fast one-pass algorithm for dynamic clustering of an input stream of data, where there is no predefined number of clusters. It is a distance-based clustering method where the cluster centers are presented by evolved nodes in an online mode (Kasabov 2002). For any such cluster, the maximum distance *MaxD*, between an example point and the closes cluster center, cannot be larger than a distance threshold value, *Dthr*; that is a pre-set of parameter. This parameter would affect the number of evolved cluster. The threshold value *Dthr* can be made adjustable during the online clustering process, depending on some optimization and self-tunning criteria, such as current error or number of clusters.

The following steps summarizes the ECM algorithm:

Step 1: Create the new cluster by simply taking the position of the fist example from the input data stream as the first cluster center. The radius for this cluster is set to 0.

Step 2: Until all the examples from the data stream have been presented, the clustering process finishes. Otherwise, the current input example is taken to be check where it belongs.

Step 3: If the current input belongs to one of the already created clusters and the distance value of the current input is below the radius of that cluster or the current input not belong to any of the already created cluster, which means new cluster has to be created then go back to Step 2. Else:

Step 4: Find the already created cluster where the current input belongs, and update its radius by adding the distance value of the current input with the old radius value.

Step 5: If the new radius value is not greater than 2 x *Dthr* then that current input does not belong to any existing cluster. A new cluster is created in the same way as Step 1. The algorithm then goes back to Step 2.

Step 6: If the new radius value is not greater than 2 x *Dthr* then the cluster center is updated accordingly as the new radius for that cluster is set. The algorithm then goes back to Step 2.

After being captured, network traffic is clustered using ECM based on its traffic type. This will eliminate unnecessary traffic when apply the traffic aggregation techniques. A clustered traffic can also be useful to further scope down the focus of analysis based on specific type of traffic that caused the frequently occurred network attacks/problems.

The next step is to aggregate the clustered traffic further. The aggregation process is performed on each cluster created. For example, in Figure 2 the illustration shows that ECM is set to cluster three different type of traffic. In this case the aggregation process will be applied to each cluster, which in the end different kind of attacks or symptoms of IPv6 traffic flow threats can be detected.

At this stage, we propose to use a simple aggregation technique to perform the traffic aggregation. The general idea is proposed by (Kanamaru et al. 2000) and basically the aggregation process is simply to look for three combination of aggregated traffic. The combination model is as follows:

- **One-to-One** model: Packets delivered from a specific source address to a specific destination address. This type of packet flow is the most basic aggregation model.
- **Many-to-One** model: Packets delivered from any source address to a specific destination address. When more than one host send packets to one specific host, those packets will be aggregated as this type of aggregation model.
- **One-to-Many** model: Packets delivered from a specific source address to multiple destination addresses. When one specific host sends packets to multiple hosts, those packets will be aggregated as this type of aggregation model.

An analyzer using simple rule based detection engine is then being applied to the resulted aggregated traffic for each model. Sample of the rule employs by the analyzer is as follows:

```
IF Model=A && Type=B && Size=C THEN D
                ELSE Normal
```

where *A* is the aggregation model, *B* is the type of traffic, *C* is the size of the traffic flow, and *D* is type of attack or faults.

Fuzzy Alarms Generation Scheme

Another aspect that is almost as crucial as the detection of threat itself is alerting. Due to the dynamic nature of network traffic flow, there can be no fool prof system made to detect security threats with 100% accuracy. Therefore, it will be beneficial to network administrator when a threat detection system equipped with an alarms generation scheme to reduce the number of false alarms rate.

Here we propose a new scheme and procedure to generate alarms in conjunction with threats or fault detected after the traffic aggregation process. Instead of

directly raise an alarm when the online traffic aggregation analyze that an attack or fault had occurred, we further apply the proposed strategy and procedure to carefully raise the alarms. The proposed strategy and procedure is implemented using fuzzy theory and designed to reduce false alarms rate to as minimum as possible. Our general assumption in applying this strategy is that no one is using the network at late night, no heavy workload on holiday, and working day is Monday to Friday only.

There are three types of alarms generated by this strategy, Warning Alarms (WA), Attention Alarms (AA), and Critical Alarms (CA). Warning alarm is simply an alarm raised when any threats detected. WA are not categorized as an alarm that requires an attention, but instead it is more to a log activity. Whereas AA is the less important one and intents to get some attention from the network administrator. It is raised if a sequence of WA exists. And CA is the real red alert that requires an urgent immediate response and triggered using fuzzy set membership function of a sequence of both warning and attention alarms.

Our strategy is to divide the time of day into three periods of peak-time, off-peak-time, and midnight. Furthermore, the length of this period is different on each day. Working day (Monday to Friday) uses one period whereas off working day (Saturday, Sunday, and holiday) uses another period. As for deciding holiday, network administrator must set it a day before that the next day is a holiday so that the system will then uses holiday period to that day.

The standard, applies all day, mechanism procedure in raising a WA is expressed by ∀ Traffic ∃ x, where x is set of traffic which classified as threats after the online aggregation analysis process. The standard mechanism in generating an AA is WA has to be raised 4 times (default number) in a row. Whereas since CA is triggered using fuzzy membership function, the standard mechanism is using the default generalized bell membership function as follows:

$$f(x) = \frac{1}{1+\left|\frac{x-c}{a}\right|^{2b}} \quad (1)$$

where the default value of a, b, c is 2, 6, and 8 respectively.

Peak-time period

For working day, the peak-time period is from 8 AM to 8 PM. This peak-time is chosen based on extended working hours normally adopted in most universities. As of for off working day, the period is different between Saturday, Sunday, and holiday. Saturday peak-time period is 8 AM to 5 PM, Sunday peak-time is 9 AM to 3 PM, and holiday peak-time is the same as Saturday peak-time. However, network administrator can shift and adjust this period accordingly at later time when needed. The triggered WA in a row rate to raise AA is set to 6 in this period and the fuzzy membership *f(x)* threshold value to trigger CA is 0.3 and 0.9.

False alarms are likely to happen in this peak-time of working day period due to high traffic volume to be monitored. Common alarms in most anomaly detection systems are comparable with the CA in our types of alarms. As such, we count the hit of CA to measure false alarms. That means, WA and AA are not count as false alarms and thus the rates of false alarms can be reduced.

Off-peak-time period

These periods are the transitions from peak-time to midnight period and back to peak-time. Therefore, there are basically two period of this type. The off-peak-time period for working day is from 8 PM to 12 AM and from 6 AM to 8 AM, for Saturday is from 5 PM to 12 AM and from 6 AM to 8 AM, for Sunday is from 3 PM to 12 AM and from 6 AM to 9 AM, and for holiday is the same as Saturday. Basically since midnight period is the same for every day which is from 12 AM to 6 AM; if the peak-time is adjusted to 6 AM to 5 PM the second period of off-peak-time is at least one hour prior to peak-time and therefore midnight period has to be adjusted to become from 12 AM to 5 AM.

In this period usually activities inside the network are starting to slow down and thus we set the fuzzy membership *f(x)* threshold value to trigger CA to 0.2 and 0.75 in this period. By having this transitions periods, the alarms can be carefully triggered and avoid sudden gap in the threshold degree from peak-time to midnight period and thus the possibilities of CA being triggered upon normal instances can be reduced

Midnight period

The midnight period is quite fixed to 12 AM to 6 AM as it is the transition between the days (unless set differently by network administrator). The fuzzy membership *f(x)* threshold value to trigger CA is set to 0.1 and 0.5 in this midnight period. This is due to our assumption that there are no midnight activities inside the network. Nevertheless, network administrator is still able to adjust this degree manually in a case of midnight work shift.

Network Trend Analysis using Triple Exponential Smoothing

As part of the combined approach, here we propose to include a network trend analysis to complement the overall IPv6 security threat detection. Since our combined detection approach depends on analysis of the traffic flow, we believe including a trend analysis will help identifying changes in traffic pattern should an attacker try to launched an attack slowly.

However, at this stage, our proposed network trend analysis model is still in its earliest form. We propose to

use triple exponential smoothing method to predict the traffic pattern. The basic principle of this method is to separate three factors (level factor, tendency factor and period factor) from the time series, and then to use these three factors to forecast for next period. We adopt the eleven steps to develop the forecast model based on two periods of the historical data described in (Bartolomei, and Sweet 1989).

Step 1: Calculate the average value for each of the two periods using the following:

$$V_n = \frac{1}{l}\sum_{i=1}^{l} x_i \quad (2)$$

Step 2: Calculate the increment value *B* from the two periods calculated in Step 1.

$$B = \frac{1}{l}(V_2 - V_1) \quad (3)$$

Step 3: Initialize the smoothed value factor *S*.

$$S = V_2 + \frac{l-1}{2} \times B \quad (4)$$

Step 4: Compute the time period periodical factor for each of the two periods using the following:

$$C'_{tn} = \frac{x_t}{V_n - (\frac{l+1}{2} - m) \times B} \quad (5)$$

where $t=l$

Step 5: Calculate the average value of the two time period periodical factor in Step 4.

$$C''_t = \frac{1}{2}(C'_{t2} - C'_{t1}) \quad (6)$$

Step 6: Calculate the sum of l average periodical factor l':

$$l' = \sum_{t=l+1}^{2l} C''_t \quad (7)$$

Step 7: Normalize the periodical factor.

$$C_t = \frac{l}{l'} C''_t \quad (8)$$

Step 8: Use the three factors obtained (*S*, *B*, and *C*) to calculate the initial next forecast period F_{t+m}:

$$F_{t+m} = (S + mB) \times C_{t-l+m}) \quad (9)$$

Step 9: Use the correlation coefficient α,β,γ to correct the level factor, the tendency factor, and the period factor using the following:

$$S_t = \alpha \frac{x_t}{C_{t-l}} + (1-\alpha)(S+B)$$
$$B_t = \gamma(S_t - S) + (1-\gamma)B \qquad 0 < \alpha, \beta, \gamma < 1$$
$$C''_t = \beta \frac{x_t}{S_t} + (1-\beta)C_{t-l} \quad (10)$$

Step 10: Use the corrected values of the three factors to forecast the remaining period.

$$F_{t+m} = (S_t + mB_t) \times C_{t-l+m}) \quad (11)$$

where $m=1,2,...,(l-1)$

Step 11: Upon finished forecasting one period, go back to Step 7 to normalize the period factor.

Preliminary Results and Discussion

We setup a dual stack IPv4/IPv6 testbed at National Advanced IPv6 (NAv6) Center network to conduct few experiments to validate our proposed combined intelligent detection approach. The whole experimentation is planned to run for several months including within production network at a later stage. Here, we report some promising preliminary results we obtained from the first one month of our experiments.

Basically the experiments are carried out separately at this stage to validate each approach used, however, the output of each experiment is used by the next experiment in order to evaluate whether the combination might be useful. The first experiment we run is the automatic network device discovery experiment. Figure 3 shows the interface display visualization of some of the discovered devices.

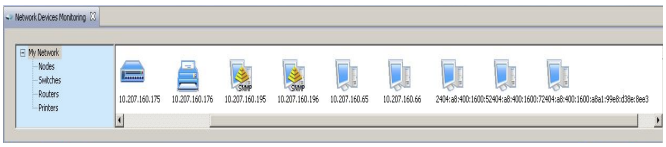


Figure 3: Automatic network device discovery output

During the experiment, our proposed Fuzzy Hybrid Network Device Discovery technique manages to discover and collect important device information ranging from PC with both IPv4 and IPv6, switches, as well network printers with minimum additional traffic overhead generated by SNMP. This information is then loaded into a dedicated database to be used by the other approaches in the next experiments.

From the experiments we also learnt that the device behavior in sending and receiving network packets might affect the performance of our proposed method. There are cases when our method is unable to detect network devices due to unavailability of the needed traffic. For example to discover one network printer, the method will look for IPP packet traffic but when that printer is not actually being used over the network, there will be no IPP packet in the network and hence the particular network printer will not be discovered immediately when our method is in passive mode. So to discover that printer, it has to wait when our method set to work in active mode and it has to be an SNMP enabled network printer.

Next we run the experiment to validate our proposed Online Two-tier Traffic Aggregation approach. In this experiment we run a prototype of ECM and its traffic aggregator with packet capture and filtering module. We also develop a packet generator and packet relay system to populate the testbed network with the needed IPv6 threats as well as some faults scenario. During the two weeks experiment, we focus only on HTTP, ARP, and ICMP type of traffic. Hence, we filter the captured packets to only forward these three types of traffic to be further clustered.

We also set a few basic rules to cover few threats based on its characteristics described earlier in this paper.

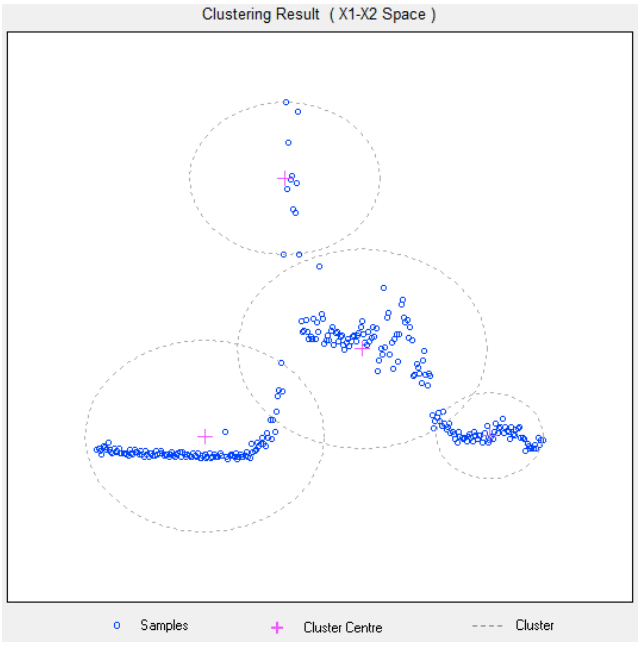


Figure 4: ECM Results Clustering ICMP Traffic

Figure 4 depicts the clustering results of ICMP traffic on one of the simulated attack-testing day. It is basically visualizing the ICMP traffic pattern from the start of day (12AM) until the end of the day. We can clearly see that the traffic during after midnight is constantly low as we did not generate any traffic and the traffic start to increase during morning period when start experimenting with few simulated attack and fault scenario. At the evening, the traffic is becoming low again. The clustering results of the other two traffic (HTTP and ARP) also following similar pattern.

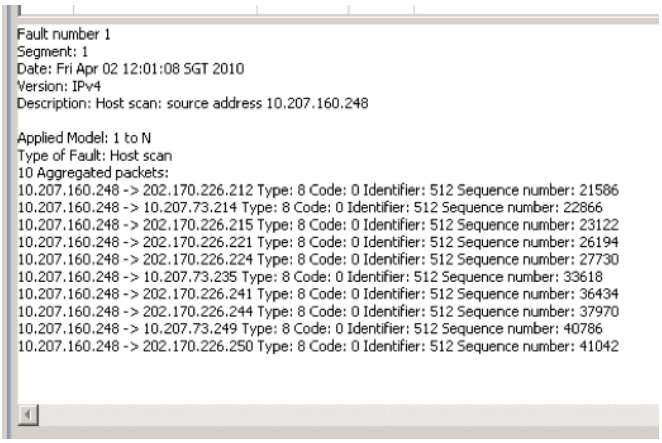


Figure 5: Traffic Aggregation Information Details Log

As the traffic aggregator module aggregate the traffic based on the three different aggregation models and the rule based analyzer is projecting the detection results as presented next. We plug a worm-infected machine into the network testbed. Since the existence of IPv6 based worm is still rare, we infect an IPv4 based worm into the machine. This worm characteristic is known to perform port-scanning activity to spread. Figure 5 shows an interface displaying a traffic aggregation information details pointing to the IP address of the infected machine.

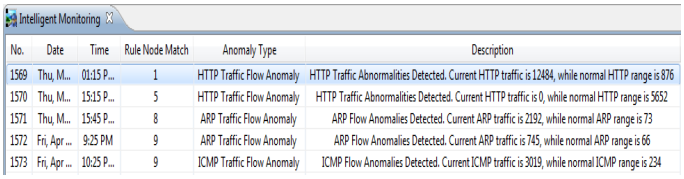


Figure 6: Security Threat Detection Log

In Figure 6, the threat detection logs shows the proposed two-tier approach manage to achieve a promising level of detection. The rule that covers the aggregated HTTP Many-to-One model is fired when we simulate a web server DDoS attack. Similarly, when we simulate a reconnaissance attack, the rule that covers the aggregated ICMP One-to-Many model is fired. In addition, though it is not part of IPv6 security threat, the current preliminary results also shows the ability of the proposed two-tier approach to detect an ARP storm, demonstrated by the firing of the aggregated ARP One-to-Many model's rule.

Next we present an early evaluation of our proposed fuzzy alarms generation scheme. Table 1 shows the alarms generation statistic during the experiment. There are 50 WA, 10 AA, and 2 CA logged. Particularly, one of the CA alarm logged is when we continuously simulate a web server DDoS attack during peak-time period. In terms of false alarms generation, we count our CA as a false alarm if there is no attack at all. Since in our current experiment, all the attacks occurred is simulated attacks, there are no alarms considered as false alarms. However, further experiments are already planned to further test the proposed fuzzy alarms generation scheme from triggering a false alarms.

Table 1: Alarms Generation Statistic

Warning	Attention	Critical
50	10	2

Lastly, we experimented our earliest experimental model of network trend analyzer to predict traffic pattern based on previously captured historical data. The experiment is divided into two phases: historical traffic collector phase and the prediction phase. Figure 7 depicts the output of the experiment. The left statistical chart shows the network traffic utilization statistic on one Tuesday afternoon

whereas the right statistical chart shows the output of the next one-hour traffic trend analysis prediction using the particular historical traffic. These simple preliminary testing shows a promising result on the ability of the triple exponential smoothing to predict traffic pattern. However, a more thorough testing is being planned to further intensify the method evaluation.

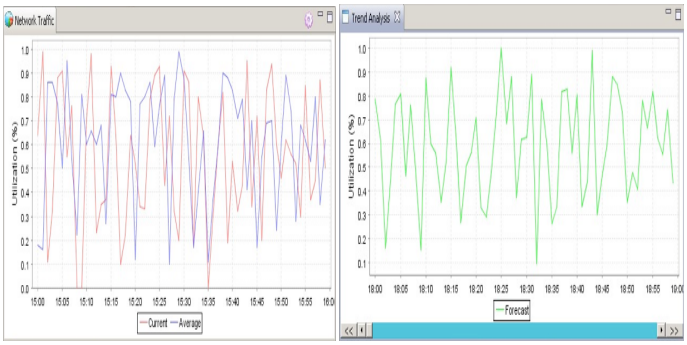


Figure 7: Network Trending based on Historical Data

Conclusion and Future Works

In this paper, we have presented our proposal towards a combined intelligent detection approach to detect IPv6 security threats. The proposed approach focus on network traffic flow based attacks. We also reports some promising preliminary results on experimenting the proposed combined approach in an IPv6 testbed network with simulated attacks.

As a work in progress, our proposed combined approach still needs improvements and intensive testing in many ways for future works. Firstly, the structure of our fuzzy alarms generation scheme still can be improved. Secondly, the proposed two-tier traffic aggregation model also can be improved by considering more IPv6 packet properties in the clustering process. Thirdly, our network trend analysis technique still needs further research and tweaking to enhance the algorithm's ability to forecast network traffic pattern with higher accuracy. And lastly, a more thorough and extensive testing will be carried out to completely evaluate the proposed combined approach.

References

Ford, M., "New Internet Security and Privacy Models Enabled by IPv6," The 2005 Symposium on Applications and the Internet Workshops, 2005. Saint Workshops 2005, vol., no.pp. 2-5, 31-04 Jan. 2005.

Mukherjee, B., Todd, H.L., and Levitt, K.N. "Network Intrusion Detection" IEEE Network, May and June, 8(3): 26 – 41. 1994.

Xia, Z., Lu, S., and Li, J. "Enhancing DDoS Flood Attack Detection via Intelligent Fuzzy Logic" Informatica 34:497-507, 2010.

Li, Y., Li, Z., and Wang, L. "Fuzzy Anomaly Detection System for IPv6 (FADS6): An Immune-Inspired Algorithm with Hash Function". In Intelligent Computing - Lecture Notes in Computer Science (LNCS) vol. 4113, Springer Verlag, 2006.

Wu, S.X. and Banzhaf, W. "The use of computational intelligence in intrusion detection systems: A review" Applied Soft Computing, 10: 1- 35, 2010.

Conta, A. and Deering, S. "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification" (December 1998), RFC 2463 at http://www.ietf.org/rfc/rfc2463.txt

Deering, S. and Hinden, R. "Internet Protocol, Version 6 (IPv6) Specification" (December 1998), RFC 2460 at http://www.ietf.org/rfc/rfc2460.txt

Postel, J. "Internet Protocol, DARPA Internet Program Protocol Specification" (September 1981), RFC 0791 at http://www.ietf.org/rfc/rfc0791.txt

Kent, S. and Atkinson, R. "Security Architecture for the Internet Protocol" (November 1998), RFC 2401 at http://www.ietf.org/rfc/rfc2401.txt

Provos, N., Mavrommatis, P., Rajab, M. A., and Monrose, F. "All your iframes point to us," 2008, pp. 1-15.

Pasha, M.F., Budiarto, R., Syukur, M., and Yamada, M. "EFIS: Evolvable-Neural-Based Fuzzy Inference System and Its Application for Adaptive Network Anomaly Detection". In Daniel Yeung, Xi-zhao WANG, Zhi-qiang Liu and Hong YAN (eds.) Advances in Machine Learning and Cybernetics, Lecturer Notes in Artificial Intelligence (LNAI), Springer-Verlag, 2006.

Abdat, M., Isawasan, P., Pasha, M.F. and Manasrah, A. "On Fuzziness in Hybrid Network Device Discovery", International Conference on Advanced Topics in Artificial Intelligence, November, Phuket, Thailand, 2010.

Kasabov, N. "Evolving Connectionist Systems: Methods and applications in bioinformatics, brain study and intelligent machines". Springer Verlag, London, 2002.

Kanamaru, A., Ohta, K., Kato, N., Mansfield, G., and Nemoto, Y. "A simple packet aggregation technique for fault detection" International Journal of Network Management, 2000

Bartolomei, S. M. and Sweet, A. L. "A Note on a Comparison of Exponential Smoothing Methods for Forecasting Seasonal Series", International Journal of Forecasting, 1989.

Collusion-Resistant Reputation Mechanism for Multi-Agents Systems

Babak Khosravifar, Jamal Bentahar, Maziar Gomrokchi, and Mahsa Alishahi
Concordia University, Canada

Abstract

We address the collusion problem in a reputation multi-agent system where agents represent service providers, consumers, and a controller. A game structure is proposed where players are supposed rational and seek for maximum payoffs. The main issue addressed in this paper is how to maintain a collusion-resistant reputation mechanism. We analyze the behavior of different players with respect to the strategies adopted by the opponents. We provide a theoretical and empirical analysis of the game and discuss the pure and mixed strategy Nash equilibrium along with best response analysis to identify conditions under which the players adopt truthful dominant strategies.

Introduction

Game theory has become an efficient method for computation of solution concepts in multi-agent systems (MAS). The method is useful to obtain the pure or mixed strategy Nash equilibrium considering the environment of interacting agents. Active agents interacting in MAS are concerned about different parameters (application and framework-dependent (Khosravifar *et al.* 2010)). One of them is the public reputation that agents obtain, which reflects their global status among other agents. Game theory can be used to maintain an incentive-based collusion-resistant reputation mechanism. It could be applied to trading systems that involve a number of agents that are selected upon high reputation.

In multi-agent systems (MASs), establishing reputation is a must, and in the recent years different approaches to reputation have been proposed. Reputation is addressed by aggregating related parameters from different perspectives (Maximilien and Singh 2002). One important issue is the agents' tendency to act maliciously to take advantage of the system's vulnerability, which can be done by colluding with other agents. There have been some efforts addressing collusion resistance in reputation frameworks (Jurca and Faltings 2007; Maximilien and Singh 2002). However, some issues such as false alarms are yet to be addressed. In many existing frameworks, the malicious actions and collusion could be maintained by agents at any time (never end). A missing factor in these approaches is convergence towards a truthful setting. In this paper, the conditions that lead to this convergence are investigated using a game structure.

In this paper, we consider a network of providers and consumers, which are equipped with mechanisms capable of maximizing their payoffs. The proposed framework also contains a controller agent whose responsibility is to make the system trustful. A consumer agent is a rational service consumer that initiates requests hoping to obtain an acceptable quality of service (we assume the existence of a function calculating the QoS of the received service as a number between 0 and 1). There is a reputation mechanism that regulates the process of consumers selecting providers by ranking them using their reputation. The controller agent denoted by Cg objectively maintains a sound reputation mechanism by taking the provider and consumer actions under surveillance. As a matter of fact, the Cg applies some penalties in order to discourage the providers and consumers from colluding and acting maliciously.

The proposed framework is similar to the previously proposed models in the literature in the sense that the application of consumer-provider is already seen (Jurca and Faltings 2007). The idea of a controller agent investigating the reputation system is used in (Khosravifar *et al.* 2009). The proposed framework is distinguished from existing frameworks, for example (Jurca and Faltings 2007; Kastidou *et al.* 2009), in the fact that a continuous game involving three players (provider, consumer, and controller agent) is modelled and analyzed. In this framework, the collusion between providers and consumers is thoroughly discussed (skipped in similar works) and the role of Cg in maintaining a sound reputation mechanism is discussed in details. The results of the proposed framework could be summarized as follows: (1) by analyzing different situations, rational agents (providers and consumers) can recognize some restrictions that encourage them to choose the best response leading to their maximum expected payoffs; and (2) the game-theoretic analysis enables the reputation mechanism designer to compute a detection threshold so that if reached, the mechanism will be collusion-resistant. These results are confirmed by the conducted simulations.

The remainder of this paper is organized as follows. In Section 2, we define some preliminaries of our proposed framework. We also describe the game settings between

the entities of MAS in the collusion scenario which involves the provider and consumer agents. In Section 3, we provide a theoretical analysis of the reputation game (involving three agents but grouped into two players) and discuss some results regarding collusion resistant reputation mechanism based on the environment characteristics. We continue our discussions on the same direction of the theoretical analysis in the implemented environment and observe the impacts that environmental characteristics impose on agents' behaviors. Section 4 discusses related work and finally Section 5 concludes the paper.

The Model

This section points out the preliminaries regarding the proposed framework.

Involved Agents

In our proposed model, we consider n consumer agents $u_1, \dots u_n$ (a typical consumer agent is denoted by u), m provider agents $w_1, \dots, w_m$ (a typical provider agent is denoted by w), and a controller agent Cg. Each consumer agent u has a budget account in the system from which his requested services are paid and is equipped with a purchase mechanism that enables him to initiate a service request from a provider w specifying some buying parameters such as response time and price. Each provider agent w is characterized by his reputation, which affects his income in the system since the number of requests he can receive is reputation-dependant. He is also equipped with a selling mechanism that enables him to dynamically set up the selling parameters (response time and price) according to which a service is offered to the consumer agent u. The controller Cg is equipped with a supervision mechanism that enables him to investigate the truthfulness of the interactions among the involved agents and has access to the consumers' accounts and providers' reputations. Agents u and w are examples that we use in the rest of the paper. But they are generalized to a number of consumer and provider agents that keep interacting with one another during their lifetimes.

Reputation Mechanism

The typical agent u posts a feedback f reflecting the extent to which the offered service by the provider agent (say w) is satisfactory. The feedback f belongs to the interval $[-1, 1]$, where $-1, 0, +1$ respectively represent complete dissatisfaction, no answer from the provider, and complete satisfaction. The accumulated feedback (posted by different consumers over time) are used to compute the reputation value of providers. There are a number of frameworks (Maximilien and Singh 2002; Vogiatzis *et al.* 2010) that address the reputation assessment problem ranging from simple mean values to more sophisticated distributions. However, the way the reputation is computed does not have impacts on our results. In fact, we are mainly interested in fraud in the feedback pool. Since choosing a service from the network of services is very selective, provider agents compete to increase their reputation (which is supposed to bring more requests from consumers). In our model, the agent Cg is responsible for investigating the feedback pool to capture malicious actions. What me mean by fraud is that a provider agent w can collude with one or many consumer agents to support him by posting faked positive feedback, which would increase w's reputation.

Consumer-Provider Strategy Profile

The consumer and provider agents follow 2 strategies: being truthful or malicious. In the truthful strategy, the provider provides the service and consumer posts the corresponding truthful feedback to the feedback pool. In the malicious strategy, the provider provides incentives (ϵ) to some consumers to motivate them providing continuous positive feedback even without receiving a service. Colluding with a consumer is aimed at increasing self reputation. This leads to increase provider's market share and thus obtain higher income. Other malicious strategies are possible such as decreasing competitors' reputation. However, to make the paper focused, we only consider the first possibility in this paper.

Collusion Scenario

To make the scenario simple but without losing generality, let us consider only one provider w colluding with one consumer u (the generalization to n consumers is straightforward). The collusion agreement established between w and u clarifies the number of fake positive feedback that u is going to post for w. Associated with the feedback pool is a window that represents a number, publicly known, of feedback that is accumulated in a fixed period of time, for instance an hour or a day. In this scenario, f_w is the percentage of this window that represents the fake positive feedback the consumer would post. The consumer would receive a number of services from the provider. In this case, u is satisfied with the received services and rationally $+1$ would be the corresponding real feedback. But there are extra number of not responded requests that u is also going to post $+1$ for in support of w (these feedback should be 0 and posting 0 would actually decrease the reputation of the provider w). Consequently, the provider receives f_w percentage increase on his positive feedback whereas he has only responded a part of that. Therefore, his reputation increases and thus, the expected number of requests increases as well (we will use ψ to denote the percentage of this increase). In this case, the malicious consumer would receive the amount of ϵ as incentive from the provider. This amount (i.e. ϵ) should be less than the amount gained by the provider ($\lambda_w \psi \beta$, where λ_w represents the mean request number in the fixed period of time relative to the window, and β denotes the price charged by w for offering the service) in the case of adopting the collusion strategy. Therefore, the provider always predicts the expected income before motivating a consumer to adopt the collusion.

Controller-Provider/Consumer Connection

The controller agent Cg continuously investigates the feedback pool and his detection strategy is based on applying some dynamically changing thresholds (this aspect is detailed in Section). Associated with the feedback pool is a

fixed size window that Cg considers in his investigation. The parameter w_c represents the percentage of this fixed size window that Cg is going to analyze. The value w_c would differ over time and the provider and consumer agents need to estimate or predict it when deciding about their strategies of acting truthfully or maliciously. The detection probability reflects the controller agent's accuracy in investigating the feedback pool. This probability could be increased over time when Cg applies some learning mechanisms to use information from past detections in the current investigation. The accuracy of the controller agent would evolve over time and reflects the extent to which the controller agent is confident on analyzing the data. This value is by default a high value which prevents a part of malicious providers to initiate the malicious action. However, since the detection procedure does not affect the results of this paper, we only focus on the role of the parameter w_c. After deciding to penalize the collusion, Cg notifies both the provider and consumer by sending them a report revealing the parameters regarding the penalty process such as the considered window parameter w_c and the percentage df_c of detected fake feedback (explained in the following). The rationality behind this information revealing is the fact that the rational provider would learn the progress of the controller agent and that would influence to some certain extent the tendency of acting maliciously.

Four possibilities can be analyzed:
1. Cg detects the actual collusion and gets $+\pi$ as payoff. This payoff is assumed as incentive to increase self efficiency. The default goal of the controller is to maximize the efficiency. This value does not bring any benefit for the controller, but in general shows the efficiency of the proposed framework. Therefore, the π payoff type is not compared with the payoff associated to the provider and consumer agents. We consider the parameter D_c as the accuracy of detection. In analyzing the feedback pool within the window percentage w_c, Cg catches a percentage of this w_c, denoted by df_c, as the fake feedback and the assigned penalty $\frac{df_c}{w_c}P_n$ to both the provider and consumer is proportional to the detected fake feedback, where the public parameter P_n represents a maximum penalty assigned by Cg. When it affects the consumer, the penalty is applied to his budget (money is taken from his budget account in the system), and when it affects the provider, the penalty concerns his reputation, which then affects his income since the number of received requests will decrease. Moreover, the provider looses some money since the paid balance regarding the fake reputation is being reduced. In this case, the report sent to both the provider and consumer reveals df_c and w_c. It is important to note that the maximum penalty P_n cannot be increased without any condition to avoid high number of false detections that would cause low efficiency of the framework.
2. Cg ignores the actual collusion and gets $-\pi$ as payoff. This is the worst case because the malicious provider increases his reputation and the malicious consumer receives the inventive of ϵ. This case, called false positive, decreases Cg's accuracy in detection.
3. Cg detects the truthful action as collusion and gets $-\pi$ as payoff. This is also harmful in the sense that the truthful provider is losing his reputation and the truthful consumer is losing money because of Cg's mistake. This is the case of false negative that negatively affects the Cg's accuracy.
4. Cg ignores the truthful actions and gets $+\pi$ as payoff. Socially speaking, this corresponds to the best situation, and the objective is to encourage all the players to converge towards such a situation.

Theoretical and Empirical Analysis

In our framework, which we model as a game, 3 players are involved. The first one is the controller agent Cg, which penalizes or ignores the actions performed by the opponents (i.e. provider and consumer agents). Because our objective is to analyze collusion, we consider providers and consumers as one group since collusion takes place only when they cooperate. When one of them refuses to collude, the other has no choice but to play truthfully. In this perspective, which player initiates the collusion is not important. A provider can collude with one or many consumers and visaversa. To simplify the game without losing generality, we consider one-to-one relations and the generalization is simply by summing the different payoffs. Thus we have a game of 3 players forming 2 groups, each player has his own payoff.

There are important details in this game. It is worthy to note that the stage game is the example of a typical moment considering three particular players, which is generalized to a repeated game between all interacting agents. Information obtained from each stage game is used in the rest of agent's involved games. In fact, rational agents consider information obtained in their history interactions for their further decisions. A part of the obtained information is not correct which the agent is not aware of this fact. The discussion on the learning mechanism used by the rational agents is out of scope of this paper and for simplicity, we implement only the obtained data of the history in the implemented environment. In this framework, we focus on identifying the optimum status where the collusion is discouraged at best. To have a better analysis on the payoffs of different players, we consider the situation where the provider and consumer act truthfully and the controller agent ignores the reaction as the ideal case. Therefore, in all the payoffs analysis in the rest of this section, we consider each payoff as the difference of the expected outcome of the chosen case with the ideal case. The expected gain or loss are computed using the variables that have been obtained in the past reports, such as the feedback window parameter w_c and the detected fake feedback df_c. If there is no past report on collusion, we assume the existence of default values that the provider (who has never been penalized) would use to analyze the expected payoff of each case.

If the collusion is detected, the controller agent penalizes the malicious provider and consumer, which would cause $\frac{df_c}{w_c}P_n\lambda_w\beta$ money loss for the provider and $\frac{df_c}{w_c}P_n$ money loss for the consumer. Therefore, the payoff of the provider would be $-\frac{df_c}{w_c}P_n\lambda_w\beta$ and of the consumer would be $-\frac{df_c}{w_c}P_n$ for the case where the collusion was actually

made, and $-\frac{\overline{df_c}}{w_c}P_n\lambda_w\beta$ and $-\frac{\overline{df_c}}{w_c}P_n$ for the case where the controller has falsely considered the truth action as collusion. In this case, $\overline{df_c}$ is considered by the provider as the percentage of falsely detected rightful feedback. These values are previously reported by the controller agent in the penalty report. In case the collusion is ignored, the expected payoff for the provider would be the gained fee over the extra requests minus the incentive given to the consumer $\lambda_w\psi\beta-\epsilon$ (recall that λ_w, ψ, β respectively represent the mean request sent to w, reputation increase amount as a result of collusion, and service fee charged by the provider). This case would bring $+\epsilon$ for the consumer. Finally the case where the provider and consumer act truthfully, they both expect 0, which corresponds to the default case.

Analysis of the Pure Strategy Nash Equilibrium (PSNE)

Table 1: Payoff Table of 3 Players in 2 Groups.

		Group 1 (Cg)	
		Penalize	Ignore
Group 2 (w & u)	Truthful	$+\pi$ $-(df_c/w_c)P_n\lambda_w\beta$, $-(df_c/w_c)P_n$	$-\pi$ $\lambda_w\beta\psi-\varepsilon$, ε
	Collusion	$-\pi$ $-(\overline{df_c}/w_c)P_n\lambda_w\beta$,$-(\overline{df_c}/w_c)P_n$	$+\pi$ 0, 0

Considering the payoffs shown in Table 1, the group 1 is Cg and the group 2 is formed by provider w and consumer u. In pure strategy Nash equilibrium, w and u would consider to collude when the obtained payoff is more than the one obtained in the case of acting truthfully. In the case where Cg ignores the collusion, obviously this collusion brings more payoff ($\lambda_w\psi\beta-\epsilon>0$ for w and $\epsilon>0$ for u). However, if Cg penalizes and if the obtained payoff is more in the collusion case, the group 2 would choose the collusion as the dominant strategy.

Proposition 1 *In the repeated game of the stage game described above, if the falsely detected rightful feedback $\overline{df_c}$ (reported by Cg) is more than the correctly detected fake feedback df_c, the provider and consumer would choose the collusion as the dominant strategy.*

Proof. In this repeated game, the payoffs are the summation of stage game payoffs, so since

$$\overline{df_c} > df_c\ ,\ \sum_{\infty}\lambda_w\psi\beta-\epsilon>0\ ,\ \sum_{\infty}\epsilon>0\ ,$$

$$\sum_{\infty}-\frac{df_c}{w_c}P_n\lambda_w\beta>-\frac{\overline{df_c}}{w_c}P_n\lambda_w\beta \text{ and } \sum_{\infty}-\frac{df_c}{w_c}P_n>-\frac{\overline{df_c}}{w_c}P_n$$

both w and u will always choose to collude whatever the strategy of Cg.

The following corollary is a direct consequence of Proposition 1.

Corollary 1 *In the repeated game of the stage game described above, if the falsely detected rightful feedback $\overline{df_c}$ is more than the correctly detected fake feedback df_c, penalizing the collusion is PSNE.*

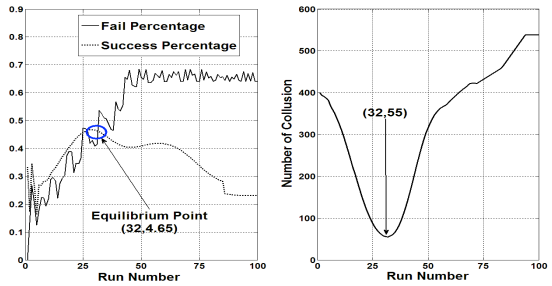


Figure 1: Pure Strategy Nash Characteristics.

If $\overline{df_c}\geq df_c$, then the obtained payoffs of the provider and consumer are both more compared to the case of acting truthful. This is due to the fact that when the controller agent is penalizing, the provider and consumer agents loose less when they choose collusion. Meanwhile when the controller agent is ignoring the action, these agents tend to collude as they gain more payoff. To this end, collusion would be the player 2's dominant strategy, and thus, penalizing the collusion would be the pure Nash equilibrium. Figure 1 shows two simulation results confirming Proposition 1 and Corollary 1. The left graph illustrates successful detection (reflected by df_c) versus failure detection (reflected by $\overline{df_c}$) of a controller agent in a network containing 1000 consumer agents and 200 provider agents. The right graph plots the number of collusion versus the time (number of runs of the simulation), which reflects the overall tendency to act maliciously. In this graph, we observe that the number of collusion increases with $\overline{df_c}-df_c$. The objective in this experiment is to capture the best moment when the least collusion is observed.

In this game, staying in PSNE is, socially speaking, a harmful situation for all the players. Continuing on PSNE, both groups would lose their performances. In fact, the PSNE is achieved due to temporary analysis that players do aiming to maximize their payoffs without considering probabilities of detection. Therefore, we need to consider the case where the payoffs of the players in group 2 are subject to the accuracy of Cg and his detection probability. To analyze the details of this case, we need to consider the mixed strategy Nash equilibrium.

Analysis of the Mixed Strategy Nash Equilibrium (MSNE)

In the mixed strategy, we need to consider two probability distributions on the strategy profiles of the two groups of players (Cg and w & u). Let p be the probability of collusion, which is maintained by the provider and consumer. Therefore, $1-p$ is the probability of acting truthfully. The probability of collusion depends on many factors in the game. In fact, w and u would consider the Cg's accuracy of detection, window size, and expected payoffs before performing any action. Let q be the probability of Cg penalizing the malicious action and therefore, $1-q$ is the probability of ignoring the action. This probability should

satisfy two properties: (1) it increases with the Cg's detection accuracy D_c, which evolves over time; and (2) it also increases with the quality of choosing w_c, the portion of the fixed size window that Cg is going to analyze. Equation 1 gives a definition of this probability that satisfies these properties. In this equation, f_w is the actual percentage of the fake feedback in the feedback pool. Notice that $q \in [0,1]$ since $0 \leq D_c,\ |w_c - f_w| \leq 1$. If the detection is completely wrong, then $|w_c - f_w| = 1$, and if it is completely correct, then $|w_c - f_w| = 0$, which means Cg is investigating a portion of feedback from the feedback pool where all of them are fake.

$$q = D_c(1 - |w_c - f_w|) \tag{1}$$

We define $Cg.pro = (q, 1-q)$ as the probability distribution of the Cg's strategy profile and $wu.pro = (p, 1-p)$ as the probability distribution of the strategy profile of group 2 players w and u. In the mixed strategy case, all the players need to estimate their expected payoffs with respect to their chosen strategies against the other player's probability distribution. Let $E_w(C, Cg.pro)$ (resp. $E_w(T, Cg.pro)$) be the expected payoff of the provider choosing to collude (resp. to act truthfully) against the probability distribution of the controller agent. In the same way we define $E_u(C, Cg.pro)$, $E_u(T, Cg.pro)$, $E_{Cg}(P, wu.pro)$ (P for penalizing) and $E_{Cg}(I, wu.pro)$ (I for ignoring). We obtain the expected values as follows:

$$\begin{cases} E_w(C, Cg.pro) = [-\frac{df_c}{w_c} P_n \lambda_w \beta](q) + [\lambda_w \psi \beta - \epsilon](1-q) \\ E_w(T, Cg.pro) = [-\frac{\overline{df_c}}{w_c} P_n \lambda_w \beta](q) + [0](1-q) \end{cases} \tag{2}$$

$$\begin{cases} E_u(C, Cg.pro) = [-\frac{df_c}{w_c} P_n](q) + [\epsilon](1-q) \\ E_u(T, Cg.pro) = [-\frac{\overline{df_c}}{w_c} P_n](q) + [0](1-q) \end{cases} \tag{3}$$

$$\begin{cases} E_{Cg}(P, wu.pro) = [+\pi](p) + [-\pi](1-p) \\ E_{Cg}(I, wu.pro) = [-\pi](p) + [+\pi](1-p) \end{cases} \tag{4}$$

Best Response Analysis

All the players in this mixed strategy game aim at maximizing their payoffs. Therefore, for all their adopted strategies, we need to consider the best responses and discard the other strategies. For instance, the provider, who is seeking for maximizing his payoff by choosing a specific strategy, would discard any strategy in which his expected payoff is less than any other strategy. Since each player in each stage game chooses between only two strategies, and since any of these strategies could be the best response in a particular situation, we analyze the case where the expected payoffs are equal. By so doing, we can compute a threshold, which is used to identify which strategy is dominant. If these expected payoffs are not equal, that means one strategy would lead to a higher payoff and therefore, the player would select that strategy as dominant. In this case, the mixed strategy probabilities would be (1,0) or (0,1), which is back to the pure strategy case.

The best response analysis in our mixed strategy game would enable the provider w and consumer u to obtain a probability threshold (μ) for their chosen strategy. Once the threshold is obtained, w and u would estimate the probability of the action to be chosen by Cg, and by comparing this probability with the threshold, they choose the best strategy that maximizes their payoffs. In the repeated mixed strategy game, w and u estimate the probability (denoted by q_w) that Cg penalizes their action based on previous detections. This probability should satisfy the same properties as for Equation1 and is computed in Equation 5. In this Equation, the accuracy of Cg is public, but the window parameter $\overline{w_c}$ is considered as the window parameter that is reported to w and u in the most recent penalty report. The parameter f_w is also known to the provider and consumer. This probability could be compared against the obtained threshold μ by w and u to maximize their expected payoffs.

$$q_w = D_c(1 - |\overline{w_c} - f_w|) \tag{5}$$

Theorem 1 *In the mixed strategy repeated game, there is a threshold μ such that if $q_w > \mu$, acting truthfully is the dominant strategy for group 2. Otherwise, colluding is the dominant strategy.*

Proof. We prove that the theorem holds for each stage game, so the repeated case follows. To find the dominant strategy for each stage game, we need to consider the case where the payoffs of the different strategies are the same. We have:

$$E_w(C, Cg.pro) = E_w(T, Cg.pro) \Rightarrow$$

$$q_w = \frac{\lambda_w \psi \beta - \epsilon}{\frac{df_c - \overline{df_c}}{w_c} P_n \lambda_w \beta + \lambda_w \psi \beta - \epsilon} = \mu_w$$

$$E_u(C, Cg.pro) = E_u(T, Cg.pro) \Rightarrow$$

$$q_w = \frac{\epsilon}{\frac{df_c - \overline{df_c}}{w_c} P_n + \epsilon} = \mu_u$$

Let $\mu = max(\mu_w, \mu_u)$. If $q_w > \mu$, then $E_w(C, Cg.pro) < E_w(T, Cg.pro)$ and $E_u(C, Cg.pro) < E_u(T, Cg.pro)$, so the first part of the theorem follows, and the second part is straightforward. □

The threshold μ is as follows:

$$\mu = max(\mu_w, \mu_u) = \begin{cases} \mu_w & \text{if } 2\epsilon < \lambda_m \psi \beta \\ \mu_u & \text{otherwise} \end{cases} \tag{6}$$

Corollary 2 that follows directly from Theorem 1 gives the the condition under which the MSNE is obtained.

Corollary 2 *If the estimated probability of penalizing q_w exceeds the obtained threshold μ, then acting truthfully by w and u and being ignored by Cg is a MSNE.*

Proof. *By exceeding this threshold, the provider and consumer would choose to act truthful as their dominant strategies. Therefore, the controller agent would find better payoff by ignoring this action.*

Theorem 2 *A collusion-resistant reputation mechanism is achieved when the controller agent maximizes the value of $\frac{df_c - \overline{df_c}}{w_c}$.*

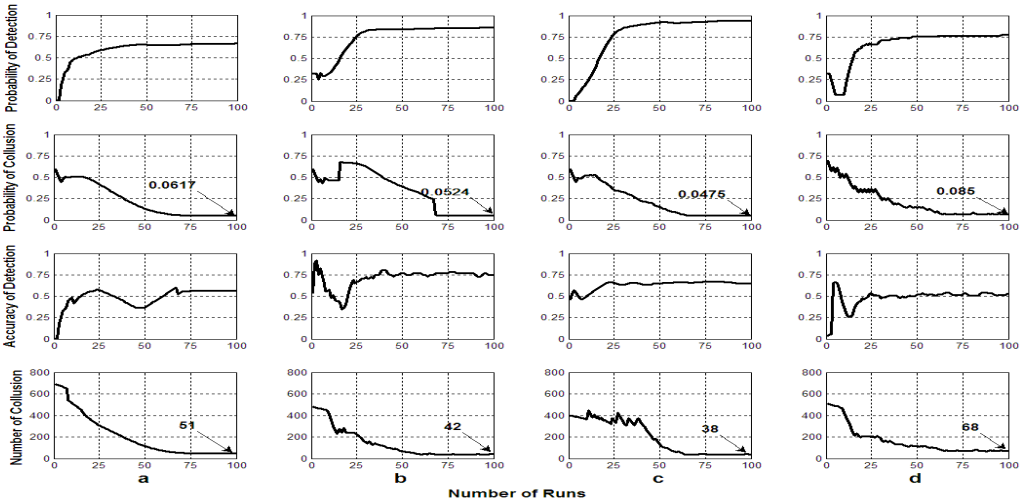


Figure 2: Mixed Strategy Nash Characteristics where in column (a) $w_c = 20\%$, in column (b) $w_c = 40\%$, in column (c) $w_c = 60\%$, and in column (d) $w_c = 80\%$.

Proof. From Theorem 1, we deduce that since $q_w \in [0, 1]$, if $\mu \geq 1$, then collusion would be the dominant strategy for the provider and consumer. From Equation 6 and the proof of Theorem 1, $\mu \geq 1$ occurs when $\overline{df_c} \geq df_c$. Therefore, to have a collusion-resistant mechanism, Cg would aim at minimizing the threshold μ used by the opponents since the estimated probability of penalizing is compared to this threshold. According to Equation 6, to minimize μ in both cases, Cg has to maximize the value at the denominator ($\frac{df_c - \overline{df_c}}{w_c} P_n$) as all the other factors are out of Cg's control. □

In the repeated game, Cg has to consider the best window factor w_c that corresponds to the fake feedback submitted to the feedback pool. The value of w_c could be learned by Cg over time (while the detection accuracy D_c also increases over time). However, finding the the best w_c is not guaranteed since there is always a risk of different fake feedback percentage maintained by the provider and consumer as a result of collusion agreement. Therefore, there is always a risk of false negative in penalizing the provider and consumer.

This reflects the fact that the accuracy of the controller agent in penalizing the collusion has a crucial impact on the analysis that provider and the consumer maintain to adopt their strategies. If the window parameter w_c is chosen with respect to some learning mechanism that maximizes $\frac{df_c - \overline{df_c}}{w_c} P_n$, the repeated game would obtain the truthful action followed by ignore of the controller agent as a situation that leads to a collusion-resistant reputation mechanism.

Empirical Observations

The learning process of the parameter w_c is skipped in this paper; however, we have analyzed the impact imposed by changing this variable in our empirical analysis. We have implemented an environment holding the explained characteristics of the rational agents. We ran a simulation consisting of 100 runs, which reflect 100 (simulated) days. We linked the fixed window value to the total feedback accumulated in each run. Therefore, the parameters w_c, f_w, and df_c are all percentages of this fixed window. We ran the simulation with 4 different mean window parameters w_c (in columns a, b, c, and d of Figure 2) and observed 4 different results: (1) the controller agent's probability of detection evolution; (2) the provider and consumer agents' probability of collusion evolving over simulation runs; (3) the controller agent's accuracy of detection; and (4) the number of successful collusion that leads to positive payoff for the colluding agents.

As shown in Figure 2, 4 different mean window parameters w_c reflect different results. Overall, in these simulations, we obtain better results when the mean w_c (as the percentage of the public window) is equal to 40% and 60%. The probability of detection evolves better in these cases compared to the ones where the mean w_c is equal to 20% or 80%. The reason behind this is the fact that Cg would generate more false positives, which means penalizing the truthful action by mistakes when the mean w_c is equal to 20% and more false negatives, which means ignoring the collusion by mistake since the window of investigation is considered too general when it is equal to 80%. To this end, there would be less number of successful detections that Cg can use in his further investigations. The probability of collusion p is decreasing in all the simulations with slightly different slopes. It is obvious that the collusion would never get to 0 as there are always risks of obtaining payoffs by colluding. This risk is totally related to the accuracy of the controller agent together with the probability of penalizing q_w. Cg's accuracy in detection is also improved over runs as the histories of previ-

ous detections are used in his further investigation process. However, the investigation procedure is linked to the window parameter that the controller considers and thus, only the reports would not enable the controller agent to detect all the collusion. Number of successful collusion is also decreasing in all the simulations, but with different slopes. The successful collusion is considered as the one that is ignored by the controller agent.

Related Work

We consider two groups of frameworks related to our proposed model. The first group is about approaches that have been proposed for reputation assessment in MAS. Maximilien and Singh (2002) provided a complete overview regarding the parameters required to evaluate an agent's reputation. Vogiatzis et al. (2010) proposed a probabilistic model for computing the trust and reputation of rational agents in MAS. They proposed a set of assumptions that enable recognizing the agents that change their behavior to estimate their rates of change. Recently, Chapman et al. (2011) developed a unified analytical framework for distributed agents investigating the best response with respect to sequential optimization problems, which consist of state evaluation, decision rule, and adjustment schedule procedures. Doing so, agents analyze the best response, which leads to their maximum payoff.

The second group are approaches proposed to develop sound reputation mechanisms, where entities are encouraged to act truthfully (Jurca *et al.* 2007; Khosravifar *et al.* 2010; Kastidou *et al.* 2009). Khosravifar et al. (2010) developed a game-theoretic approach categorized in one-shot and repeated games to emphasize the learning mechanism used by the agents, which leads them favor truth telling. Kastidou et al. (2009) developed a framework representing safe information exchange regarding reputation assessment and agents are motivated to honestly report reputation information. All these approaches aim at maintaining a sound reputation mechanism according to aggregation frameworks similar to the ones proposed in the first group. However, these frameworks consider scenarios where only a part of the network (either the provider or consumer) play important role, which reflect partial collusion, i.e. only between agents of the same type. We believe that in dynamic MAS, a sound reputation system should consider the benefits of all the players.

The closest model to our work is the collusion-resistant reputation mechanism proposed in (Jurca and Faltings 2007) in which the authors investigate incentive-compatible payment mechanisms. Malicious agents' behaviors are analyzed in different scenarios to maintain an automated mechanism design that yields the best reputation assessment in the environment. The proposed mechanism is limited in the sense that it is based on a linear optimization setting that do not perform well in specific situations like when false alarms are generated through the reputation mechanism. In general, the reputation mechanism is assumed to function correctly, which makes lying a Nash equilibrium at some point. In our paper, we investigate the collusion and address the collusion-resistant setting under the assumption that the controller agent is also making mistakes. This overall encourages the provider and consumer agents to analyze the characteristics of the controller through the collusion reports sent to them.

Conclusion

The contribution of this paper is the proposition of a collusion-resistant reputation mechanism applicable to multi agent settings. The reputation mechanism is supervised by a controller agent that reveals the reputation value of provider agents to the consumers. The proposed approach investigates the settings under which the truth action is the Nash equilibrium. In the game-theoretic analysis, rational agents compute their expected payoff and according to the available information from past events in the game, they decide about their further actions. The objective of the controller agent is to update the settings such that the truth action yields the maximum gain for the provider and consumer agents. Our model is distinguished from related work in the sense that a game-theoretic (followed by empirical) analysis considering all the agents that are involved in the reputation mechanism is provided together with its aggregation process. Moreover, false alarms generated by the controller agent are also considered by the providers and consumers in their strategies and being investigated in experiments.

Our plan for future work is to integrate optimization techniques to our game and investigate in details the best value for w_c, which potentially impacts the expected payoffs computed by the provider and consumer agents. We also need to analyze in depth the scenario where some providers can collude to destroy the reputation of others by causing false alarms generated by the controller.

References

Archie Chapman, Alex Rogers, Nicholas Jennings, and David Leslie. A unifying framework for iterative approximate bestresponse algorithms for distributed constraint optimisation problems. *Knowledge Engineering Review (in press)*, 2011.

Radu Jurca and Boi Faltings. Collusion-resistant, incentive-compatible feedback payments. In *Proc. of the ACM Conf. on E-Commerce*, pages 200–209, 2007.

Radu Jurca, Boi Faltings, and Walter Binder. Reliable QoS monitoring based on client feedback. In *Proc. of the 16'th Int. World Wide Web Conf.*, pages 1003–1011, 2007.

Georgia Kastidou, Kate Larson, and Robin Cohen. Exchanging reputation information between communities: A payment-function approach. In *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 195–200, 2009.

B. Khosravifar, J. Bentahar, P. Thiran, A. Moazin, and A. Guiot. An approach to incentive-based reputation for communities of web services. In *Proc. of IEEE 7'th Int. Con. on Web Services (ICWS)*, pages 303–310, 2009.

Babak Khosravifar, Jamal Bentahar, Ahmed Moazin, and Philippe Thiran. On the reputation of agent-based web ser-

vices. In *Proc. of the 24'th Conf. on Artificial Intelligence (AAAI)*, pages 1352–1357, 2010.

E. Michael Maximilien and Munindar P. Singh. Conceptual model of web service reputation. *SIGMOD Record, ACM Special Interest Group on Management of Data*, 31(4):36–41, 2002.

George Vogiatzis, Ian MacGillivray, and Maria Chli. A probabilistic model for trust and reputation. In *Proc. of 9'th Int. Conf. on Authonomous Agent and Multi Agent Systems (AAMAS)*, pages 225–232, 2010.

Alert correlation in intrusion detection: Combining AI-based approaches for exploiting security operators' knowledge and preferences

Karim Tabia[1], **Salem Benferhat**[1], **Philippe Leray**[2], **Ludovic Mé**[3]
[1] Universite Lille-Nord de France, Artois, CRIL, CNRS UMR 8188, F-62307 Lens
{tabia,benferhat}@cril.univ-artois.fr
[2] LINA/COD UMR CNRS 6241 Ecole Polytechnique de l'université de Nantes
philippe.Leray@univ-nantes.fr
[3] Supélec, SSIR Group (EA 4039)
ludovic.me@supelec.fr

Abstract

Alert correlation is a crucial problem for monitoring and securing computer networks. It consists in analyzing the alerts triggered by intrusion detection systems (IDSs) and other security related tools in order to detect complex attack plans, discover false alerts, etc. The huge amounts of alerts raised continuously by IDSs and the impossibility for security operators to efficiently analyze them requires tools for eliminating false and redundant alerts on the one hand and prioritize them according the detected activities' dangerousness and preferences of the analysts on the other hand. In this paper, we describe an architecture that combines AI-based approaches for representing and reasoning with security operators' knowledge and preferences. Moreover, this architecture allows to combines experts' knowledge with machine learning and classifier based tools. This prototype collects the alerts raised by security related tools and analyzes them automatically. We first propose formalisms for representing both background and contextual knowledge on the monitored network, known attacks and vulnerabilities. We then propose another logic-based formalism for representing and reasoning with operators' preferences regarding the events and alerts they want analyze in priority. We after that propose probabilistic models for detecting and predicting attack plans and severe attacks. Finally, we provide further discussions and future work directions.

Introduction

Computer security continuously faces new problems and challenges as information systems become more networked and technologies are changing and increasingly complex, open and dynamic. There are two kinds of solutions that are currently deployed in order to ensure the integrity, confidentiality or availability of computer and network resources/services: *preventive solutions* (such as firewalls and access control systems) aiming at preventing malicious users from performing unauthorized actions and *detective solutions* such as intrusion detection systems (IDSs) whose objective is detecting any malicious action targeting the information system resources and services (Axelsson 2000). IDSs act as burglar alarms and they are either signature-based (Roesch 1999) or anomaly-based (Patcha and Park 2007) or a combination of both the approaches.

Computer security practitioners often deploy multiple security products and solutions in order to increase the detection rates by exploiting their mutual complementarities. For instance, signature-based IDSs are often combined with anomaly-based ones in order to detect both known and novel attacks and anomalies. It is important to note that all existing anomaly-based approaches have a major drawback consisting in very high false alarm rates. These systems build profiles and models of legitimate activities and detect attacks by computing the deviations of the analyzed activities from normal activity profiles. In the literature, most anomaly-based IDSs are novelty or outlier approaches (Patcha and Park 2007)(Smith et al. 2008) adapted for the intrusion detection problem. Moreover, all modern IDSs (even the *de facto* standard network IDS Snort[1]) are well-known to trigger large amounts of alerts most of which are redundant and false ones. This problem is due to several reasons such as bad parameter settings and inappropriate IDS tuning, etc. (Tjhai et al. 2008). As a consequence, huge amounts of alerts are daily reported making the task of the security administrators time-consuming and inefficient. In order to cope with such quantities of alerts, alert correlation approaches are used (Debar and Wespi 2001)(Cuppens and Miège 2002).

Alert correlation is the task of analyzing the alerts triggered by one or multiple IDSs in order to provide a *synthetic* and *high-level* view of the *interesting* malicious events targeting the information system. Alert correlation approaches aim either at reducing the number of triggered alerts by eliminating redundant and irrelevant ones (Debar and Wespi 2001) or detecting multi-step attacks (Ning, Cui, and Reeves 2002) where the different alerts may correspond to the execution of an attack plan consisting in several steps.

Most alert correlation approaches address only some specific issues such as aggregating similar alerts, detecting attack scenarios, etc. Regarding the used approaches, most are statistically based and do not combine heterogeneous approaches and formalisms. In this paper, we propose an alert correlation prototype that addresses several alert correlation issues and combines different AI-based approaches.

[1] www.snort.org

More precisely, this prototype can be used to (i) discard false alerts, (ii) prioritize alerts, (iii) detect attack scenarios, (iv) predict severe attacks and (v) handle the IDSs reliability when reasoning about the alerts. Our prototype combines logic-based formalisms and probabilistic ones to help security operators. The prototype and contributions we synthesize in this paper are done in the framework of a French research project named PLACID [2]. We provide brief descriptions of our key contributions in this project and give references to published works detailing each issue.

Alert correlation: Approaches and challenges

In this section, we briefly review existing approaches in the alert correlation field then present the two main challenges related to this problem and for which we propose a solution. The input data for alert correlation tools is gathered from various sources such as IDSs, firewalls, Web server logs, etc. Correlating alerts reported by multiple analyzers and sources has several advantages such as exploiting the complementarities of multiple analyzers. The main objectives of alert correlation are:

1. *Alert reduction and Redundant alerts elimination:* The objective of alert correlation here is to eliminate redundant alerts by aggregating or fusing similar alerts (Debar and Wespi 2001). In fact, IDSs often trigger large amounts of redundant alerts due to the multiplicity of IDSs and the repetitiveness of some malicious events such scans, floodings, etc.
2. *Multi-step attack detection:* Most IDSs report only elementary malicious events while several attacks perform through multiple steps where each step can be reported by an alert. Detecting multi-step attacks requires analyzing the relationships and connections between several alerts (Bin and Ghorbani 2006)(Ning, Cui, and Reeves 2002).

In the literature, alert correlation approaches are often grouped into similarity-based approaches (Debar and Wespi 2001), predefined attack scenarios (Ning, Cui, and Reeves 2002), pre and post-conditions of individual attacks (Cuppens and Miège 2002) and statistical approaches (Valdes and Skinner 2001)(Julisch and Dacier 2002).

In most similarity-based approaches, the objective consists in reducing the large volumes of alerts generated by the analyzers by aggregating *similar* ones on the basis of their features (victim/attacker IP addresses, etc.). Examples of such approaches can be found in (Debar and Wespi 2001)(Valdes and Skinner 2001)(Dain and Cunningham 2001).

Approaches based on pre/post-conditions aim at detecting whether an attack plan (also called *attack scenario*) is in progress. An attack plan designates a complex multi-step attack consisting in several malicious actions. It often consists in a set of actions executed in a predefined sequence. Hence, in order to detect attack plans, there is a need to first detect the individual actions and correlate them in order to find which attack plan is ongoing. Pre/post-condition approaches encode attack plans by specifying for each action its pre-conditions (the actions/conditions that must executed/fulfilled before executing the current one) and post-conditions (corresponding generally to the consequences of an action). In (Al-Mamory and Zhang 2009), the authors propose a grammar-based approach to encode attack plans. In (Ning, Cui, and Reeves 2002), the authors propose a logic-based approach for attack scenario detection while it is a graph representation-based approach that is used in (Lingyu, Anyi, and Sushil 2006). It is important to note that most works on multi-step attack detection heavily rely on expert's knowledge. For instance, the model proposed in (Cuppens and Miège 2002) requires identifying for each elementary attack, the preceding attacks and its consequences. In (Ilgun, Kemmerer, and Porras 1995), the authors propose an approach based on state transition analysis where the knowledge on existing multi-step attacks is encoded using a dedicated language STATL.

Several works propose statistical and data mining techniques for the alert correlation problem. The advantage of these techniques is their ability to exploit large data volumes and they do not require lot of expert knowledge. For instance, in (Dain and Cunningham 2001) the authors apply clustering and data mining approaches to discover attack clusters which are then aggregated.

One of the most important issues in alert correlation that has not received much attention is the one of alert prioritization. Among the huge amounts of triggered alerts, security administrators must select a subset of alerts according to their dangerousness and the current contexts. Alert filtering/prioritization aims at presenting to the administrators only the alerts they want to analyze. The other problem which has not been appropriately addressed is the one of handling IDSs' reliability given that these tools often trigger false alerts, issue imprecise information, etc. The prototype described in this paper addresses these issues.

Several works highlighted the need to combine different approaches and tools to deal with alert correlation issues. Among these works, the ones of (Goldman and Harp 2009)(Goldman et al. 2001) are similar to ours in the sense that they combine probabilistic network-based approaches with some background knowledge (security policy, event dictionary, etc.) encoded within an intrusion reference model (using an ontology). However, in these works the authors deal essentially with data reduction (by fusing the reports raised by IDSs) and plausibility assessment. They search for relevant and plausible events and do not consider the issue of event sorting and filtering according the security operators preferences. Moreover, their works do not deal with severe attack prediction. In (Li and Tian 2010), the authors also propose a multi-agent alert correlation architecture using an ontology to encode the background knowledge but do not deal with alert prioritization and predicting severe attacks.

Model overview

The objective targeted in our works is to build a complete alert correlation system capable to

- collect and reason with data issued by several sources and analyzers (IDSs, firewalls, etc.),
- encode and reason with expert knowledge and prefer-

[2] http://placid.insa-rouen.fr/

ences,

- build models for detection and prediction purposes capable of handling the uncertainty regarding the observations and analyzers' reliability.

Our model is based on two principal modules that interact through querying interfaces. Each module is equipped with a preprocessing tool and reasoning engines to answer the users' queries or the others engines' ones. As shown in Figure 1 , the data collected from IDSs and other security related sources is first normalized and preprocessed in order to be used by our reasoning engines. Once the data is preprocessed, it is used by the knowledge and preferences engine for detecting for instance false alerts. The same data is also used by the second module to detect complex attacks and severe ones. The output of the reasoning engines are a set of prioritized alerts: those that are directly raised by the IDSs or those corresponding to the malicious events (for instance attack plans) that are detected by the probabilistic-based reasoner. The outputs are sorted according the security operators' preferences.

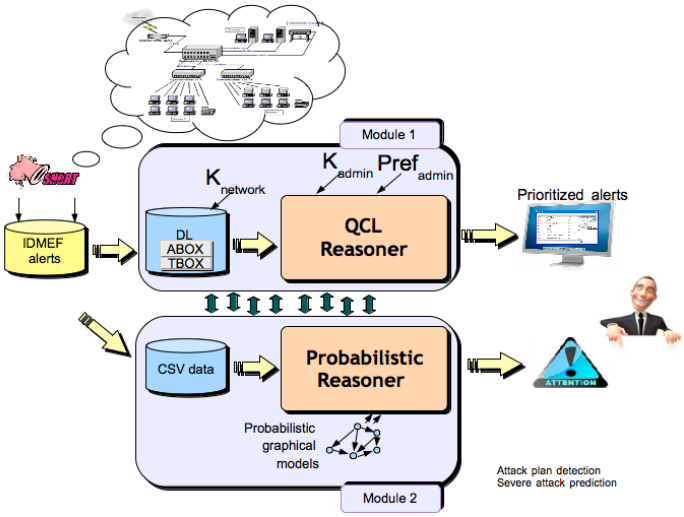


Figure 1: Building blocks of our alert correlation system

Knowledge and preferences module

This module aims to allow the security operators to represent their knowledge and preferences. The first sub-module contains the domain knowledge about the alerts, existing attacks, vulnerabilities, sensors and analyzers, victims and attackers. The second one aims to represent the security operators' preferences in order to give for instance more priority to the ones considered by the operators as the most dangerous. This module uses logic-based formalisms for representing and reasoning with the security operators' information.

Detection and prediction module

The detection and prediction module consists of probabilistic graphical models used to detect complex attacks or attack plans, predict severe attacks and handle the IDSs' reliability. More precisely, this module uses causal Bayesian networks (Jensen and Nielsen 2007) and Bayesian network classifiers (Friedman et al. 1997) to achieve the detection and prediction of attack plans and severe attacks respectively. Bayesian networks are efficient tools for representing and reasoning with uncertain information. In particular, they allow an easy and compact representation of uncertain information, they can be automatically learnt from empirical data and they are very efficient in prediction tasks such as classification.
The data analyzed by our model consists in the raw alerts raised by IDSs and other security monitoring tools. This data is collected in the IDMEF[3] format. An example of a preprocessing task is the one of transforming IDMEF alerts into presence/absence data in order to detect complex attacks. As it is shown in Figure 1, the two modules composing our architecture are designed to communicate in order to exchange relevant information regarding the tasks they perform. For instance, the attack plan detection model would need to know whether the detected attack presents a danger in case where the victim machine is actually vulnerable to the detected attack. This information can be obtained by querying the knowledge on the attacks and vulnerabilities represented in the knowledge and preferences module. In the following three sections, we provide descriptions on the formalisms and reasoners involved in our alert correlation system.

Representing background and contextual knowledge

The relevant knowledge for our alert correlation tasks is of two kinds : Knowledge on the monitored network and knowledge on attacks and vulnerabilities.
Knowledge on the network : This knowledge is needed in order to capture the architecture of the network and its software and hardware configuration. For instance, we build a corpus of the core concepts (machine, software, operating system, sub-network, IDS, etc.) and represent the configuration and topology of the network using this vocabulary.
Knowledge on attacks and vulnerabilities : Similarly to the knowledge on the network, we first define a corpus of core concepts to represent attacks, vulnerabilities, sources, etc. then encode known attacks and vulnerabilities.
The formalisms we relied on here are based on IDDL, an Intrusion Detection Description Language we especially developed for alert correlation purposes. The choice of description logics (DL) is motivated by the need to develop an intrusion detection description language that can provide security components with a formal framework to characterize their observations, share their knowledge with third-party components and reason about complementary evidence information. Moreover, IDDL can be supported by the existing DL-based reasoning engines.

IDDL - Intrusion Detection Description Language

The definition of a shared vocabulary to describe the different information is of a major importance for our alert corre-

[3]IDMEF stands for the Intrusion Detection Message Exchange Format. `http://www.ietf.org/rfc/rfc4765.txt`

lation application. This information is structured, often represented using the XML standard. For instance, this is the case of alerts reported in IDMEF format as well as vulnerabilities in OVAL (Open Vulnerability and Assessment Language). However, XML is limited to a syntactic representation. Given that this representation is devoid of semantics, we propose to use a fragment of first order logic, namely Description Logics (DL for short), to represent contextual information in intrusion detection. In fact, DLs are convenient to represent structured information. Moreover, they are decidable in the sense that reasoning can be achieved in a finite time. Moreover, a number of sophisticated DL-based reasoners have been developed like FaCT++[4] and Pellet[5]. DLs have been used in many applications like Semantic Web where they constitute the formal basis for a number of ontology languages like OWL. A DL knowledge base includes two components:

- A TBox (terminological part) introducing the terminology, i.e., the vocabulary of the domain of the application at hand and
- an ABox (assertional part) containing assertions with respect to instances in terms of this vocabulary.

The vocabulary is a set of concepts, which denotes sets of individuals, and roles which refer to binary relations between instances. Using DLs, our alert correlation system can easily share contextual information with the other modules to monitor and predict some severe attacks for instance. Here, the knowledge base encoded in DLs is queried through the communication interface to provide inputs to the detection and prediction module. This information is more appropriate than directly using raw alerts provided by IDSs. For instance, using our DLs representation, some conflicts between analyzers can be avoided and the volume of alerts reduced. Let us now give some examples of relevant knowledge represented in IDDL.

1. *IDMEF alerts in IDDL:* In order to reason about the reported alerts, we need to represent them in IDDL. An alert in IDMEF consists in a set of attributes such as $Identifier$, $CreateTime$, $DetectTime$, $AnalyserTime$, $Analyser$, $Source$, $Target$, etc. In order to represent IDMEF alerts in IDDL, we built a TBox containing definition axioms as well as inclusion axioms. For instance, the concept of an alert is encoded as follows:
   Alert$\sqsubseteq\forall$messageId.String $\sqcap$ = 1 messageId $\sqcap$
   hasCreateTime.Time $\sqcap$= 1 hasCreateTime $\sqcap$
   hasDetectTime.Time $\sqcap\leq$ 1 hasDetectTime $\sqcap$
   hasAnalyserTime.Time $\sqcap\leq$ 1 hasAnalyserTime $\sqcap$
   hasAnalyser.Analyser $\sqcap$= 1 hasAnalyser $\sqcap$
   hasSource.Source $\sqcap$
   hasTarget.Target $\sqcap$
   hasClassification.Classification $\sqcap$ = 1 hasClassification $\sqcap$
   hasAssessement.Assessment $\sqcap\leq$ 1 hasAssessment $\sqcap$
   hasAdditionalData.AdditionalData
   Such an axiom means that an alert admits a unique identifier which is a string, a unique field $DetectTime$ of time type, a unique field $CreateTime$ of time type, etc. Moreover, with an alert, we can associate a source (resp. target) or many. Besides, an alert has a unique classification, at most a field assessment and a field additional data.
2. *Topology in IDDL:* The topology of the network is needed for example to know whether an IDS is capable to detect a given alert. It denotes the nodes as well as their interconnexions. In the M4D4 model, each network is identified by a unique network address. This information is encoded in IDDL as follows:
   Network $\sqsubseteq\forall$netaddress.String$\sqcap$=1 netaddress

   A node denotes any computer connected to the network and it belongs to a network.
   Node $\sqsubseteq\forall$ nodeaddress.String$\sqcap$= 1 nodeaddress$\sqcap$
   $\forall$hasNodeNet.Network

   As for gateways, they are particular nodes interconnecting networks. Hence, a gateway belongs to more than one network.
   Gateway$\sqsubseteq$Node$\sqcap>$1 hasNodeNet.Node$\sqcap$
   $\neg$Gateway$\sqsubseteq$= 1 hasNodeNet
3. *Computer configuration in IDDL:* We provide here examples for encoding some core concepts in our IDDL language: Software, Node, Process and Service. According to the M4D4 model [6], a software product is characterized by a unique name, a unique version, a unique type and a unique architecture. In IDDL, this corresponds to the following $Software$ concept:
   Software$\sqsubseteq$
   softwareName.String$\sqcap$= 1 softwareName$\sqcap$
   softwareVersion.String$\sqcap$= 1 softwareVersion$\sqcap$
   $\forall$softwareType.String$\sqcap$= 1 softwareType$\sqcap$
   $\forall$softwareArchitecture.String$\sqcap$= 1 softwareArchitecture

   The concept "process" consisting in a software executed by a user is encoded by:
   Process$\sqsubseteq\forall$hasSoftware.Software$\sqcap$= 1 hasProduct$\sqcap$
   $\forall$hasUser.User$\sqcap$= 1 hasUser

   Similarly, the concept "service" is defined as a process listening on one port:
   Service$\sqsubseteq\forall$hasProcess.Process$\sqcap$= 1
   hasProcess$\sqcap\forall$port.Integer$\sqcap$ = 1 port
4. *Vulnerabilities in IDDL:* Vulnerabilities are failures and weaknesses in a system that can be used to achieve malicious actions. A vulnerability is often characterized by its severity (dangerousness), the access level needed to exploit it, its potential consequences and its publication date. This is encoded as follows:

   Vulnerability$\sqsubseteq$ $\forall$severity.{high, medium, low}
   $\forall$requires.{remote, local, user} $\sqcap$
   $\forall$losstype.{confidentiality, integrity,
   availibality, privilege escalation} $\sqcap$
   $\forall$published.Date

[4] http://www.cs.man.ac.uk/~horrocks/FaCT/
[5] http://clarkparsia.com/pellet

[6] M4D4 is a first order logic-based data model used in computer security to query and assert information about security related events and incidents and the actual context where they happen (Mé et al. 2008).

In the M4D4 model, a vulnerability is related to a list of products (software). The properties of vulnerabilities can be extracted from several online databases (such as NVD, OSVDB and OVAL).

More details on representing security operators' knowledge can be found in (Yahi, Benferhat, and Kenaza 2010) and in technical reports of the PLACID project. We implemented our IDDL engine using the Pellet[7] reasoner. In the following section, we present the way we represent and reason with security operators' preferences.

Representing security operators' preferences

Representing and reasoning with security operators' preferences is a key issue for designing efficient alert correlation systems. Our system relies on extensions of Qualitative Choice Logic (QCL) (Brewka, Benferhat, and Le Berre 2004) to deal with this issue. Our objective here is to develop logics that can

- Represent all the alerts with their attributes.
- Extract and encode the knowledge and preferences of the security operators.
- Classify and sort the alerts according to the knowledge and the preferences of the security operators.
- Select only the preferred alerts (which satisfy the knowledge and operator preferences).

Qualitative Choice Logic and its extensions

Qualitative Choice Logic (QCL for short) (Brewka, Benferhat, and Le Berre 2004) is an efficient formalism for representing and reasoning with 'basic' preferences. This logic presents however some limitations when dealing with complex preferences that, for instance, involve negated preferences.

QCL is an extension of propositional logic. The non-standard part of QCL logic is a new logical connective $\vec{X}$, called Ordered disjunction, which is fully embedded in the logical language. Intuitively, if A and B are propositional formulas then $A\ \vec{X}\ B$ means: "if possible A, but if A is impossible then at least B". As a consequence, QCL logic can be very useful to represent preferences for that framework. In QCL, when a negation is used on a QCL formula with ordered disjunctions, that negated QCL formula is logically equivalent to a propositional formula obtained by replacing the ordered disjunction ($\vec{X}$) by the propositional disjunction. We proposed in the framework of PLACID project new logics that correctly address QCL's limitations (Benferhat and Sedki 2008b) . These extensions are particularly appropriate for handling prioritized preferences, which are very useful for aggregating preferences of users having different priority levels. We proposed in (Benferhat and Sedki 2008b) a new logic called PQCL (Prioritized QCL) where the negation, conjunction and disjunction departs from the ones used in standard QCL. However, it is based on the same QCL language. This logic is dedicated for handling prioritized preferences and its inference relation correctly deals with negated preferences. In many applications, agent's preferences do not have the same level of importance. For instance, an agent who provides the two preferences : "I prefer Air-France to KLM", and "I prefer a windows seat to a corridor seat, may consider that the first preference statement is more important that the second preference statement. Our logic can manage such prioritized preferences using prioritized conjunction and disjunction.

The second logic we proposed to represent security operators preferences is $QCL+$ (for Positive QCL). This latter is appropriate for handling positive preferences. QCL+ shares many characteristics with PQCL where the semantics of any formula is based on the degree of satisfaction of a formula in a particular model I. Negation in QCL+ is the same as the one of PQCL, hence in QCL+ a double negation of a given formula should recover the original formula.

Let us now show how can our logics help representing a security operator's preferences. He can express his preferences in terms of (i) what he wants to first analyze and (ii) what he would like to ignore. This will be represented with a set of PQCL/QCL+ formulas T. The preference base T contains the set of universally quantified formulas which represents the preferences of the network administrator. For instance, let ϕ be a universally quantified formula ϕ:

$\forall x, \forall y,\ IDS(x, Snort) \wedge Class(x, Probe) \wedge IDS(y, Prelude) \wedge Class(y, DOS) \rightarrow Present\text{-}alert(x) \vec{X} Present\text{-}alert(y)$.

Intuitively, this formula means that if an alert x is provided by the IDS Snort and concerns a Probe attack, and if an alert y is provided by the IDS Prelude and concerns an DOS attack, then the administrator prefers first to analyze the alert x, then the alert y. The following is an example of a preference denoting an alert the administrator wants to ignore:

$\forall x,\ Protocole(x, ICMP) \rightarrow \neg Present\text{-}alert(x)$.

This formula (a sort of integrity constraint) indicates that he wants to ignore the alerts relative to ICMP protocol.

One can have more details on our QCL-based logics for handling security opetators' preferences and their application in the alert correlation field in (Benferhat and Sedki 2008b) (Benferhat and Sedki 2010) (Benferhat and Sedki 2008a).

Probabilistic models for alert correlation

This section presents models based on probabilistic graphical models used for (i) detecting attack plans, (ii) predicting severe attacks and (iii) handling IDSs' reliability. Let us first recall some basic notions on probabilistic graphical models.

Bayesian networks

Bayesian networks are powerful graphical models for modelling and reasoning with uncertain and complex information (Jensen and Nielsen 2007). They are specified by:

i) *A graphical component* consisting in a DAG (Directed Acyclic Graph) allowing an easy representation of the domain knowledge in the form of an influence network (vertices represent events while edges represent *dependence* relations between these events), and

ii) *A probabilistic component* allowing to quantify the uncertainty relative to the relationships between domain vari-

[7] http://clarkparsia.com/pellet

ables using conditional probability tables (CPTs).

Bayesian networks are used for different types of inference such as the maximum a posteriori (MAP), most plausible explanation (MPE), etc. As for applications, they are used as expert systems for diagnosis, simulation, classification, etc. In our system, we use them essentially as classifiers (systems which predict the class label of an item).
Supervised classification consists in predicting the value of a non observable variable given the values of observed variables. Namely, given observed variables A_1,..,A_n describing the objects to classify, it is required to predict the *right* value of the class variable C among a predefined set of class instances. Bayesian network-based classification is a particular kind of probabilistic inference ensured by computing the greatest a posteriori probability of the class variable given the instance to classify. In order to use probabilistic graphical models for analyzing raw IDMEF alerts, these latter are formatted in order to structure them and eliminate the redundant and irrelevant alerts.

Raw IDMEF alerts preprocessing

In order to eliminate the redundant alerts (for instance, alerts with same identifier and targeting the same victim by the same attacker), raw alerts $Alert_1$, $Alert_2$,.., $Alert_k$ generated by IDSs are summarized in alert windows where the occurrences of each alert $Alert_i$ is associated with a variable A_i whose domain is $\{0, 1\}$ where the value 0 means that the alert $Alert_i$ was not observed in the analyzed sequence while value 1 denotes the fact that the alerts $Alert_i$ have been reported. As it is shown in Figure 2, raw alert

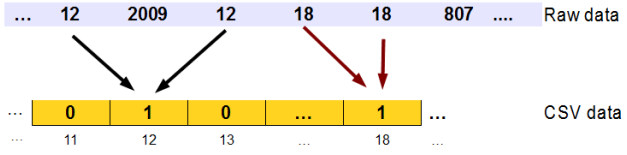


Figure 2: Example of formatting raw alert sequences into presence/absence data

sequences are formatted into CSV[8] data where several occurrences of a given alert is denoted by the value 1. For instance, the alert with identifier 12 has been triggered twice in this example but in the formatted data, we have only once the value 1 corresponding to this alert and denoting that the alert 12 has been observed. As for the alert whose identifier is 11, its value is 0 since it has not been reported in this example. Hence, even if during an alert sequence, an alert $Alert_i$ has been reported several times, this information will be represented with one variable A_i. In the redundant alert elimination step, we eliminate the redundant alerts by substituting several occurrences of the same alert by a single variable value. However, for predicting severe attacks and detecting attack scenarios, there is not need to use every alert. In our application, feature selection can help us to eliminate the irrelevant alerts. We developed a preprocessing tool for IDMEF alerts into CSV data and built a preprocessed and labelled benchmark.

Detecting attack plans using causal networks

In our model for detecting attack plans, this problem is considered as a classification problem. Here, the domain of the class variable contains all the monitored events plus the instance "normal" that corresponds to the absence of all monitored events. We use causal[9] naive Bayes networks to encode the influence of each action A_i on the set of monitored events S by computing conditional probability distributions from the reported alerts. Once probability distributions on different nodes of the naive Bayes network are updated, this model can be used to predict whether an event from E may occur or not, according to a partial or complete observation of S. E will represent the class variable of naive Bayes network and S represent attribute or observation variables. Given a monitored event e_i, we can distinguish three kinds of action a_j:
- Actions with negative influence on e_i which decrease the probability that e_i may occur: $P(e_i|a_j)<P(e_i)$.
- Actions with positive influence on e_i which increase the probability that e_i may occur without exceeding a given threshold: $P(e_i|a_j)>P(e_i)$ and $P(e_i|a_j)<threshold$.
- Actions with critical influence on e_i which indicate that e_i is strongly expected: $P(e_i|a_j)>P(e_i)$ and $P(e_i|a_j)>threshold$.

In (Benferhat, Kenaza, and Mokhtari 2008)(Kenaza, Tabia, and Benferhat 2010), we provide several case studies and experimental evaluations showing high detection rates of attacks scenarios on different benchmarks.

Severe attack prediction as a classification problem

Severe attack[10] prediction consists in analyzing sequences of alerts or audit events in order to predict future severe attacks. In this paper, severe attack prediction is modelled as a classification problem where the variables are defined as follows:

1. **Predictors (attribute variables):** The set of predictors (observed variables) is composed of the set of relevant alerts for predicting the severe attacks. Namely, with each relevant alert variable A_i reports the presence/absence of alert $Alert_i$ in the analyzed sequence.
2. **Class variable:** The class variable C represents the severe attacks variable whose domain involves all the severe attacks $Attack_1$,.., $Attack_n$ to predict and another class instance $NoSevereAttack$ representing alert sequences that are not followed by severe attacks.

Once this model is built on a preprocessed and labelled data, it can analyze in real-time alert sequences to predict severe attacks. Interested readers can find more details and experimental evaluations in (Tabia and Leray 2010b)(Tabia and Leray 2010a).

[8] Comma separated values

[9] A causal network refers to a Bayesian network where the arcs denote cause-effect relationships.

[10] Severe attacks are those associated with high severity levels and representing real danger.

Handling IDSs' reliability

Handling IDSs' reliability allows explicitly taking into account the reliability of the used IDSs. In our application, we rely on Pearl's virtual evidence method (Pearl 1988) which offers a natural way for handling and reasoning with uncertain evidence in the framework of probabilistic networks. In this method, the uncertainty indicates the confidence on the evidence: to what extent the evidence is believed to be true. In our context, if an IDS triggers an alert and we know (from past experience for example) that this event is a false alarm in 95% of the cases then we are in presence of uncertain evidence. In order to apply Pearl's virtual evidence method for efficiently handling IDSs' reliability, we must first assess the IDSs' reliability by means of empirical evaluations (an expert can examine for each alert type triggered by an IDS, the proportion of true/false alerts). An expert can also subjectively (by experience) fix the reliability of the IDSs composing his intrusion detection infrastructure. Now, after assessing the reliability of the IDSs in triggering the alerts A_1,..,A_n, the handling of the uncertainty regarding an alert sequence, proceeds as follows:

1. For each alert A_i, add a child variable R_i as a virtual evidence recasting the uncertainty on A_i. The domain of R_i is D_{R_i}=$\{0, 1\}$ where the value 0 is used to recast the uncertainty regarding the case A_i=0 (alert A_i was not triggered) while 1 is used to take into account the uncertainty in the case A_i=1 (alert A_i was triggered).
2. Each probability distribution $p(R_i/A_i)$ encodes the reliability that the observed values (triggered alerts) are actually true attacks. For example, the probability $p(R_i$=1/A_i=1) denotes the probability that the observation R_i=1 is actually due to a real attack.

When analyzing an alert sequence $r_1 r_2..r_n$ (an instance of observation variables R_1,..,R_n), we compute $argmax_{c_i}(p(c_k/r_1..r_n))$ in order to predict severe attacks.

Controlling prediction/false alarm rate tradeoffs

The objective here is to give the ability to security operators to control the detection/prediction and false alarm rate tradeoffs. Classification with reject option (Chow 1970) is an efficient solution allowing to identify and reject the data objects that will be *probably* misclassified. The reject option is crucial in our application especially for limiting the false alarm rates because the reliability of inputs. This approach allows the user to control the tradeoffs between the severe attach prediction and the underlying false alarm rates. There are two kinds of classification with reject option: (i) **Ambiguity reject:** where the object to classify *belongs* to several classes simultaneously, which makes the classifier confused, and (ii) **Distance reject** which occurs when the instance to classify does not belong to any of the classes represented by the classification model.

Bayesian network-based classifiers are naturally suitable for implementing the classification with reject option as classification is ensured by computing a posteriori probabilities of class instances given the data to be classified. Each probability $P(c_i/a_1..a_n)$ can be interpreted as the confidence of the classifier that the instance to classify belongs to class instance c_i. In our model, we are interested in controlling the attack prediction/false alarm rate tradeoffs according to the contexts and needs of each final user. For example, a user may want an alert correlation tool with high confidence (with minimum false alerts). Let us define the *confidence* concept in our application as the unsigned value of the difference of the probability that the instance to classify $a_1 a_2..a_n$ is not a severe attack and the probability that $a_1 a_2..a_n$ is actually a severe attack. It is done by measuring the gap between the probability that the alert sequence will not be followed by a severe attack (namely $p(c_i = 0/a_1..a_n)$) and the greatest probability that the event will be followed by a severe attack. Namely,

$$\varphi(a_1..a_n) = |p(c_i = 0/a_1..a_n) - \max_{c_i \neq 0} p(c_i/a_1..a_n)|, \quad (1)$$

where c_i=0 denotes the class instance representing alert sequences that are not followed by severe attacks while class instance $c_i \neq 0$ denote class instances associated with the severe attacks to predict. The value of $\varphi(a_1 a_2..a_n)$ gives an estimate of the classifier's confidence that the analyzed alert sequence will/will not be followed by a severe attack. Then, the Bayesian decision rule is reformulated accordingly to implement the reject option (Tabia and Leray 2010b).

Regarding the evaluation of our probabilistic graphical models, several case studies and experimental evaluations are carried out on real and representative datasets collected in the framework of the PLACID project (Tabia and Leray 2010b)(Tabia and Leray 2010a)(Kenaza, Tabia, and Benferhat 2010). The obtained results are very promissing and we argue that such formalisms combined with logic-based ones are an efficient approach to address most of the issues related to the alert correlation problem.

Discussion and concluding remarks

The key contribution described in this paper is the design of a complete alert correlation platform combining security operators' knowledge and preferences with probabilistic graphical formalisms. In order to achieve our objectives, new non-classical logics for t representing complex information and preferences are developed. In particular, the application to alert correlation has identified a class of algorithms for effectively reasoning about alerts. Regarding description logics, the proposals for the handling of inconsistances, based for instance on the lexicographic inference relation, are original and interesting. This issue is very important to solve inconsistancy problems in the security operators' knowledge and preferences. Eliciting and agregating the operators' preferences is also an important issue to address. The knowledge base (TBOX and ABOX) obtained from encoding IDMEF alerts and built in the framework of PLACID project will serve as benchmarks for future works on reasoning in description logics. Regarding the reasoning under uncertainty and classification field, the project first developed a new method of modelling a classification problem over event sequences. This method can be reused in other application areas such as the surveillance of abnormal events (in airports or in the health sector) from observations supplied by sensors. The PLACID project has also proposed

methods for the difficult problem of classification under uncertain observations. A future work direction is to integrate, valid and issue a first version of our alert correlation platform. A second interesting issue is to apply our contributions for similar problems in other fields such as the one of managing alerts issued by sensors monitoring old persons or patients from assisted living houses or hospitals. In this application, doctors' knowledge and medical ontologies can be used to build the domain knowledge and preferences of operators can be represented. The probabilistic models can be used to assess the plausibility of some events or for generating hypotheses, etc.

Acknowledgements

This work is supported by the (ANR) SETIN 2006 PLACID project `http://placid.insa-rouen.fr/`). We thank the ANR agency and all the co-authors of works described in this paper.

References

Al-Mamory, S. O., and Zhang, H. 2009. IDS alerts correlation using grammar-based approach. *Journal in Computer Virology* 5(4):271–282.

Axelsson, S. 2000. Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Chalmers Univ.

Benferhat, S., and Sedki, K. 2008a. Alert correlation based on a logical handling of administrator preferences and knowledge. In *SECRYPT 2008, Proceedings of the International Conference on Security and Cryptography, Porto, Portugal, July 26-29*, 50–56.

Benferhat, S., and Sedki, K. 2008b. Two alternatives for handling preferences in qualitative choice logic. *Fuzzy Sets and Systems* 159(15):1889 – 1912.

Benferhat, S., and Sedki, K. 2010. An alert correlation approach based on security operator's knowledge and preferences. *Journal of Applied Non-Classical Logics* 20(1-2):7–37.

Benferhat, S.; Kenaza, T.; and Mokhtari, A. 2008. A naive bayes approach for detecting coordinated attacks. In *COMPSAC*, 704–709.

Bin, Z., and Ghorbani, A. 2006. Alert correlation for extracting attack strategies. *I. J. Network Security* 3(3):244–258.

Brewka, G.; Benferhat, S.; and Le Berre, D. 2004. Qualitative choice logic. *Artif. Intell.* 157:203–237.

Chow, C. 1970. On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory* 16(1):41–46.

Cuppens, F., and Miège, A. 2002. Alert correlation in a cooperative intrusion detection framework. In *IEEE Symposium on Security and Privacy*, 187–200.

Dain, O., and Cunningham, R. K. 2001. Fusing a heterogeneous alert stream into scenarios. In *In Proceedings of the 2001 ACM workshop on Data Mining for Security Applications*, 1–13.

Debar, H., and Wespi, A. 2001. Aggregation and correlation of intrusion-detection alerts. In *Recent Advances in Intrusion Detection*, 85–103. London, UK: Springer.

Friedman, N.; Geiger, D.; Goldszmidt, M.; Provan, G.; Langley, P.; and Smyth, P. 1997. Bayesian network classifiers. *Machine Learning* 131–163.

Goldman, R., and Harp, S. A. 2009. Model-based intrusion assessment in common lisp. In *International Lisp Conference*.

Goldman, R. P.; Heimerdinger, W.; Harp, S. A.; Geib, C. W.; Thomas, V.; and Carter, R. L. 2001. Information modeling for intrusion report aggregation. In *in Proceedings of the DARPA Information Survivability Conference and Exposition II (DISCEX-II*, 329–342.

Ilgun, K.; Kemmerer, R. A.; and Porras, P. A. 1995. State transition analysis: A rule-based intrusion detection approach. *IEEE Trans. Softw. Eng.* 21:181–199.

Jensen, F. V., and Nielsen, T. D. 2007. *Bayesian Networks and Decision Graphs (Information Science and Statistics)*. Springer.

Julisch, K., and Dacier, M. 2002. Mining intrusion detection alarms for actionable knowledge. In *Eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 366–375. New York, NY, USA: ACM.

Kenaza, T.; Tabia, K.; and Benferhat, S. 2010. On the use of naive bayesian classifiers for detecting elementary and coordinated attacks. *Fundam. Inform.* 105(4):435–466.

Li, W., and Tian, S. 2010. An ontology-based intrusion alerts correlation system. *Expert Systems with Applications* 37(10):7138 – 7146.

Lingyu, W.; Anyi, L.; and Sushil, J. 2006. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. *Comput. Commun.* 29(15):2917–2933.

Mé, L.; Debar, H.; Morin, B.; and Ducassé, M. 2008. M4D4: a Logical Framework to Support Alert Correlation in Intrusion Detection. *Information Fusion* –.

Ning, P.; Cui, Y.; and Reeves, D. S. 2002. Constructing attack scenarios through correlation of intrusion alerts. In *9th ACM conference on Computer and communications security*, 245–254. NY, USA: ACM.

Patcha, A., and Park, J. 2007. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks* 51(12):3448–3470.

Pearl, J. 1988. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Roesch, M. 1999. Snort - lightweight intrusion detection for networks. 229–238.

Smith, R.; Japkowicz, N.; Dondo, M.; and Mason, P. 2008. Using unsupervised learning for network alert correlation. In *21st conference on Advances in artificial intelligence*, 308–319. Berlin, Heidelberg: Springer-Verlag.

Tabia, K., and Leray, P. 2010a. Bayesian network-based approaches for severe attack prediction and handling IDSs' reliability. In *Information Processing and Management of Uncertainty in Knowledge-Based Systems, IPMU 2010, Dortmund, Germany, June 28 - July 2,*, 632–642.

Tabia, K., and Leray, P. 2010b. Handling IDS' reliability in alert correlation - a bayesian network-based model for handling ids's reliability and controlling prediction/false alarm rate tradeoffs. In *Proceedings of the International Conference on Security and Cryptography, Athens, Greece, July 26-28*, 14–24.

Tjhai, G. C.; Papadaki, M.; Furnell, S.; and Clarke, N. L. 2008. Investigating the problem of IDS false alarms: An experimental study using snort. In *23rd International Information Security Conference SEC 2008*, 253–267.

Valdes, A., and Skinner, K. 2001. Probabilistic alert correlation. In *Recent Advances in Intrusion Detection*, 54–68. London, UK: Springer-Verlag.

Yahi, S.; Benferhat, S.; and Kenaza, T. 2010. Conflicts handling in cooperative intrusion detection: A description logic approach. In *22nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2010, Arras, France, 27-29 October*, 360–362.

Personalized Vulnerability Analysis through Automated Planning

Mark Roberts, Adele Howe,
Indrajit Ray, Malgorzata Urbanska
Computer Science Dept.,
Colorado State University,
Fort Collins, CO 80524 USA
{mroberts, howe, indrajit, urbanska}@cs.colostate.edu

Zinta S. Byrne
Janet M. Weidert
Psychology Dept.,
Colorado State University,
Fort Collins, CO 80524 USA
{Zinta.Byrne, Janet.Weidert}@colostate.edu

Abstract

Attack trees and attack graphs are a well-known representation for computer security vulnerabilities. They capture how malicious activity can result in compromised systems. Unfortunately, attack graphs scale poorly, are targeted primarily to capture key components of vulnerabilities in *industrial* settings and focus on attacker actions. We extend the attack graph formalism by integrating user actions and supporting personalization that restricts the focus to those vulnerabilities present in a particular user/system combination (i.e., considering specific computer configuration and a user's normal activities). The enhanced attack graph is captured in the Planner Domain Definition Language (PDDL) with user specific attributes being included as facts. Given the PDDL representation, we can use a planner to 1) prune the attack graph to only those vulnerabilities for which we can derive a plan, 2) perform what-if analyses of user actions and 3) help identify interventions that are most likely to reduce the system's susceptibility to attacks.

Personalizing Security

Computer security models usually focus on identifying and preventing network breaches from malicious actors. Attack trees (Schneier 1999) and attack graphs (Sheyner et al. 2002) capture how malicious activity can result in compromised systems or conversely how computer systems might be vulnerable to intentionally destructive actions. The systems are in workplaces, and the non-malicious users are largely disregarded. Yet studies (CERT 2008; Sasse, Brostoff, and Weirich 2001) suggest that the user, rather than an active adversary, may well pose the largest security risk.

In this paper, we discuss some initial steps in "personalizing" security vulnerability analysis. In particular, we augment the canonical form of the attack graph to admit user actions and associated system states, thus personalizing the notion of attack to encompass user culpability. Additionally, we exploit the enhanced representation to help define attack graphs that can be automatically tailored to specific users, their behaviors, and their specific computer systems. The resulting attack graphs need only encompass vulnerabilities that are likely for a particular user/personal computer combination. The personalized attack graphs can be analyzed to determine what system states to monitor and how to intervene – either preventing vulnerabilities or taking corrective actions once a breach has occurred.

We highlight two systematic ways to model a networked system's risk to malicious attacks: attack trees (Dawkins, Campbell, and Hale 2002; Dewri et al. 2007; Schneier 1999; Tidwell et al. 2001) and attack graphs (Ammann, Wijesekera, and Kaushik 2002; Artz 2002; Cuppens and Ortalo 2000; Dawkins and Hale 2004; Jajodia, Noel, and O'Berry 2003; Jha, Sheyner, and Wing 2002; Ritchey and Ammann 2000; Sheyner et al. 2002; Swiler et al. 2001; Templeton and Levitt 2000). They help to organize and analyze intrusion and/or misuse scenarios (also called attack scenarios) by enumerating known vulnerabilities or weak spots in the system, and capturing the cause-consequence relationships between system configuration and these vulnerabilities in the form of an And-Or tree (attack trees) or a directed graph (attack graph).

For every system resource (such as, a web server, account information on a host, or a piece of confidential information) that needs to be protected, an attack tree/graph is defined. In their simplest forms, the nodes in the tree/graph represent certain system states of interest to an attacker, with edges connecting them to indicate the cause-consequence relationship among the states. The root node in the case of attack trees or a final node in the case of attack graphs, represents the attacker's ultimate goal, namely, causing damage to the protected resource. The interior nodes represent possible system states during the execution of the attack (attacker sub-goals). The leaf nodes or one or more source nodes represent system states that must exist for an attacker to successfully initiate the attack. System states can include different levels of partial compromise effectuated by the attacker (e.g., access to the password file), configuration information (e.g., operating system and hardware configuration, network connectivity etc.), state changes achieved on specific system components (e.g., implanting a keystroke logger), or any other subgoal that will ultimately lead to the final goal. Edges represent actions taken by the attacker causing a progression of the attack.

Researchers have also proposed different variants of attack trees and attack graphs to capture more information about system risks. Examples of these are the multiple-prerequisite graph (Lippman et al. 2006) and the exploit-

dependency graph (Noel et al. 2003) that includes the different conjunctive and disjunctive relationships between various nodes in the graph.

Although different attack scenarios are easily perceived in attack graphs, attack graphs can potentially suffer from a state space explosion problem. Ammann et al. (2002) proposed a refined formulation of attack graphs with the assumption of monotonicity. The monotonicity property supposes that if a state is reached as a result of some attack, it remains so during subsequent attacks. Such an assumption can greatly reduce the number of nodes in the attack graph. Other approaches to address scalability include aggregating a set of nodes representing similar host configurations into single nodes (Noel and Jajodia 2004; Swiler et al. 2001), building restricted attack graphs simply for answering very specific queries (Sheyner et al. 2002), building reduced attack trees that are minimal spanning trees in graphs (Birkholz et al. 2010), and reducing attack graphs using planning techniques (Ghosh and Ghosh 2010).

Almost all of these approaches to attack trees and attack graphs are constructed to capture vulnerabilities in enterprise/industrial settings and focus on attacker actions. None of them integrate user actions and support personalization that restricts the focus to those vulnerabilities present for a particular user/system combination. Attack graphs are designed to support network analysis. Thus, they do not include the level of detail – or dynamism in the representation – required to direct active monitoring for attacks and determine consequences of interventions. In contrast, automated planning has been used to help simulate and analyze security breaches in networked computer systems. Boddy et al. (2005) built a mixed initiative planning system, called the Behavioral Adversary Modeling System (BAMS), that could identify potential vulnerabilities and countermeasures in cyber security for large organizations. The authors wrote a domain model in the Planner Domain Definition Language (PDDL), which is a common input language for many recent planners (Fox and Long 2003). They were able to produce "insider subversion" plans of 40–60 steps. A key contribution of the work was showing that automated planning could highlight novel (sometimes unexpected) attack scenarios for the analyst driving BAMS. The BAMS PDDL domain was later released as a suite of problems for the International Planning Competition.

Ghosh and Ghosh (2010) proved they could reduce the computational complexity of an attack graph by iteratively applying a planner to eliminate from further consideration possible attack scenarios. They use a PDDL model that is similar to that of Boddy et al. and generate minimal attack paths. Their approach allows the model to scale to realistic, complex networks. A similar planner/model setup as that of Ghosh and Ghosh was integrated into a penetration testing framework (Obes, Sarraute, and Richarte 2010). Their model contained 1800 actions and was extensively tested against medium-sized networks.

Our work complements the previous approaches by leveraging these PDDL domains with a detail-oriented, user-centered extension to the formal model of the attack tree/graph. The representation is intended to form the core

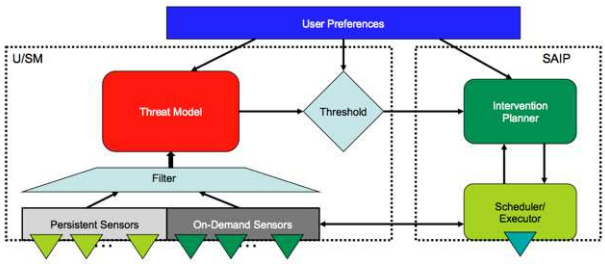


Figure 1: The proposed Mixed Initiative Planning and Scheduling Agent (MIPSA) Architecture.

of an agent that will help home computer users maintain security on their machines by monitoring some of their actions and recommending actions to reduce security risks. The agent will be developed from studies of users. In this paper, we describe the project in brief and present a proof of concept of the enhanced attack graph and its representation in PDDL.

Psychologically Motivated Computer Security

Home users are often considered the weakest link in computer security. In our project, we combine expertise from psychology, computer security and artificial intelligence to study the nature of the risks inherent in normal activity on the Internet, the perception of those risks, the judgment about trade-offs in behavior and the design of a personalized agent that can alert users to risky behavior and help to protect them. The key insight is that security and privacy protection requires the efforts of *both* the computer and the user; thus, the security measures must work in concert with the user's activities and needs.

The goal of our project is to develop a semi-autonomous, intelligent and personalized approach to computer security that leverages psychological studies of what users want/need, what security and privacy risks are imminent based on the status of the system and the user's actions and what interventions will be most effective. The intelligent agent will act as an interface between the system, the security software and the user. As shown in Figure 1, we anticipate having two conceptual components: a user/system monitor (U/SM) and a semi-autonomous action planner (SAIP). The actions will include coaching the user about the consequences of their choices, administrating the security software and identifying system modifications that mitigate threats.

Thus far, we have completed policy-capturing studies (Byrne et al. 2011) of subject responses to security vignettes of four Internet security threats: *availability*, *integrity*, *confidentiality*, and *unwitting accomplice*. Availability refers to when a user's computer resources are improperly made inaccessible (e.g., a program deletes a user's browsing history or other documents). *Integrity* refers to when information on the user's computer is improperly modified by unauthorized entities (e.g., a program adds sites to a user's browsing history). *Confidentiality* threats occur when sensitive information is exposed to those who do not have rightful access

(e.g., a program tracks a user's browsing history or obtains access to private and personal information). Finally, a user can become an *unwitting accomplice* when a breach of security results in the user unknowingly spreading the threat to other users (e.g., email contact information is used to forward the attack).

Policy-capturing is a method for understanding what information people use in decision-making (Zedeck 1977). People are asked to read a series of vignettes, wherein cues thought to influence decision choices are systematically varied. Based on responses, researchers identify which cues are most influential in decision-making. The subjects were composed of two especially vulnerable groups of computer users (Fox and Long 2003): college students and adults over 50 years of age. The first group knows enough to be savvy about computer applications, but not necessarily enough to understand the consequences of their actions on their computers. The second group tends to lack computer knowledge that leads to inadvertent security breaches because they do not understand what they are doing. Results using hierarchical linear modeling showed that age, self-rated levels of computer knowledge, both basic and extensive levels of computer knowledge, frequency of computer use, and having one's privacy previously invaded factor into, in varying degrees, predicting perceptions of risk and vulnerability to each of the four threats. Additionally, women were more likely than men to click again on an embedded link across all scenarios.

In our next study, we will be simulating interactions with a mailer and browser to gauge the perceived value of a variety of risky behaviors, the user's actual actions when placed in risky situations and the user's perceptions of risk and the consequences of their actions. This study should provide the first quantitative user values needed for our personalized security model such as cost/benefit of action and probabilities. The remainder of this paper describes the first steps toward building personalized security models that can incorporate such information, as well as information about specific computer system configurations and user activity profiles, and can be used to reason about what vulnerabilities are likely to occur and how to prevent or mitigate them.

Personalized Attack Graph

At the heart of our MIPSA architecture is a threat model that will be used to monitor system and user activity, identify when a threat is imminent and help determine appropriate action to neutralize the threat. Attack graphs provide an excellent starting point for such a model. We propose to "personalize" attack graphs by explicitly representing user, attacker and potentially system (ultimately agent) actions, tailoring the representation to specific users through pruning and analyzing the graphs to design the security interventions to best match the needs of specific users.

Figure 2 shows an example of an instantiated attack graph with three possible attack scenarios resulting in a compromised system. In our discussion that follows we will make reference to the attack graph, though it is occasionally clear from context that the graph under discussion is an And-Or attack tree. We call this an graph *instantiated* because this graph would not actually exist in a graphical form as shown in the figure, but rather the actual set of nodes that get instantiated will depend on the system/user state. Our enhanced graph includes not only possible states of the system but also possible user actions that can result in information leakage. User actions are in color. The first scenario uses the *VB-Script MsgBox()* vulnerability (NIST 2010) which exists in the OS Windows 2000 SP4 and Windows XP SP2 operating systems. The attack can be launched in two steps: 1) the key-logger is installed in the help file and 2) it is activated by the user by pressing the F1 key. The second scenario describes the *browser SSL certificate* vulnerability (NIST 2009), which exists in some versions of the Firefox browser prior to version 3.0.13, the SeaMonkey browser prior to version 1.1.18 and the Mozilla Network Security Services prior to version 3.12.3. The vulnerability allows a SSL spoofing attack by improperly handling a '\0' character in a domain name in the subject's Common Name (CN) field of an X.509 certificate. This permits the attacker to masquerade as an arbitrary SSL server and thereby cause information leakage. The last illustrates a phishing email attack that exploits risky user behavior. This graph will be analyzed using a Planner to determine which portions are sufficient for protecting a particular user.

PDDL Model of User and Attacker Actions

The attack graph is an excellent visual representation, but it does not enumerate the details needed to either analyze what can happen or to monitor events. We translate the attack graph into PDDL, which requires all of the salient attributes of system/user state. In our case, the domain corresponds to an attack graph enhanced with actions and states possible for a range of users. Problem descriptions allow us to refine the model to *specific* user/computer combinations. There were two existing domains we could leverage in constructing our domain.

The *BAMS* domain presented in Boddy et al. (2005) is automatically constructed from macros after an analyst sets up a problem description using a GUI. The macros enabled the domain to adjust to new objects or actions based on what the analyst entered. The final domain contains over 25 types that capture details concerning computing objects (hosts, network configuration, security tokens, etc.) and physical objects (rooms, people, doors, keys, etc.). The domain has 124 predicates that indicate particular physical state such as `(can-unlock key1 lock1)` as well as security concerns such as `(knows bob bob-password)`. The authors point out that actions involving a large number (5-10) of parameters and preconditions bogged down the planner. To overcome this problem, the authors split large actions into chains of smaller actions.

Ghosh and Ghosh (2010) presented an *AttackGraph* domain that was similar to that in Boddy et al. (2005). This domain encodes network configurations that describe the types of services running on hosts, vulnerability instances that describe the types of vulnerabilities present in the system, functions that describe who has access to what resources, and exploit descriptions that describe the intruder and network pre/post conditions. The domain examples and solu-

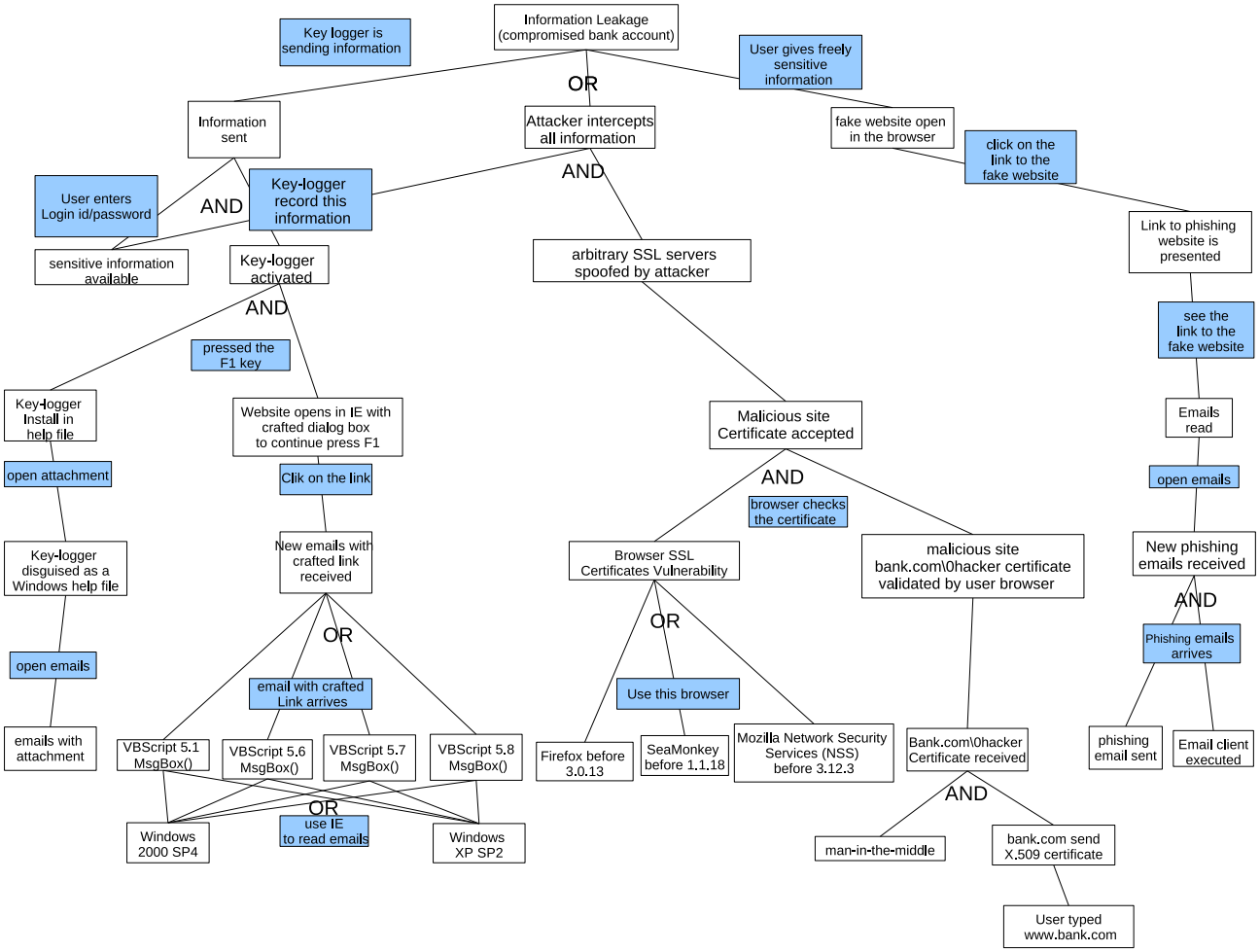


Figure 2: An instantiated attack graph augmented with user actions. This example graph shows three attack scenarios for how a user's bank account credentials can become compromised. The leftmost subgraph is a keylogger attack, the middle subgraph is a SSL spoof attack, and the right subgraph is a phishing attack.

tions that the authors present appear to be relatively small in comparison to the BAMS domain.

Our PDDL model follows a similar composition to the two previous works. Facts encode information such as installed software, system state and permissions. As in the previous research, actions encompass those of attackers and software changes to system state; in our case, we also add actions of the user and software that may contribute to possible security breaches. A plan constructed from such a model has been used to show the sequence of actions that result in an attacker causing a networked vulnerability (Boddy et al.) or a path that cannot be pruned from the attack graph (Ghosh et al.). In our case, we leverage the plan for both of these purposes as well as to monitor a personal computer's security status and steer a user's actions away from risky behavior.

The Personalized AttackGraph Domain

To demonstrate how the planner can be used to refine a model for MIPSA, we constructed by hand a PDDL model from the augmented attack graph in Figure 2. Figure 3 shows the domain description. It encodes all three possible attack paths – key-logger, ssl, phishing – as potential means by which a planner can achieve the goal predicate *(information-leakage ?Account)*. To distinguish between the three possible goal paths, we include a predicate *(exploit-vulnerability ?V)*, where *?V* can be any of {key-logger, ssl, or phishing}.

An important detail of this model is that it integrates states from the attack graph with user and attacker actions. Vulnerabilities can exploit specific versions of software or system state but rely upon user/attacker actions to fully instantiate. For example, the model contains state for the type and version of the operating system as well as the type and version of the Internet browser. User and attacker behaviors are tracked through the application of actions, which serve as transition functions between states.

Our model employs four action types. **User actions** track what the user is required to do to enable the vulnerability; these include reading email or visiting a webpage. **Attacker actions** represent steps the attacker must do to exploit a vulnerability, e.g., intercepting a certificate, presenting a

```
(define (domain attackgraph)
(:requirements :strips :equality :disjunctive-preconditions)

(:predicates (user ?user) (account ?account) (browser ?browser
 ?version) (information-leakage ?account) (login ?user ?account)
 (information-available ?user ?account) (installed ?key-logger)
 (activated ?key-logger) (mail-attachment ?file) (site ?Site)
 (open-attachment ?file) (key-logger-trojan ?file ?key-logger)
 (records ?KeyLogger ?Account) (vb-script-version ?script) (OS ?type
 ?version) (mailer ?email-program) (use-mailer ?email-program)
 (use-browser ?browser-program) (user-hit-F1-in-IE) (user-visits ?user
 ?site) (receive-email ?User ?email) (read-email ?email) (clicked-link
 ?email) (exploit-vulnerability ?vulnerability-type)
 (certificate-accepted ?site ?certificate) (certificate-authorized
 ?certificate) (browser-ssl-compromised ?browser) (ssl-server-spoofed
 ?domain) (make-safe-vulnerability ?vulerability) (has-trojan ?File))

(:action user-read-email :parameters (?User ?Mailer)
 :precondition (and (user ?User) (mailer ?Mailer))
 :effect (use-mailer ?Mailer))

(:action user-open-attachment :parameters (?User ?File ?Mailer)
 :precondition (and (user ?User) (use-mailer ?Mailer)
                    (mail-attachment ?File))
 :effect (open-attachment ?File))

(:action key-logger-installed :parameters (?User ?File ?KeyLogger)
 :precondition (and (user ?User) (open-attachment ?File)
    (has-trojan ?File) (key-logger-trojan ?File ?KeyLogger))
 :effect (installed ?KeyLogger))

(:action user-visits-site-keylogger-installed
 :parameters (?User ?Browser ?Site ?KeyLogger)
 :precondition (and (user ?User) (installed ?KeyLogger)
   (browser ?Browser any-version) (site ?Site)
   (or (vb-script-version VB-5-1) (vb-script-version VB-5-6)
       (vb-script-version VB-5-7) (vb-script-version VB-5-8)))
 :effect (and (use-browser ?Browser) (user-visits ?User ?Site)))

(:action user-presses-F1  :parameters (?User ?Browser ?Site)
 :precondition (and (user ?User) (use-browser ?Browser)
   (= ?Browser browser-IE) (= ?Site compromised-site)
   (user-visits ?User compromised-site))
 :effect (user-hit-F1-in-IE))

(:action user-visits-site  :parameters (?User ?Browser ?Site)
 :precondition (and (user ?User) (site ?Site))
 :effect (and (use-browser ?Browser) (user-visits ?User ?Site)))

(:action key-logger-activated :parameters (?KeyLogger)
 :precondition (and (installed ?KeyLogger) (user-hit-F1-in-IE))
 :effect (activated ?KeyLogger))

(:action attacker-intercepts  :parameters (?KeyLogger ?Account)
 :precondition (and (records ?KeyLogger ?Account)
       (exploit-vulnerability vulnerability-key-logger))
 :effect (information-leakage ?Account))

(:action user-login :parameters (?User ?Account)
 :precondition (and  (user ?User) (account ?Account)
                (information-available ?User ?Account))
 :effect (login ?User ?Account))

(:action user-login-with-keylogger-activated
 :parameters (?User ?Account ?KeyLogger)
 :precondition (and  (user ?User) (account ?Account)
                (activated ?KeyLogger)
                (information-available ?User ?Account))
 :effect (and (login ?User ?Account)  (records ?KeyLogger ?Account)))

(:action attacker-sends-phishing-email
 :parameters (?User ?Email)
 :precondition (and (user ?User))
 :effect (receive-email ?User ?Email))

(:action user-reads-phishing-email
 :parameters (?User ?Program ?Email)
 :precondition (and (user ?User) (use-mailer ?Program)
  (= ?Email phishing-email) (receive-email ?User ?Email))
 :effect (read-email phishing-email))

(:action user-clicks-link-in-phishing-email :parameters
   (?User ?Email)  :precondition (and (user ?User)
   (read-email ?Email)) :effect (clicked-link ?Email))

(:action phishing-site-opens :parameters (?User ?Email ?Site)
 :precondition (and (user ?User) (clicked-link ?Email)
    (= ?Site phishing-site)) :effect (user-visits ?User ?Site))

(:action user-enters-information-in-phishing-site
 :parameters (?User ?Account ?Site)
 :precondition (and (user ?User) (account ?Account)
   (read-email phishing-email) (user-visits ?User ?Site)
   (exploit-vulnerability vulnerability-phishing)
   (= ?Site phishing-site))
 :effect (information-leakage ?Account))

(:action browser-contains-ssl-vulnerability
 :parameters (?Browser ?Version)
 :precondition (or (and (browser ?Browser ?Version)
        (= ?Browser browser-firefox) (= ?Version v3-0-13))
  (and (browser ?Browser ?Version)
       (= ?Browser browser-seamonkey)(= ?Version v1-1-18))
  (and (browser ?Browser ?Version)
       (= ?Browser browser-mozilla) (= ?Version v3-12-3)))
 :effect (browser-ssl-compromised ?Browser))

(:action attacker-obtains-certificate :parameters (?Certificate)
 :precondition (and (= ?Certificate malicious-certificate))
 :effect (certificate-authorized ?Certificate))

(:action browser-accepts-certificate
 :parameters (?User ?Browser ?Site ?Certificate)
 :precondition (and (user ?User) (user-visits ?User ?Site)
               (browser-ssl-compromised ?Browser)
               (certificate-authorized ?Certificate))
 :effect (certificate-accepted ?Site ?Certificate))

(:action attacker-spoofs-ssl-server
 :parameters (?User ?Browser ?Certificate ?Site)
 :precondition (and (user-visits ?User ?Site)
    (certificate-accepted ?Site ?Certificate)
    (browser-ssl-compromised ?Browser)
    (= ?Certificate malicious-certificate))
 :effect (ssl-server-spoofed ?Site))

(:action user-login-under-spoof :parameters (?User ?Account ?Site)
 :precondition (and (user ?User) (account ?Account)
   (user-visits ?User ?Site) (ssl-server-spoofed ?Site)
   (exploit-vulnerability vulnerability-ssl))
 :effect (information-leakage ?Account))

(:action intervene-clean-all-attachments :parameters (?File)
 :precondition (and (exploit-vulnerability vulnerability-key-logger)
   (mail-attachment ?File) (has-trojan ?File))
 :effect (not (has-trojan ?File))))
```

Figure 3: PDDL problem description of a user being compromised by a SSL-server spoof

compromised certificate, or intercepting compromised information. **System actions** enable the vulnerability through programmatic means, e.g., a browser accepting a (compromised) certificate or a key-logger capturing and forwarding keystrokes. Finally, **intervention actions** disrupt either system state or other actions that would otherwise lead to a compromised system, e.g., scanning an attachment for viruses or validating a certificate checksum.

The AttackGraph Domain Problems

Each PDDL problem description includes a set of world objects, an initial condition, and a goal description. We generated a number of problems to test the domain. For our tests, we ran each problem with FF 2.3 (Hoffmann and Nebel 2001), due to its simplicity and speed. The models we present did not rely on metric evaluation, fluents, or action costs, although adding them is a future goal.

Figure 4 shows the problem description that leads to the ssl vulnerability. Note that the initial state includes the required predicates that would activate the vulnerability; note also that we have set *(exploit-vulnerability ssl)*.

```
(define (problem user-compromised-ssl)}
(:domain attackgraph)
(:objects user1  browser-IE browser-firefox
     ... phishing-email ... phishing-website
  vulnerability-ssl  key-lggr1)
(:init (site bank-site)(account account-bank)
       (user user1)(browser browser-firefox v3-0-13)
       (OS WINDOWS 2000SP4)(vb-script-version VB-5-1)
       (mail-attachment file-with-trojan)
       (mail-attachment file-without-trojan)
       (key-logger-trojan file-with-trojan key-lggr1)
       (information-available user1 account-bank)
       (exploit-vulnerability vulnerability-ssl))
(:goal (and (information-leakage account-bank))))


step 0: ATTACKER-OBTAINS-CERTIFICATE
          MALICIOUS-CERTIFICATE
     1: BROWSER-CONTAINS-SSL-VULNERABILITY
          BROWSER-FIREFOX V3-0-13
     2: USER-VISITS-SITE USER1 USER1 BANK-WEBSITE
     3: BROWSER-ACCEPTS-CERTIFICATE USER1
          BANK-WEBSITE MALICIOUS-CERTIFICATE
     4: ATTACKER-SPOOFS-SSL-SERVER USER1
          BROWSER-FIREFOX MALICIOUS-CERTIFICATE
          BANK-WEBSITE
     5: USER-LOGIN-UNDER-SPOOF USER1 ACCOUNT-BANK
          BANK-WEBSITE
```

Figure 4: PDDL problem description (top) of a user being compromised by a SSL-server spoof with the resulting plan from FF (bottom)

How a Planner Helps Personalize the Attack Graph

We can demonstrate several benefits of using automated planning in conjunction with the attack graph. Specifically, we show that we can generate viable attack paths, perform what-if analysis using the planner to search and validate scenarios, derive and include possible interventions to prevent attack, and motivate how the planner can personalize the attack graph for specific users/systems.

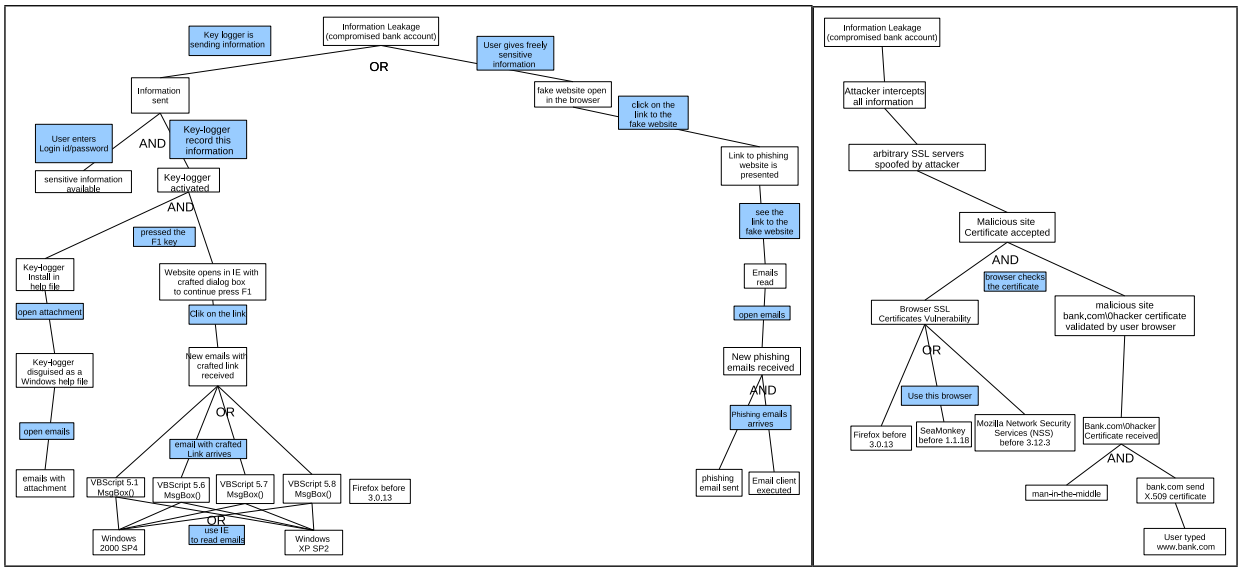
The PDDL model can isolate paths that lead to vulnerabilities. Figure 4 (bottom) shows the plan found by FF for the ssl problem. Planning monitoring and interventions is a critical component of securing any system. Too much monitoring is potentially intrusive, annoying to the user and detrimental to system performance. Interventions can take place in the system level or through user interaction. For key-logger, interventions can include (among others) telling the user to upgrade their browser, scanning the attachment to remove the key-logger, or advising the user not to open the attachment. The last action in the domain description is an example of an intervention. When we replaced (view-attachment . . .) with this new action, FF could not find a plan that satisfied *(information-leakage ?Account)*. Our example is a hand-picked intervention; we discuss in future work how we intend to automate this process through details about the domain.

We show a simple example of how the planner can aid in performing what-if analysis by changing the browser in the initial conditions for the problem in Figure 4. In this case, FF cannot satisfy the leakage goal because the key-logger vulnerability requires a specific action in a compromised browser based on Mozilla. Performing what-if analysis such as this could lead to an intervention informing the user to use a different browser on the current system. Similarly, we might be able to determine that intervention is unnecessary if the user is already using the IE browser, but that intervention becomes critical if a compromised browser were suddenly opened (by the user or otherwise).

We demonstrate how the planner is able to personalize and prune the attack graph in two motivating use case examples. While we recognize that specific individuals will vary in their computer knowledge, we consider two hypothetical users who match the target age groups for the psychological study. UserA is a retired person who was recently given a Windows XP machine that runs Internet explorer (IE). UserA is familiar with the inventory computer system from a recent job but is a new user of the Internet and email. UserB is a 20-year-old college student with a portable laptop running Windows7. UserB has used computers since kindergarten, is very confident when using them, and insists on browsing the Internet with Firefox.

MIPSAs running on the machines of both users could use the *same* attack graph (see Figure 2), which allows the agents to leverage centralized knowledge about existing vulnerabilities. But using the same graph may also cause the MIPSA to consider possibilities irrelevant to the current system, user profile, or user's task. For example, the MIPSA for UserA need not consider any portion of the graph related to the Firefox/Netscape vulnerability because UserA does not use Firefox. So this subgraph can be pruned (see Figure 5a). Similarly, the MIPSA running on UserB's laptop can prune the IE-related attacks; the phishing subgraph is also pruned since UserB is unlikely to believe the spurious phishing email (see Figure 5b). The pruned graphs contain information relevant to the somewhat static user profile. Dynamic changes could impact the graph if, for example, the user is performing a task like playing solitaire or word processing. In such a case, the graphs for both users would be completely pruned since this example attack graph does not contain vulnerabilities specific to such activities.

Automated planning can be used to prune impossible or irrelevant portions of the attack graph from consideration of the MIPSA. To accomplish this pruning, we hand-wrote a domain model in PDDL. Then we provided goals for each of the subgraph under consideration; in this example, our subgraph are labeled *key-logger*, *ssl*, *phishing*. If the planner is able to find a path from the initial conditions then this vulnerability is considered viable. In the absence of any solution (and assuming the domain model is correct), the



(a) user A (b) user B

Figure 5: The resulting attack graphs for two hypothetical users after applying their profiles (see text for details).

planner has identified that the *current* system/user configuration is safe. Thus, we can eliminate the entire subgraph for these three paths. Because we are adding more information to the attack graph to support user personalization and system monitoring, it is critical that we prune the graph to support scalability.

Future Work

This work is a first step towards incorporating the user's actions into the security risk assessment and mitigation problem. The intervention plan suggested by the planner clearly shows how a user could be protected in cyber space. However, as noted throughout this paper, a significant portion – namely, the generation of the attack graph, and the translation of the attack graph into the PDDL model – was done manually. This is not only labor intensive but prone to errors. The next step, thus, is to automate this process. Several researchers have addressed the problem of generating attack graphs automatically from vulnerability descriptions (Amenza Technologies Limited 2002; Ghosh and Ghosh 2010; Birkholz et al. 2010). Their research will be used as a starting point for our future work in this regard. However, these methods are applicable only in networked environments since host reachability is one of the major parameters used in the algorithms – a host connected with another host that has been compromised, itself becomes exploitable. On a single host this connectivity property is analogous to application software dependencies. This information, unfortunately, is more difficult to obtain in an automated manner.

The next enhancement to the security model is to incorporate probabilistic information about the likelihood of attacks and cost/value judgments about user and intervention actions. Within the current framework, all attack scenarios are equally likely. However, an attacker has certain capabilities and typically launches an attack along a path that is more easily exploitable than another. Thus, intuitively it seems that some attacks have a higher probability of occurring than others. Moreover, an attacker would most likely try to maximize the return on his investment (steps preceding the attack and the attack itself). Similarly, users would most likely place different cost values on the various resources that they would like to protect. Such information, if incorporated within the attack graph model, would give a much truer picture of the security risk. Our psychological studies will help us determine what cost/value information is needed to personalize the models to specific users and how likely interventions are to succeed for typical users.

Ultimately, we will explore how much automation is possible for future models by interpreting known vulnerability reports, the user behavior models from the psychological studies, details from a questionnaire given each user, analysis of the typical home user computer systems, and online analysis of what the user is doing.

Conclusions

We have described a first step toward adapting attack graphs from industrial network environments to home computer user environments. We have shown how an attack graph can be augmented and the augmented form can be represented using the PDDL language. Our next steps are to significantly expand the representation, motivate them based on studies of target users and use them as the core of an intelligent agent for helping protect unsophisticated computer users.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 0905232. We gratefully acknowledge their support. We also thank the anonymous reviewers whose comments helped improve the paper.

References

Amenza Technologies Limited. 2002. Secur/tree.

Ammann, P.; Wijesekera, D.; and Kaushik, S. 2002. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 217–224.

Artz, M. 2002. Netspa: A network security planning architecture. M.s. thesis, Massachusetts Institute Of Technology.

Birkholz, H.; Edelkamp, S.; Junge, F.; and Sohr, K. 2010. Efficient automated generation of attack trees from vulnerability databases. In *Proceedings of the 2nd Workshop on Intelligent Security*.

Boddy, M.; Gohde, J.; Haigh, T.; and Harp, T. 2005. Course of action generation for cyber security using classical planning. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling*.

Byrne, Z.; Liff, J.; Weidert, J.; Smith, C.; Howe, A.; and Ray, I. 2011. Start-up effects in policy-capturing: Stabilizing regression coefficients after warm-up. In *Proceedings of the Society for Industrial and Organizational Psychology Conference*.

CERT. 2008. Home computer security. http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-0483.

Cuppens, F., and Ortalo, R. 2000. Lambda: A language to model a database for detection of attacks. In *Proceedings of the 3rd International Workshop on Recent Advances in Intrusion Detection*.

Dawkins, J., and Hale, J. 2004. A systematic approach to multi-stage network attack analysis. In *Proceedings of the 2nd IEEE International Information Assurance Workshop*.

Dawkins, J.; Campbell, C.; and Hale, J. 2002. Modeling network attacks: Extending the attack tree paradigm. In *Proceedings of the Workshop on Statistical Machine Learning Techniques in Computer Intrusion Detection*.

Dewri, R.; Poolsappasit, N.; Ray, I.; and Whitley, D. 2007. Optimal security hardening using multi-objective optimization on attack tree models of networks. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*.

Fox, M., and Long, D. 2003. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Ghosh, N., and Ghosh, S. 2010. A planner-based approach to generate and analyze minimal attack graph. *International Journal of Applied Intelligence*.

Hoffmann, J., and Nebel, B. 2001. The ff planning system: fast plan generation through heuristic search. *J. Artif. Int. Res.* 14:253–302.

Jajodia, S.; Noel, S.; and O'Berry, B. 2003. Topological analysis of network attack vulnerability. In Kumar, V.; Srivastava, J.; and Lazarevic, A., eds., *Managing Cyber Threats: Issues, Approaches and Challenges*, Massive Computing. Springer.

Jha, S.; Sheyner, O.; and Wing, J. M. 2002. Two formal analysis of attack graphs. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, 49–63.

Lippman, R.; Ingols, K.; Scott, C.; Piwowarski, K.; Kratkiewicz, K.; and Cunningham, R. 2006. Validating and restoring defense in depth using attack graphs. In *Proceedings of the Military Communications Conference*.

NIST. 2009. National vulnerability database. Vulnerability Summary for CVE-2009-2408. http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-2408.

NIST. 2010. National vulnerability database. Vulnerability Summary for CVE-2010-0483. http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-0483.

Noel, S., and Jajodia, S. 2004. Managing attack graph complexity through visual hierarchical aggregation. In *Proceedings of the ACM Workshop on Visualization and Data Mining for Computer Security*.

Noel, S.; Jajodia, S.; O'Berry, B.; and Jacobs, M. 2003. Efficient minimum-cost network hardening via exploit dependency graphs. In *Proceedings of the 19th Annual Computer Security Applications Conference*, 86–95.

Obes, J. L.; Sarraute, C.; and Richarte, G. 2010. Attack planning in the real world. In *Working notes of SecArt'2010 at AAAI 2010*.

Ritchey, R., and Ammann, P. 2000. Using model checking to analyze network vulnerabilities. In *Proceedings of the IEEE Symposium on Security and Privacy*, 156–165.

Sasse, M. A.; Brostoff, S.; and Weirich, D. 2001. Weakest link' — a human/computer interaction approach to usable and effective security. *BT Technical Journal* 19(3):122–131.

Schneier, B. 1999. Attack trees. *Dr. Dobb's Journal*.

Sheyner, O.; Haines, J.; Jha, S.; Lippmann, R.; and Wing, J. M. 2002. Automated generation and analysis of attack graphs. In *Proceedings of the IEEE Symposium on Security and Privacy*, 273–284.

Swiler, L.; Phillips, C.; Ellis, D.; and Chakerian, S. 2001. Computer-attack graph generation tool. In *Proceedings of the DARPA Information Survivability Conference and Exposition II*, 307–321.

Templeton, S., and Levitt, K. 2000. A requires/provides model for computer attacks. In *Proceedings of the Workshop on New Security Paradigms*.

Tidwell, T.; Larson, R.; Fitch, K.; and Hale, J. 2001. Modeling internet attacks. In *Proceedings of the 2nd Annual IEEE SMC Information Assurance Workshop*.

Zedeck, S. 1977. An information processing model and approach to the study of motivation. *Organization Behavior and Human Performance* 18:47–77.

FIDIUS: Intelligent Support for Vulnerability Testing

Dominik Elsbroek and **Daniel Kohlsdorf** and **Dominik Menke** and **Lars Meyer**

{elsbroek,dkohl,dmke,lmeyer}@tzi.org
Center for Computing and Communication Technologies
University Bremen

Abstract

Security consultants are often hired to detect attack vectors of a client's computer network. The FIDIUS system is designed to support security consultants performing security checks of a computer network. By following a plan defined by files containing PDDL or intelligently choosing the next victim, the FIDIUS system is able to infiltrate a computer network automatically or control a specific host and check an IDSs' output for resulting incidents. We report first results of the FIDIUS project, the basic idea behind it and describe the current implementation of the system.

1 Introduction

Over the last decades computer security has become more and more important to companies and governments. Nowadays industrial espionage using security issues in computer networks is increasing. Also computer sabotage is a serious threat, not just since the emergence of Stuxnet (Schneier 2010). Companies hire professional penetration testers to elicit and fight these threats. These professionals examine the security condition of the companies' network, operating systems and used software. The penetration testers are security professionals with a corresponding knowledge about security and their tool collection.

Depending on the scenario the penetration tester will use different tools and methods to find possible attack vectors.

A professional intruder (also called "white hat") is for example consulted to check the offered services reachable through the Internet. The job of the white hat in this scenario is to audit the security and the possible attack vectors resulting from the networks' setup. Such services could be a mail service offered by a mail server, a web-service offered by a web server or a special service offered by a proprietary software.

To proceed with their security checks they will conduct several steps. They will collect basic information about the network first. Including addressing scheme, its differing zones or subnets and installed security measures. These could be intrusion detection systems or firewalls. Second he has to elicit knowledge about services used in the network and its hosts offering these services including their specific software version. Third, the consultant has to find out the vulnerabilities and attack vectors resulting of the network specific setup and the offered services within the network. To create a proof of concept for different attack scenarios the consultant could, if agreed, break into systems and demonstrate access to some documents as evidence for his success.

For every step mentioned above there are several public available tools to simplify these tasks. Common tools for host exploration within network systems from in- and outside are often used. Such a library will include active network scanners like *NMap*[1] and also passive tools like *p0f* ("Passive OS Fingerprinting")[2]. NMap is a tool for active network scanning standard coming with many build-in scanning types and a scripting engine.

After exploring a network with its services, vulnerabilities for specific software versions of offered services can be checked against contents of vulnerability databases such as the CVE (Common Vulnerabilities and Exposures)[3] and by searching public available mailing lists[4]. The latter ones also cover up-to-date information about known and newly discovered vulnerabilities (namely the Full Disclosure mailing list[5]). The found vulnerabilities can either be tested by using exploits coming with these vulnerability reports (or any other public available resource) or by the exploits and tools coming with a framework such as Metasploit (MSF)[6].

The MSF comes with a great number of exploits, payloads, scanners and fuzzers[7] which are also a common type of tool to check the robustness and thus the security of software (see (Dai, Murphy, and Kaiser 2010)). In section 5.3 we will discuss this framework in more detail.

These and many other tools and resources are part of the usual equipment of a security specialist (see (Bhattacharyya and Alisherov 2009)).

With the FIDIUS system we aim to support two scenarios. The first is a "gray box" scenario in which users have basic knowledge about assets and the net's topology. The second is a "black box" scenario in which the user wants to explore an unknown network without knowledge about the net. We propose two different agents, one for each use case.

[1] *http://nmap.org/*
[2] *http://www.cougarsecurity.com/p0f*
[3] e. g. US-Cert *http://www.us-cert.gov/* or *https://cve.mitre.org/*
[4] *http://seclists.org/*
[5] *http://seclists.org/fulldisclosure/*
[6] *http://www.metasploit.com/*
[7] *https://secure.wikimedia.org/wikipedia/en/wiki/Fuzz_testing*

Both are described later (see Section 5.2). For us, "gray box" users are system administrators using the system along with a net plan. For them the tool will find real attack vectors. The "black box" users are security experts, hired to break into a computer networks. In this case most properties of the net are unknown and the agent has limited knowledge.

2 Motivation

Black hats[8] attacking a network have unlimited time. In contrast, security consultants have a limited time budget because of financial constraints. Thus it is desirable to have software which is able to support. Such a software should support its user by automatically executing steps or suggesting next possible steps.

This software could not only been used by hired security specialists. Companies are also interested in testing how perceptive and sensitive their intrusion detection systems (IDS) are. Additionally scientific projects such as FIDeS[9] — which are aimed to create a new generation intrusion detection system — could be compared to common Security Incident and Event Management Systems (SIEMS) like ArcSight[10] and Prelude-IDS[11] in conjunction with Snort[12] and e. g. Prelude-LML or any other sensors which are able to report incidents using the IDMEF[13]. A software which can automatically intrude into a network could be used to compare different SIEMS.

FIDIUS[14] is such a software. FIDIUS is a framework which combines methods of the artificial intelligence (AI) with tools and frameworks designed for vulnerability testing. The architecture will be described in the sections 4 and 5.

Security specialists who have their own tools and processes are enabled to model them into the FIDIUS architecture. Or if these tools are even better located within the MSF they can use their tools nevertheless since FIDIUS uses a specially designed interface to the MSF using DRb[15]. Both ways should make their work more comfortable and efficient.

What is needed is a tool assisting a security consultant and a tool which automatically checks if an intrusion detection system recognizes all incidents. FIDIUS is designed to do both. With a component we call *EvasionDB*[16] we are able to examine which exploits will cause events thrown by an IDS. We count how many events are thrown with which severity level. This enables the AI to become more stealthy while blackbox testing. Furthermore FIDIUS is able to let security specialists model their procedures to intrude a network in the Planning Domain Definition Language (PDDL). The modules coming with FIDIUS can efficiently parse PDDL files, execute the defined steps and check the results. This makes it easy to create a set of repeatable tests which can be run like unit tests known from software development.

Additionally we have an AI component working with a neural network. It can be trained just by taking action with the graphical user interface. This AI component is tracking the steps taken by the user and can learn from these.

All the mentioned abilities of FIDIUS can help to save consultants time. It also enables security experts to share their knowledge by exchanging PDDL files. In addition FIDIUS can be used to test the configuration of a companies' IDS, IPS or DLPS.

3 Related Work

There are scripts and tools available which are able to attack a network nearly without user interaction. The free version of the Metasploit framework comes with a script called *db_autopwn*. This script attacks a host or a subnet with available exploits matching the operating system (OS) after scanning these hosts. This causes many events generated by an IDS though there are some options for fine tuning.

There is also the Metasploit framework in its professional version[17]. It enables the user among other features to create tasks which can be repeated automatically.

The novel approach of FIDIUS is to assimilate AI methods into such a system. Inspired by the intrusion detection systems extended by methods known from the research in AI we have played with some different methods like planning, neural networks and anomaly detection. Our goal is to automate attacks and integrate the planner into a testing tool. Therefor we needed a more flexible domain using sensing actions. The results of our approach are similar to the attack trees of (Junge 2010).

One of our AI components is based on "Action State Planning" using PDDL. An early approach for planning in vulnerability testing is described in the work of (Boddy et al. 2005). The domain includes social engineering and was designed for classic planners. The authors assume an omniscient attacker for their domain also known as whitebox testing.

Another approach was published on SecArt10 by (Jorge Lucangeli Obeset al. 2010). The authors presented a solution for transforming information from a pentesting tool into a planning domain scaling very well on medium sized networks. The precondition in this paper is also a whitebox scenario.

4 Architecture Overview

FIDIUS uses the MSF as a basis since it provides a vast list of exploits and payloads for different operating systems

[8] This term refers to a computer hacker and comes from the western movie's evidence being a "bad guy" when wearing a black hat. This term is used unlike "white hat" as an intruder in a good manner for hackers breaking into networks with bad intention. See also *https://secure.wikimedia.org/wikipedia/en/wiki/Black_hat*.

[9] Early Warning and Intrusion Detection System Based on Combined AI Methods, see *http://www.fides-security.org/*

[10] *https://www.arcsight.com/*

[11] *https://www.prelude-ids.com/*

[12] *http://www.snort.org/*

[13] see RFC 4765

[14] formerly *FIDIUS Intrusion Detection with Intelligent User Support*, see *http://fidius.me/*

[15] see *http://segment7.net/projects/ruby/drb/introduction.html*

[16] *http://fidius.me/en/news/released-rubygem-fidius-evasiondb*

[17] *https://www.rapid7.com/products/metasploit-pro.jsp*

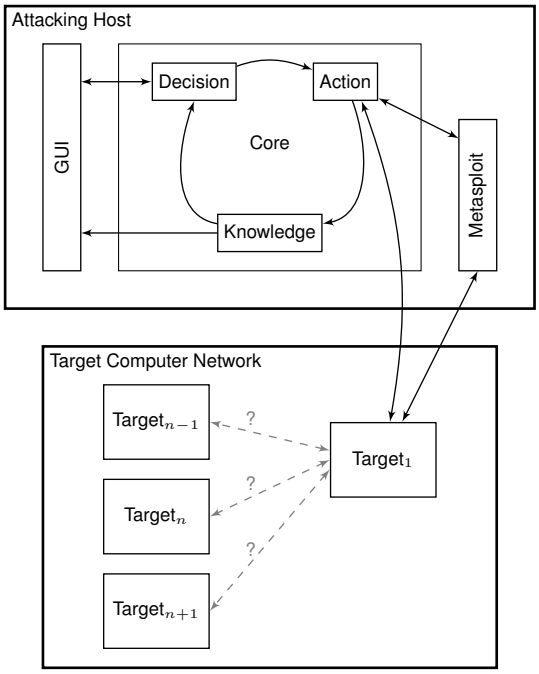


Figure 1: FIDIUS architecture overview

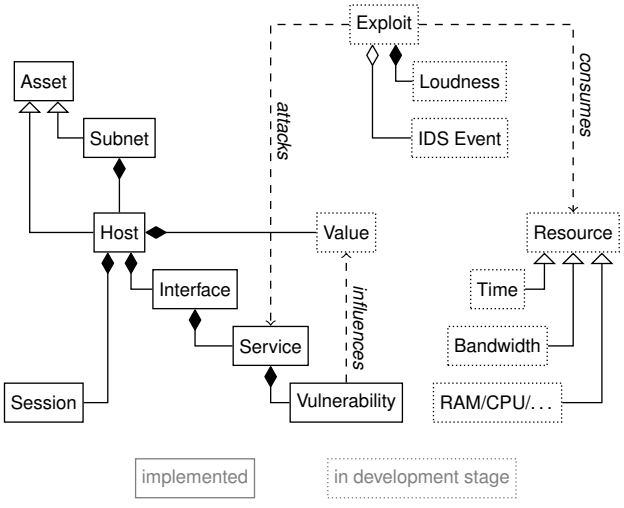


Figure 2: Current and future relationships in the FIDIUS Knowledge

and architectures. On top of this framework FIDIUS defines an architecture model as another layer which is inspired by the Model-View-Controller (MVC) design pattern. We use a *Knowledge Base* as Model, actions (i. e. scans and exploits with payloads) are taken by classes dedicated to the *Action* component. The *Decision* classes are able to use the AI methods and act as an interface to the GUI.

The actions taken by classes placed in the *Action* part result in changes within the knowledge base: Newly discovered hosts will affect the knowledge base since there might also be new services with specific versions etc. The decision component observes the knowledge base and might alter the planning actions (if the newly acquired knowledge indicates this to be useful).

This structure makes the architecture of FIDIUS very expandable and flexible. Third parties are invited to implement their own action or decision components.

4.1 Knowledge

As introduced above, the *FIDIUS Knowledge* is figuratively equivalent to the "model" part in the MVC design pattern.

So the knowledge component (cf. Figure 2) contains various information about *Assets*. An Asset may be a single *Host* or an entire *Subnet* and may provide many different *Services* on different *Interfaces*. A Service in return could have known *Vulnerabilities*. A Host may has also zero or more *Sessions*, which indicates a previous (and/or active) intrusion. Any host has a specific *Value*, which is estimated by the running services and hidden information, such as the number of interfaces or the amount of free disk space.

Besides these asset-related information, there is an exploit-related view, which is currently in discussion. Targeting a barely observable intrusion in a computer network, there are two factors making this actions observable: On the one hand there is something like "*Loudness*" as a result of incidents thrown by an IDS and on the other hand there are "*Resources*" consumed. The main concepts behind this discussion will be further described in section 4.3.

4.2 Decision

The *Decision* component is responsible for planning the next steps based on the current knowledge. Decisions can be made either by an intelligent agent or a user by using the user interface (see section 5.1). A decision component can read from the knowledge database (see section 4.1) and trigger *Actions* to be performed (see section 4.3). Furthermore, changes in the knowledge database are recognized and the agents are updated accordingly.

4.3 Action

A module or class placed in the *Action* component is built to take actions on the network to be attacked or a host located within the network. It corresponds to the "control" part in the MVC pattern. Actions are taken on networks or hosts to gain knowledge or (almost unauthorized) access to an asset.

Since all actions consume resources it is useful to have a measurement making actions comparable w. r. t. their usage of bandwidth, CPU, RAM etc.

The MSF uses "jobs" as an indicator for resource consumption of tasks without any differentiation. But running a web server with a website containing malicious code is not as resource consumptive as running a network scan with NMap. Also waiting for someone falling into a trap is not as visible for some IDSs as sending an exploit downloading a payload having a size of many megabytes after successfully exploiting a vulnerability. Thus we have defined some more meaningful attributes. They will be described in the following section.

Resource Consumption To evaluate the costs of an action we have to consider the consumption of CPU and RAM on both sides since an increasing RAM usage could alert Host Intrusion Detection Systems (HIDS) or monitoring tools like Nagios[18] or Munin[19]. One should also consider the count and size of packets need to be sent to get an exploit working. The latest exploit for the Exim4 mail transfer agent for example, needs an email of at minimum 50 megabytes to be sent[20].

To distinguish the different types of resources we have created classes which are listed below.

Bandwidth usage: The amount of traffic sent over a network by an action should be known. Protocols like Netflow or IPFIX (IP Flow Information Export) are able to account traffic to hosts and sensors watching parameters like "traffic per hour" could generate alerts or even warnings. Anomaly based IDS also could generate a warning or even an alert if a host produces abnormal amount of traffic.

Host resources: The execution of actions will consume resources of both the local and remote host, such as time and bandwidth available by NICs (Network Interface Card) but as mentioned above also CPU, RAM and also additional use of memory on hard disc drives is possible. Also the count of started processes can be monitored by corresponding tools and thus should be taken into consideration.

Intrusion detection: An exploit sent over the network detected by an IDS is *very loud*[21]. This factor is also hard to determine, since almost all rule based IDS have different rule sets and anomaly based IDS have different definitions of normal traffic. Additionally both kinds of IDS can have different configurations[22].

Value/cost ratio: The decision to attack a specific host will be made based on measurable factors. The precedence of the obvious factor — the host's value — should be lowered down, if an exploit has a high resource usage and/or has a great loudness.

[18] *http://www.nagios.org/*

[19] *http://munin-monitoring.org/*

[20] see CVE-2010-4344, *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4344*

[21] in sense of many generated alerting events

[22] In FIDIUS a group started with "evasion testing" to find out which exploits (and their configuration) will not be detected by intrusion detection systems. Therefore, exploits are mapped to IDS alerts and saved as knowledge in a database.

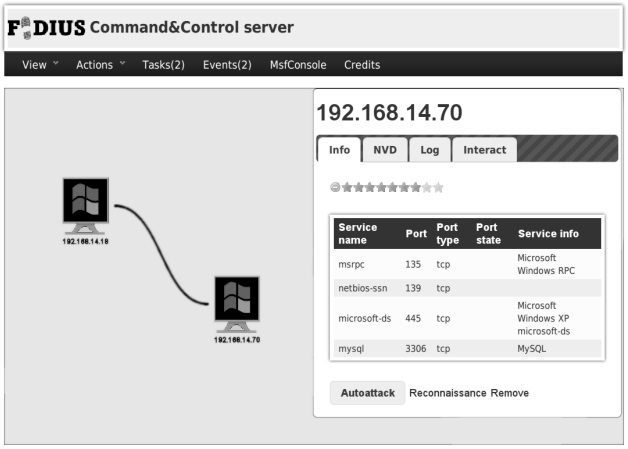


Figure 3: Screenshot of the FIDIUS Command&Control server

Not all of these factors are measurable exactly. For example how much RAM or CPU the execution of an exploit will use on the remote host will depend on different factors. Also the generated network traffic and its conspicuousness will depend on the network and its daily usage. Measuring these factors and taking them into account for decisions taken by the AI have to be done. They are mentioned here for the sake of completeness since IDSs can also take these values as indicators for intruders.

5 Current Implementation

5.1 User Interface

The FIDIUS user interface (see Figure 3), called *Command&Control server*, is basically a (quite simple) web application, which displays the knowledge (see Section 4.1) in form of a network graph.

The Command&Control server itself is written in Ruby using the Ruby on Rails framework[23] and is connected to the architecture's core through a XML RPC interface. The latter fact means two important things:

1. The GUI does not need any database to manage running attacks, information on overtaken hosts, etc.
2. The Rails application might be replaced by any other preferred GUI toolkit.

The AI component (and action component, implicitly) won't run any step without user confirmation for security reasons.

5.2 Artificial Intelligence

We implemented two intelligent agents for our decision component. One using "action state planning" for attack plan generation, the other for the prediction of a host's value. Depending on the scenario and the user we recommend to choose one of these agents. For the "gray box" scenario we implemented an "action state planning" agent and for the

[23] *http://www.rubyonrails.org/*

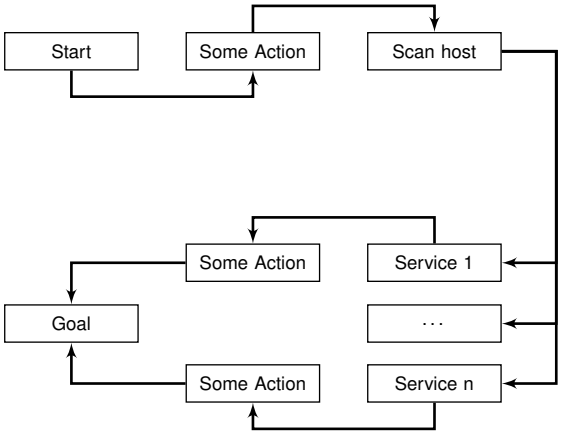


Figure 4: A contingent plan branching after a sensing action

"black box" scenario a "host's value prediction agent" which is a predictive artificial neural net adopting to the users preferences using NMap scan data. In the following we will describe how the agent works.

Action State Planning Agent In the beginning the agent reads all the relevant information from the knowledge database and converts it into PDDL using the following main predicates:

1. **onHost**(X) Our agent is currently on host X.
2. **hostVisible**(X, Y) A host Y is visible from host X.
3. **serviceRunning**(X,Y) Is a service Y running on host X.
4. **inSubnet**(X, Y) Host X is located in subnet Y.

The domain also supports knowledge about running IDSs and Anti Virus software. The planner avoids subnets with an IDS running and hosts with Anti Virus software. Furthermore, the domain contains actions and predicates for plumbing[24], password cracking and domain controllers. If a computer is plumbed, the password cracked or it is in the same subnet with a domain controller, it can be accessed without exploiting it.

All objects in the domain are typed. Existing types are computers, services and nets. The computer type is split into clients and servers.

In our implementation we have two existing methods for plan generation. One for normal plans, the other for contingent plans. For the classical planning we use Hoffmann's "Fast Forward" (FF) (Hoffmann and Nebel 2001) planner and for the contingent plans the "Contingent Fast Forward" (cFF) (Hoffmann 2005) planner. When using the classical planner the user has to specify all hosts, its connections, its subnets and its services in advance. When using contingent planning the user needs the same specification as above except for the services per host. In this case the user specifies in which services he is interested in (he knows an exploit for) and the agent includes a sensing action for these services called "scan". The resulting plan is different when including such actions. The result of the FF planner is a list of actions leading to the goal. cFF results are also a list of actions, except for the sensing actions. In a sensing action the planner creates a branch for each possible outcome of the action (see Figure 4). In our case one branch per possible service. Another sensing action is checking for a web server on a host. The benefit of sensing actions is that we do not have to know everything in advance. For example services might change very quickly while the main net's topology remains. So excluding knowlede that changes often allows more dynamic plans. Furthermore, listing all possibilities in a contingent plan has the advantage that we can decide how to scan or avoid scanning. The information about running services can be derived from context information. For example established connections from one host to another are sufficient indicatiors for the services connected to. Another example is passive scanning by sniffing pakets from the network or monitoring connections using netflow.

The types used in the domain are computers, services and nets. Furthermore there are servers and clients which are of the type computer. The current location of the attacker is given by the host he is currently operating on and the subnet of this host.

The important predicates of our domain for common goals are:

1. **server_owned**(X): Have we exploited a server in net X?
2. **iframe_injected**(X): Do we have an iFrame trap in a net X?
3. **plum_in**(X): Do we have the opportunity to come back in a subnet X through a back door?
4. **powned_dc**(X): Do we have control over a domain controller in a net X?

These predicates are mainly used to create our goals. In some cases the goal might be to hack a specific host, so the goal is specified using the on_host predicate. Another example is to conquer an important source of information in a specific subnet. Assuming that such sources are normally servers, we can use the $server\_owned$ predicate. If we want to install a trap for users of a subnet the goal might be to inject an iframe in this net ($ifrmae\_injected$ predicate). The last example is the option for a return in an exploited net. In this case our goal is to conquer a domain controller or install a back door in the net.

The plan can use the following main actions to accomplish a goal:

1. **Move** Moving from host X to host Y. If Y is visible from X, Y is exploited and the agent has control over host X, the agent can perform this action. The result is the change of location from X to Y.
2. **Scan** The sensing action. Scanning a host Y being on host X. This action is possible if the agent's location is host X and host Y is visible from X. It observes if a service S is running on host Y.
3. **Exploit** Exploiting a host Y being on host X using an exploit for service S. Therefore the agent has to be on host X, host Y is visible from host X and the service S is running on host Y.

[24] We exploited the host before and installed a backdoor.

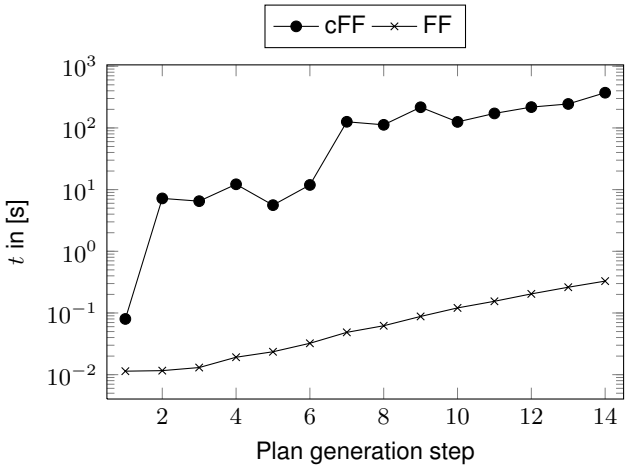


Figure 5: The result of the performance experiment for the FF and the cFF planner

Furthermore, our domain includes actions for different exploiting actions like password cracking, installing back doors or iframe injection. When a password is known for a host or a back door is installed on it, the planner mustn't exploit a host first. Furthermore we have the possibility to include knowledge about domain controllers, NIDS and Anti Virus Software. When a domain controller is hacked, all hosts in the same subnet can be accessed without exploiting them first. Subnets including an NIDS and hosts that include Anti Virus Software, are avoided as targets by the planner.

As indicated above the agent plans all the steps beforehand. During execution the agent takes the next action in the plan, executes it using the associated functionality from the *Action* component and evaluates the results. Because the agent plans everything in advance (except for services in the cFF version) it can't be used in a black box scenario. Because the agent will be integrated in an interactive tool we did a performance experiment. In the experiment we generate random computer networks with n nodes, each with a branching factor of b and m services. Then we executed the planning and measured how long it took the agent to generate a plan. This procedure was repeated while increasing n, m, b by doubling.

As one can see in Figure 5 the cFF planner is slow. The value added by cFF is the possibility for sensing actions. In our case this are nmap scans. If one is willing to provide all running service per host to the planer in advance the agent can use the faster FF planner.

Host's Value Prediction Agent In a black box scenario our planning approach is not an option because all the knowledge has to be known in advance. So we developed an agent which decides the next host to exploit online using an artificial Neural Net. When a new host is added to the knowledge base, it is scanned using the NMap action component. The agent inserts all ports in a feature vector $X = (x_1, ..., x_n)$ with $x_i \in \{0, 1\}$ for $i = \{1 \dots n\}, n \in \mathbb{N}$. The index i refers to a port number while $x_i = 0$ indicates an open port and $x_i = 1$ a closed one. Furthermore, a user specified value v is assigned with each host. We train the neural net with a training data set, where each instance is a tuple of (X, v) in order to predict v for an input X. The training is performed in $100-400$ iterations. The net's topology is an input layer of the size of the feature vector, $m \in \mathbb{N}$ hidden layers, where m is a user specified parameter with default value $m = 10$. The output layer is just one node. We implemented the net using the `ai4r` library[25].

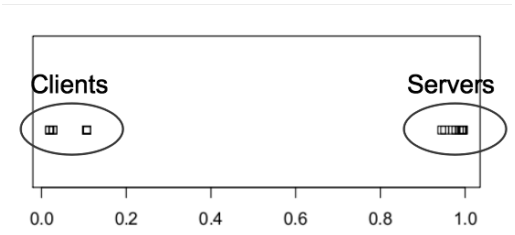


Figure 6: Where the Neural Net maps clients and servers.

During a session the net predicts the value for each host and inserts it into a priority queue. The host with the highest value-prediction is the next host to exploit. Therefore, each "host value" tuple is inserted into a priority queue during run time. The priority queue used is from the ruby `algorithms` library[26]. If the user disagrees with the prediction he can adjust the predicted value and the net is retrained.

In an experiment we collected a small data set of hosts' service vectors. Our use case was a user preferring to exploit servers. So all client's values in the data set were set to 0 and all server's values to 1. The prediction error (mean square error) is 0.03 so the prediction is working. In Figure 6 we can see that the Neural Net is mapping clients near to 0 and servers near to 1.

In a second scenario the intruder wants to gather user data from hacked hosts. We assume that user data can be found in data bases and on the clients. So we mapped all values of clients and all servers with a data base to 1. We mapped all other host values to 0. The prediction error is 0.035, so our method is also working properly in this case.

In the last scenario the intruder wants to obtain access to mail servers. So we set all servers with an open mail port to 1 all other hosts to 0. The prediction error is higher then in the previous scenarios but is also small 0.15.

Our results indicate that the neural net can predict user's preferences if the preferences follow a specific pattern. In our experiment the classes are linear separable. However, Neural Nets are able to adopt to complex non linear functions. So we think this method will also work for other user preferences following a specific pattern.

[25] *http://ai4r.rubyforge.org/*

[26] *http://www.igvita.com/2009/03/26/ruby-algorithms-sorting-trie-heaps/*

5.3 Metasploit framework

The Metasploit framework (MSF) has originally been developed as a framework which enables exploit developers to concentrate on their specific work by providing often required functions, protocols and configurable payloads which are in nearly every exploit the same. In fact, most people want an exploit which automatically opens a port with a command line interface on the remote host after successfully exploiting a host. This is given as a payload by the MSF whereby the port is configurable as an option of the payload.

Nowadays, the MSF is basically a large penetration testing framework which is open-source and contains several exploits for various kind of software. In general, most of the security issues which are used by the exploits coming with the MSF are fixed on an up-to-date system, but there is a good chance that the recent ones will work very well. In fact, most exploits published on security mailing lists[27] will be integrated into the framework short time after certain information is available. For our purpose, testing different IDSs and their configuration, there are also websites hosting old versions of different services. So its also possible to run services with outdated software versions to check what kind of attacks and which steps of an attack are recognized by an IDS.

Besides these exploits, the MSF contains a lot of helper classes (called "auxiliaries"), for many purposes like application version detection or password sniffing. Other useful auxiliaries are different vulnerability scanners, brute-force password cracking tools or a set of simple server systems. In addition, the MSF provides output obfuscation — which allows outsmarting most of the string-matching based detection systems.

Attacking a host means basically to gain access to the operating system's command line interface. But the command line interface differs by operating system. Therefore, the MSF provides a special payload called "Meterpreter"(Miller 2004). It abstracts the operating system's command line interface and brings different advantages. First, it allows to run arbitrary commands without starting a new process, so that *host-based intrusion detection systems* (HIDS) do not recognize an increasing process count. Second, it provides an operating system independent high-level shell for the attacker. This will allow the Meterpreter either to gain higher user privileges or to exploit other hosts. Third, the Meterpreter may also run "Meterpreter scripts" to automate some steps again without creating any recognizable processes.

Usage in FIDIUS The MSF requires manual interaction for each step (i. e. run auxiliaries, prepare and run an exploit, analyse the exploit's output, etc.). Also the framework itself doesn't have a detailed knowledge of the network setup it attacks. Fortunately, the MSF has an API to use the framework not only as an application, but also as a library. There are still some disadvantages, e. g. the MSF requires all modules (including exploit, payload etc.) to have a specific format. The framework then loads all these module files into a manageable format. All modules are loaded into main memory on start up. This results in an almost unacceptable loading time especially while developing new features. Testing any tool using the framework as library can be very time exhausting this way. In FIDIUS we separate the MSF in another application so that the framework is available through network connections.

Thus in the *Action* part of *FIDIUS Core* there is an exploiting part and a scanning part. The exploiting part is an extended and cleaned version of the *db_autopwn* method of the MSF. We have therefore sorted the exploits by date as a first approach. So the order of exploits run against a service starts with the most recent ones. First this results in a much better performance. With a time ordered list of exploit it took us five tries to exploit a "Windows XP with SP2". With a list of exploits only ordered by operating system the amount of exploits tested is 26. Second this results in a much lower resource consumption such as network bandwidth and CPU load. And third, sending only five exploits through a network which is possibly monitored by an IDS will result in much less events thrown by that IDS than sending possibly 40 or even more exploits including shell-code and suspicious OS-API calls.

For the actions to scan hosts or networks we have written different classes implementing different kinds of scans such as ARP scan, Ping scan or port scan. We also changed the OS version detection part thus we can use exploits matching the given hosts set up in more detail. Thus our actions become much more silent measured by the parameters described in section 4.3.

To decrease the great loading time we use the `msfdrbd` *(Metasploit Distributed Ruby Daemon)*. It uses the Ruby standard library `DRb` *(Distributed Ruby)*[28] which allows the bidirectional sharing of (binary) objects (even on different machines). In this way, the `msfdrbd` can constantly run in a background process, instantiate the Metasploit framework and wait for incoming requests. Thus we don't have to wait for all modules to be loaded by the framework while developing the *FIDIUS Core*. Furthermore we have a more scalable software design. Additionally a simple control interface allows the user to interact with the daemon to load and unload server-side framework plug ins. Thus, the execution of exploits will be simplified by providing wrapper methods.

6 Conclusion & Future Work

We presented an architecture for an intelligent vulnerability testing tool and its current implementation. In the architecture part we described the Knowledge, Decision and Action components. The current implementation using *FIDIUS Core* is a user interface, a planning component, a neural net and an exploiting component using the MSF.

In the future, the final FIDIUS system should be an improvement for both penetration testers and network administrators. It simplifies their work and saves their time.

There are still many things we would like to implement and improve. The recognition and definition of the value

[27] not only Full Disclosure, *http://seclists.org/fulldisclosure/*, but also the Metasploit mailing list, *http://mail.metasploit.com/mailman/listinfo/framework* and others

[28] see *http://segment7.net/projects/ruby/drb/introduction.html*

of a host and an exploits loudness have to be fine-tuned. This is important to improve the results of the used AI algorithms. Furthermore we need a bigger knowledge base with IDMEF events from different IDSs with different configurations. This would make an assigned loudness more reliable. We have started this by creating the *EvasionDB* mentioned in section 2 but there are still several tests to be done. In the future we plan to create a database connecting each exploit with all tested IDS wrt their configuration and the events caused by this exploit by an IDS with specific configuration. We plan do use this database to train a neural net. Such a database could probably also be used for other AI methods. This is most important for black box testing.

Additionally, we are still testing the adaptiveness of the neural net in a user study and will test the planner in preconfigured test nets.

7 Acknowledgement

FIDIUS is a project at the Department for Mathematics and Computer Science of the University Bremen. We want to thank all FIDIUS project members who worked with us and made this publication possible. Besides the authors this are: Christian Arndt, Andreas Bender, Jens Frber, Willi Fast, Kai-Uwe Hanke, Dimitri Hellmann, Bernd Katzmarski, Hauke Mehrtens, Marten Wirsik. Furthermore, we would like to thank the project's supervisors which are Carsten Bormann, Stefan Edelkamp, Mirko Hostmann, Henk Birkholz and Jonas Heer.

We also want to mention and to thank the FIDeS project for various inputs and providing a network for testing purposes.

References

Bhattacharyya, D., and Alisherov, F. A. 2009. Penetration testing for hire. In *International Journal of Advanced Science and Technology*, volume 8.

Bishop, C. M. 2007. *Pattern Recognition and Machine Learning*. Springer.

Boddy, M.; Gohde, J.; Haigh, T.; and Harp, S. 2005. Course of action generation for cyber security using classical planning. In *ICAPS 2005*, 12–21. AAAI Press.

Lucangeli Obes, Jorge; Sarraute, Carlos and Richarte, Gerardo 2010. Attack Planning in the Real World. AAAI Press.

Dai, H.; Murphy, C.; and Kaiser, G. 2010. Conguration fuzzing for software vulnerability detection. In *ARES'10: International Conference on Availability, Reliability and Security*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. In *Journal of Artificial Intelligence Research*, volume 14.

Hoffmann, J. 2005. Contingent planning via heuristic forward search with implicit belief states. In *ICAPS'05*, 71–80. AAAI.

Junge, F. 2010. *Dynamische Erstellung von Angriffsbäumen für Rechnernetze mittels eines modularen Werkzeugs*. Diploma thesis, Universitt Bremen.

Miller, M. 2004. Metasploit's meterpreter. *Metasploit documentation*. Online available at *http://www.metasploit.com/documents/meterpreter.pdf*. Accessed 2011/03/21.

Russel, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition.

Schneier, B. 2010. Stuxnet. *Schneier on Security (Weblog)*. Online available at *http://www.schneier.com/blog/archives/2010/10/stuxnet.html*. Accessed 2011/03/29.

Penetration Testing == POMDP Solving?

Carlos Sarraute
Core Security Technologies & ITBA
Buenos Aires, Argentina
carlos@coresecurity.com

Olivier Buffet and **Jörg Hoffmann**
INRIA
Nancy, France
{olivier.buffet,joerg.hoffmann}@loria.fr

Abstract

Penetration Testing is a methodology for assessing network security, by generating and executing possible attacks. Doing so automatically allows for regular and systematic testing without a prohibitive amount of human labor. A key question then is how to generate the attacks. This is naturally formulated as a planning problem. Previous work (Lucangeli et al. 2010) used classical planning and hence ignores all the incomplete knowledge that characterizes hacking. More recent work (Sarraute et al. 2011) makes strong independence assumptions for the sake of scaling, and lacks a clear formal concept of what the attack planning problem actually *is*. Herein, we model that problem in terms of partially observable Markov decision processes (POMDP). This grounds penetration testing in a well-researched formalism, highlighting important aspects of this problem's nature. POMDPs allow to model information gathering as an integral part of the problem, thus providing for the first time a means to intelligently mix scanning actions with actual exploits.

Introduction

Penetration Testing (short *pentesting*) is a methodology for assessing network security, by generating and executing possible attacks exploiting known vulnerabilities of operating systems and applications (e.g., (Arce and McGraw 2004)). Doing so automatically allows for regular and systematic testing without a prohibitive amount of human labor, and makes pentesting more accessible to non-experts. A key question then is how to automatically generate the attacks.

A natural way to address this issue is as an *attack planning* problem. This is known in the AI Planning community as the "Cyber Security" domain (Boddy et al. 2005). Independently (though considerably later), the approach was put forward also by the pentesting industry (Lucangeli et al. 2010). The two domains essentially differ only in the industrial context addressed. Herein, we are concerned exclusively with the specific context of regular automatic pentesting, as in Core Security's "Core Insight Enterprise" tool. We will use the term "attack planning" in that sense.

Lucangeli et al. (2010) encoded attack planning into PDDL, and used off-the-shelf planners. This already is useful,[1] however it is still quite limited. In particular, the planning is classical—complete initial states and deterministic actions—and thus not able to handle the uncertainty involved in this form of attack planning. We herein contribute a planning model that does capture this uncertainty, and allows to generate plans taking it into account. To understand the added value of this technology, it is necessary to examine the relevant context in some detail.

The pentesting tool has access to the details of the client network. So why is there any uncertainty? The answer is simple: pentesting is not Orwell's "Big Brother". Do *your* IT guys know everything that goes on inside *your* computer?

It is safe to assume that the pentesting tool will be kept up-to-date about the structure of the network, i.e., the set of machines and their connections—these changes are infrequent and can easily be registered. It is, however, impossible to be up-to-date regarding all the details of the configuration of each machine, in the typical setting where that configuration is ultimately in the hands of the individual users. Thus, since the last series of attacks was scheduled, the configurations may have changed, and the pentesting tool does not know how exactly. Its task is to figure out whether any of the changes open new dangerous vulnerabilities.

One might argue that the pentesting tool should first determine what has changed, via *scanning* methods, and then address what is now a classical planning problem involving only *exploits*, i.e., hacking actions modifying the system state. There are two flaws in this reasoning: (a) scanning doesn't yield perfect knowledge so a residual uncertainty remains; (b) scanning generates significant costs in terms of running time and network traffic. So what we want is a technique that (like a real hacker) can deal with uncertainty by intelligently inserting scanning actions where they are useful for scheduling the best exploits. To our knowledge, ours is the first work that indeed offers such a method.

There is hardly any related work tackling uncertainty measures (probabilities) in network security. The few works that exist (e.g., (Bilar 2003; Dawkins and Hale 2003)) are concerned with the defender's viewpoint, and tackle a very different kind of uncertainty attempting to model what an attacker would be likely to do. The above mentioned work on classical planning is embedded into a pentesting tool running a large set of scans as a pre-process, and afterwards ignoring the residual uncertainty. This incurs both drawbacks (a) and (b) above. The single work addressing (a) was per-

[1] In fact, this technology is currently employed in Core Security's commercial product, using a variant of Metric-FF.

formed in part by one of the authors (Sarraute et al. 2011). On the positive side, the proposed attack planner demonstrates industrial-scale runtime performance, and in fact its worst-case runtime is low-order polynomial. On the negative side, the planner does not offer a solution to (b)—it still reasons only about exploits, not scanning—and of course its efficiency is bought at the cost of strong simplifying assumptions. Also, the work provides no clear notion of what attack planning under uncertainty actually *is*.

Herein, we take the opposite extreme of the trade-off between accuracy and performance. We tackle the problem in full, in particular addressing information gathering as an integral part of the attack. We achieve this by modeling the problem in terms of partially observable Markov decision processes (POMDP). As a side effect, this modeling activity serves to clarify some important aspects of this problem's nature. A basic insight is that, whereas Sarraute et al. (2011) model the uncertainty as non-deterministic actions—success probabilities of exploits—this uncertainty is more naturally modeled as an uncertainty about *states*. The exploits as such are deterministic in that their outcome is fully determined by the system configuration.[2] Once this basic modeling choice is made, all the rest falls into place naturally.

Our experiments are based on a problem generator that is not industrial-scale realistic, but that allows to create reasonable test instances by scaling the number of machines, the number of possible exploits, and the time elapsed since the last activity of the pentesting tool. Unsurprisingly, we find that POMDP solvers do not scale to large networks. However, scaling is reasonable for individual pairs of machines. As argued by Sarraute et al. (2011), such pairwise strategies can serve as the basic building blocks in a framework decomposing the overall problem into two abstraction levels.

We next provide some additional background on pentesting and POMDPs. We then detail our POMDP model of attack planning, and our experimental findings. We close the paper with a brief discussion of future work.

Background

We fill in some background on pentesting and POMDPs.

Penetration Testing

The objective of a typical penetration testing task is to gain control over as many computers in a network as possible, with a preference for some machines (e.g., because of their critical content). It starts with one controlled computer: either outside the targeted network (so that its first targets are machines accessible from the internet), or inside this network (e.g., using a Trojan horse). As illustrated in Figure 1, at any point in time one can distinguish between 3 types of computers: those under control (on which an agent has been installed, allowing to perform actions); those which are reachable from a controlled computer because they share a sub-network with one of them: and those which are unreachable from any controlled computer.

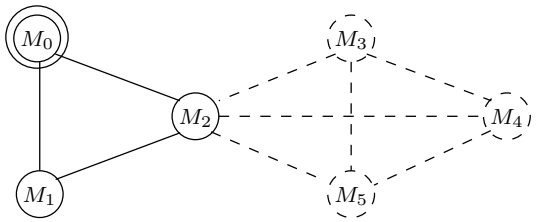


Figure 1: An example network made of two sub-networks (cliques): (M_0, M_1, M_2) (e.g., M_0 an outside computer, M_1 a web server and M_2 a firewall), and (M_2, M_3, M_4, M_5). 1 computer is under control (M_0), 2 are reachable (M_1 and M_2), and 3 are unreachable (M_3, M_4, M_5).

Given currently controlled machines, one can perform two types of actions targeting a reachable machine: tests—to identify its configuration (OS, running applications, ...)—, and exploits—to install an agent by exploiting a vulnerability. A successful exploit turns a reachable computer into a controlled one, and all its previously unreachable neighbors into reachable computers.

A "classic" pentest methodology consists of a series of fixed steps, for example:

- perform a network discovery (obtain a list of all the reachable machines),
- port scan all the reachable machines (given a fixed list of common ports, probe if they are open/closed/filtered),
- given the previous information, perform OS detection module(s) on reachable machines (e.g., run nmap tests),
- once the information gathering phase is completed, the following phase is to launch exploits against the (potentially vulnerable) machines.

This could be improved—a long-term objective of this work—as POMDP planning allows for more efficiency by mixing actions from the different steps.

More details on pentesting will be given later when we describe how to model it using the POMDP formalism.

POMDPs

POMDPs are usually defined (Monahan 1982; Cassandra 1998) by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, O, r, b_0 \rangle$ where, at any time step, the system being in some state $s \in \mathcal{S}$ (the *state space*), the agent performs an action $a \in \mathcal{A}$ (the *action space*) that results in (1) a transition to a state s' according to the *transition function* $T(s, a, s') = Pr(s'|s, a)$, (2) an observation $o \in \mathcal{O}$ (the *observation space*) according to the *observation function* $O(s', a, o) = Pr(o|s', a)$ and (3) a scalar *reward* $r(s, a)$. b_0 is the initial probability distribution over states. Unless stated otherwise, the sets $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{O}$ are finite.

In this setting, the problem is for the agent to find a decision *policy* π choosing, at each time step, the best action based on its past observations and actions so as to maximize its future gain (which can be measured for example through the total accumulated reward). Compared to classical deterministic planning, the agent has to face the difficulty in accounting for a system not only with uncertain dynamics but also whose current state is imperfectly known.

[2] Sometimes, non-deterministic effects are an adequate abstraction of state uncertainty, as in "crossing the street". The situation in pentesting is different because repeated executions will yield identical outcomes.

The agent typically reasons about the hidden state of the system using a *belief state* $b \in \mathcal{B} = \Pi(\mathcal{S})$ (the set of probability distributions over $\mathcal{S}$) using the following Bayesian update formula when performing action a and observing o:

$$b^{a,o}(s') = \frac{O(s', a, o)}{Pr(o|a, b)} \sum_{s \in \mathcal{S}} T(s, a, s')b(s),$$

where $Pr(o|a, b) = \sum_{s,s'' \in \mathcal{S}} O(s'', a, o)T(s, a, s'')b(s)$. Using belief states, a POMDP can be rewritten as an MDP over the belief space, or *belief MDP*, $\langle \mathcal{B}, \mathcal{A}, \mathcal{T}, \rho \rangle$, where the new transition and reward functions are both defined over $\mathcal{B} \times \mathcal{A} \times \mathcal{B}$. With this reformulation, a number of theoretical results about MDPs can be extended, such as the existence of a deterministic policy that is optimal. An issue is that this belief MDP is defined over a continuous—and thus infinite—belief space.

For a finite horizon[3] $T > 0$ the objective is to find a policy verifying $\pi^* = \arg\max_{\pi \in \mathcal{A}^{\mathcal{B}}} J^\pi(b_0)$ with

$$J^\pi(b_0) = E\left[\sum_{t=0}^{T-1} \gamma^t r_t \middle| b_0, \pi\right],$$

where b_0 is the initial belief state, r_t the reward obtained at time step t, and $\gamma \in (0, 1)$ a discount factor. Bellman's principle of optimality (Bellman 1954) lets us compute this function recursively through the *value function*

$$V_n(b) = \max_{a \in \mathcal{A}} \left[\rho(b, a) + \beta \sum_{b' \in \mathcal{B}} \phi(b, a, b')V_{n-1}(b')\right],$$

where, for all $b \in \mathcal{B}$, $V_0(b) = 0$, and $J^\pi(b) = V_{n=T}(b)$.

For our experiments we use SARSOP (Kurniawati et al. 2008), a state of the art point-based algorithm, i.e., an algorithm approximating the value function as the upper envelope of a set of hyperplanes, these hyperplanes corresponding to a selection of particular belief points.

Modeling Penetration Testing with POMDPs

As penetration testing is about acting under partial observability, POMDPs are a natural candidate to model this particular problem. They allow to model the problem of knowledge acquisition and to account for probabilistic information, e.g., the fact that certain configurations or vulnerabilities are more frequent than others. In comparison, classical planning approaches (Lucangeli et al. 2010) assume that the whole network configuration is known, so that no exploration is required. The present section discusses how to formalize penetration testing using POMDPs. As we shall see, the uncertainty is located essentially in the initial belief state. This is different from modeling the uncertainty in pentesting using probabilistic action outcomes as in (Sarraute et al. 2011), which does not account for the real dynamics of the system. Also, as indicated previously, unlike our POMDPs, the approach of Sarraute et al. (2011) only chooses exploits, assuming a naive *a priori* knowledge acquisition and thus ignoring the interaction between these two.

[3] In practice we consider an infinite horizon.

```
states :
terminal
M0-win2000
M0-win2000-p445
M0-win2000-p445-SMB
M0-win2000-p445-SMB-vuln
M0-win2000-p445-SMB-agent
M0-win2003
M0-win2003-p445
M0-win2003-p445-SMB
M0-win2003-p445-SMB-vuln
M0-win2003-p445-SMB-agent
M0-winXPsp2
M0-winXPsp2-p445
M0-winXPsp2-p445-SMB
M0-winXPsp2-p445-SMB-vuln
M0-winXPsp2-p445-SMB-agent
M0-winXPsp3
M0-winXPsp3-p445
M0-winXPsp3-p445-SMB
```

Figure 2: A list of states in a setting with a single computer (M0) which can be a Windows 2000, 2003, XPsp2 or XPsp3, may have port 445 open and, if so, may be running a SAMBA server which may be vulnerable (except for XPsp3) and whose vulnerability may have been exploited.

States

First, any sensible penetration test will have a finite execution. There is nothing to be gained here by infinitely executing a looping behavior. Every pentest terminates either when some event (e.g., an attack detection) stops it, or when the additional access rights that could yet be gained (from the finite number of access rights) do not outweigh the associated costs. This implies that there exists an absorbing *terminal* state and that we are solving a Stochastic Shortest Path problem (SSP).

Then, in the context of pentesting, we do not need the full state of the system to describe the current situation. We will thus focus on aspects that are relevant for the task at hand. This state for example does not need to comprise the network topology as it is assumed here to be static and known. But it will have to account for the configuration and status of each computer on the network.

A computer's *configuration* needs to describe the applications present on the computer and that may (i) be vulnerable or (ii) reveal information about potentially vulnerable applications. This comprises its operating system (OS) as well as server applications for the web, databases, email, ... The description of an application does not need to give precise version numbers, but should give enough details to know which (known) vulnerabilities are present, or what information can be obtained about the system. For example, the open ports on a given computer are aspects of the OS that may reveal not only the OS but also which applications it is running.

The computers' configurations (and the network topology) give a *static* picture of the system independently of the progress of the pentest. To account for the current situation one needs to specify, for each computer, whether a given agent has been installed on it, whether some applications have crashed (e.g., due to the failure of an exploit), and which computers are accessible. Which computers are accessible depends only on the network topology and on where agents have been installed, so that there is no need to explicitly add this information in the state. Fig. 2 gives a `states` section from an actual POMDP file (using the file format of Cassandra's toolbox) in a setting with a single machine `M0`, which is always accessible (not mentioning the computer from which the pentest is started).

Note that a computer's configuration should also provide

```
actions :                    Exploit-M0-win2000-SMB
Terminate                    Exploit-M0-win2003-SMB
Probe-M0-p445                Exploit-M0-winXPsp2-SMB
OSDetect-M0
```

Figure 3: A list of actions in the same setting as Fig. 2, with 1 `Terminate` action, 2 tests, and 3 possible exploits.

information on whether having access to it is valuable in itself, e.g., if there is valuable data on its hard drive. This will be used when defining the reward function.

Actions (& Observations)

First, we need a `Terminate` action that can be used to reach the `terminal` state voluntarily. Note that specific outcomes of certain actions could also lead to that state.

Because we assume that the network topology is known *a priori*, there is no need for actions to discover reachable machines. We are thus left with two types of actions: *tests*, which allow to acquire information about a computer's configuration, and *exploits*, which attempt to install an agent on a computer by exploiting a vulnerability. Fig. 3 lists actions in our running example started in Fig. 2.

Tests Tests are typically performed using programs such as *nmap* (Lyon 1998), which scans a specific computer for open ports and, by analyzing the response behavior of ports, allows to make guesses about which OS and services are running. Note that such observation actions have a cost either in terms of time spent performing analyses, or because of the probability of being detected due to the generated network activity. This is the reason why one has to decide which tests to perform rather than perform them all.

In our setting, we only consider two types of tests:

OS detection: A typical OS detection will return a list of possible OSes, the ones likely to explain the observations of the analysis tool. As a result, one can prune from the belief state (=set to zero probability) all the states corresponding with non-matching OSes, and then re-normalize the remaining non-zero probabilities.

Keeping with the same running example, Fig. 4 presents the transition and observation models associated with action `OSDetect-M0`, which can distinguish winXP configurations from win2000/2003; and following is an example of the evolution of the belief state:

initial	$(0,0,0,0,0,0,\frac{1}{8},\frac{1}{8},\frac{1}{8},\frac{1}{8},\frac{1}{8},0,\frac{1}{8},\frac{1}{8},\frac{1}{8},\frac{1}{8},\frac{1}{8},0,0,0,0)$
`winXP`	$(0,0,0,0,0,0,0,0,0,0,0,\frac{1}{4},\frac{1}{4},\frac{1}{4},\frac{1}{4},0,0,0,0)$
`win2000/2003`	$(0,0,0,0,0,0,0,\frac{1}{4},\frac{1}{4},\frac{1}{4},\frac{1}{4},0,0,0,0,0,0,0,0,0)$

Port scan: Scanning port X simply tells if it is open or closed; by pruning from the belief state the states that match the open/closed state of port X, one implicitely refines which OS and applications may be running.

Action `Probe-M0-p445`, for example, is modeled as depicted on Fig. 5 and could give the following evolution:

initial	$(0,0,0,0,0,0,\frac{1}{8},\frac{1}{8},\frac{1}{8},\frac{1}{8},\frac{1}{8},0,\frac{1}{8},\frac{1}{8},\frac{1}{8},\frac{1}{8},\frac{1}{8},0,0,0,0)$
`open-port`	$(0,0,0,0,0,0,0,\frac{1}{6},\frac{1}{6},\frac{1}{6},\frac{1}{6},0,0,\frac{1}{6},\frac{1}{6},\frac{1}{6},\frac{1}{6},0,0,0,0)$
`closed-port`	$(0,0,0,0,0,0,0,\frac{1}{2},0,0,0,0,\frac{1}{2},0,0,0,0,0,0,0,0)$

Note that a test has no state outcome (the state remains the same), and that its observation outcome is considered as deterministic: given the—real, but hidden—configuration of a computer, a given test always returns the same observation.

```
T: OSDetect-M0 identity
O: OSDetect-M0: * : * 0
O: OSDetect-M0: * : undetected 1
O: OSDetect-M0: M0-win2000                : win 1
O: OSDetect-M0: M0-win2000-p445           : win 1
O: OSDetect-M0: M0-win2000-p445-SMB       : win 1
O: OSDetect-M0: M0-win2000-p445-SMB-vuln  : win 1
O: OSDetect-M0: M0-win2000-p445-SMB-agent: win 1
O: OSDetect-M0: M0-win2003                : win 1
O: OSDetect-M0: M0-win2003-p445           : win 1
O: OSDetect-M0: M0-win2003-p445-SMB       : win 1
O: OSDetect-M0: M0-win2003-p445-SMB-vuln  : win 1
O: OSDetect-M0: M0-win2003-p445-SMB-agent: win 1
O: OSDetect-M0: M0-winXPsp2               : winxp 1
O: OSDetect-M0: M0-winXPsp2-p445          : winxp 1
O: OSDetect-M0: M0-winXPsp2-p445-SMB      : winxp 1
O: OSDetect-M0: M0-winXPsp2-p445-SMB-vuln : winxp 1
O: OSDetect-M0: M0-winXPsp2-p445-SMB-agent: winxp 1
O: OSDetect-M0: M0-winXPsp3               : winxp 1
O: OSDetect-M0: M0-winXPsp3-p445          : winxp 1
O: OSDetect-M0: M0-winXPsp3-p445-SMB      : winxp 1
```

Figure 4: Transition and observation models for action `OSDetect-M0`. The first line specifies that this action's transition matrix is the identity matrix. The remaining lines describe this action's observation function by giving the probability (here 0 or 1) of each possible state-observation pair, defaulting to the `undetected` observation for all states.

```
T: Probe-M0-p445 identity
O: Probe-M0-p445: * : * 0
O: Probe-M0-p445: * : closed-port 1
O: Probe-M0-p445: M0-win2000-p445           : open-port 1
O: Probe-M0-p445: M0-win2000-p445-SMB       : open-port 1
O: Probe-M0-p445: M0-win2000-p445-SMB-vuln  : open-port 1
O: Probe-M0-p445: M0-win2000-p445-SMB-agent : open-port 1
O: Probe-M0-p445: M0-win2003-p445           : open-port 1
O: Probe-M0-p445: M0-win2003-p445-SMB       : open-port 1
O: Probe-M0-p445: M0-win2003-p445-SMB-vuln  : open-port 1
O: Probe-M0-p445: M0-win2003-p445-SMB-agent : open-port 1
O: Probe-M0-p445: M0-winXPsp2-p445          : open-port 1
O: Probe-M0-p445: M0-winXPsp2-p445-SMB      : open-port 1
O: Probe-M0-p445: M0-winXPsp2-p445-SMB-vuln : open-port 1
O: Probe-M0-p445: M0-winXPsp2-p445-SMB-agent: open-port 1
O: Probe-M0-p445: M0-winXPsp3-p445          : open-port 1
O: Probe-M0-p445: M0-winXPsp3-p445-SMB      : open-port 1
```

Figure 5: Transition and observation models for action `Probe-M0-p445`. The transition is again the identity, and the observation is `closed-port` by default, and `open-port` for all states in which port 445 is open.

Another interesting point is that (i) tests provide information about computer configurations and (ii) computer configurations are static, so that there is no use repeating a test as it cannot provide or update any information.

Exploits Exploits make use of an application's vulnerability to gain (i) some control over a computer from another computer (remote exploit), or (ii) more control over a computer (local exploit / privilege escalation). Local exploits do not differ significantly from remote exploits since it amounts to considering each privilege level as a different (virtual)

```
T: Exploit-M0-win2003-SMB identity
T: Exploit-M0-win2003-SMB: M0-win2003-p445-SMB-vuln
                           : * 0
T: Exploit-M0-win2003-SMB: M0-win2003-p445-SMB-vuln
                           : M0-win2003-p445-SMB-agent 1
O: Exploit-M0-win2003-SMB: * : * 0
O: Exploit-M0-win2003-SMB: * : no-agent 1
O: Exploit-M0-win2003-SMB: M0-win2003-p445-SMB-agent
                           : agent-installed 1
```

Figure 6: Transition and observation models for action `Exploit-M0-win2003-SMB`. The transition is the identity except if M0 is vulnerable, where an agent gets installed. The observation is `no-agent` by default and `agent-installed` if the exploit is successful.

computer in a sub-network. As a consequence, for the sake of clarity, we only consider one privilege level per computer.

More precisely, we consider that any successful exploit will provide the same control over the target computer, whatever the exploit and whatever its configuration. This allows (i) to assume that the same set of actions is available on any controlled computer, and (ii) to avoid giving details about which type of agent is installed on a computer.

The success of a given exploit action E depends deterministically on the configuration of the target computer, so that: (i) there is no use in attempting an exploit E if none of the probable configurations is compatible with this exploit, and (ii) the outcome of E—either success or failure—provides information about the configuration of the target. In the present paper, we even assume that a computer's configuration is completely observed once it is under control.

`Exploit-M0-win2003-SMB` is modeled in Fig. 6, and an example evolution of the belief under this action is:

initial	$(0,0,0,0,0,0,\frac{1}{8},\frac{1}{8},\frac{1}{8},\frac{1}{8},0,\frac{1}{8},\frac{1}{8},\frac{1}{8},\frac{1}{8},0,0,0,0)$
`success`	$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0)$
`failure`	$(0,0,0,0,0,0,0,\frac{1}{7},\frac{1}{7},\frac{1}{7},\frac{1}{7},0,\frac{1}{7},\frac{1}{7},\frac{1}{7},0,0,0,0,0)$

Rewards

First, no reward is received when the `Terminate` action is used, or once the terminal state is reached. Otherwise, the reward function has to account for various things:

Value of a computer (r_c): The objective of a pentest is to gain access to a number of computers. Here we thus propose to assign a fixed reward for each successful exploit (on a previously uncontrolled machine). In a more realistic setting, one could reward accessing for the first time a given valuable data, whatever computer hosts these data.

Time is money (r_t): Each action—may it be a test or an exploit—has a duration, so that the expected duration of the pentest may be minimized by assigning each transition a cost (negative reward) proportional to its duration. One could also consider a maximum time for the pentest rather than minimizing it.

Risk of detection (r_d): We do not explicitely model the event of being detected (that would lead to the terminal state with an important cost), but simply consider transition costs that depend on the probability of being detected.

As a result, a transition s, a, s' comes with a reward that is the sum of these three components: $r = r_c + r_t + r_d$. Although some rewards are positive, we are still solving an SSP since such positive rewards cannot be received multiple times and thus cyclic behavior is not sensible.

POMDP Model Generation

Generating a POMDP model for pentesting requires knowledge about possible states, actions, and observations, plus the reward function and the initial belief state. Note first that the POMDP model may evolve from one pentest to the next due to new applications, exploits or tests.

Action and observation models for the various possible tests and exploits can be derived from the documentation of testing tools (see, e.g., nmap's manpage) and databases such as CVE (Common Vulnerabilities and Exposures)[4]. Information could presumably be automatically extracted from such databases, which are already very structured. In our experiments, we start from a proprietary database of Core Security Technologies. The two remaining components of the model—the reward function and the initial belief state—involve quantitative information which is more difficult to acquire. In our experiments, this information is estimated based on expert knowledge.

Regarding rewards, statistical models can be used to estimate, for any particular action, the probability of being detected, and the probabilistic model of its duration. But a human decision is required to assign a value for the cost of a detection, for gaining control over one target computer or the other, and for spending a certain amount of time.

The definition of the initial belief state is linked to the fact that penetration testing is a task repeated regularly, and has access to previous pentesting reports on the same network. The pentester thus has knowledge about the previous configuration of the network (topology and machines), and which weaknesses have been reported. This information, plus knowledge of typical update behaviors (applying patches or not, downloading service packs...), allows an informed guess on the current configuration of the network.

We propose to mimick this reasoning to compute the initial belief state. To keep things simple, we only consider a basic software update behavior (assuming that softwares are independent from each other): each day, an application may probabilistically stay unchanged, or be upgraded to the next version or to the latest version. The updating process of a given application can then be viewed as a Markov chain as illustrated in Fig. 7. Assuming that (i) the belief about a given application version was, at the end of the last pentest, some vector $\boldsymbol{v}_0$, and (ii) T days (the time unit in the Markov chain) have passed, then this belief will have to be updated as $\boldsymbol{v}_T = U^T \boldsymbol{v}_0$, where U is the matrix representation of the chain. For Fig. 7, this matrix reads:

$$U = \begin{pmatrix} p_{1,1} & 0 & 0 & 0 & 0 \\ p_{1,2} & p_{2,2} & 0 & 0 & 0 \\ 0 & p_{2,3} & p_{3,3} & 0 & 0 \\ 0 & 0 & p_{3,4} & p_{4,4} & 0 \\ p_{1,5} & p_{2,5} & p_{3,5} & p_{4,5} & p_{5,5} \end{pmatrix}.$$

[4] http://cve.mitre.org/

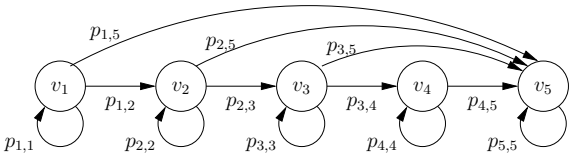


Figure 7: Markov Chain modeling an application's updating process. Vertices are marked with version numbers, and edges with transition probabilities per time step.

This provides a factored approach to compute initial belief states. Of course, in this form the approach is very simplistic. A realistic method would involve elaborating a realistic model of system development. This is a research direction in its own right. We come back to this at the end of the paper.

Solving Penetration Testing with POMDPs

We now describe our experiments. We first fill in some details on the setup, then discuss different scaling scenarios, before having a closer look at some example policies generated by the POMDP solver.

Experiments Setup

The experiments are run on a machine with an Intel Core2 Duo CPU at 2.2 GHz and 3 GB of RAM. We use the APPL (Approximate POMDP Planning) toolkit[5]. This C++ implementation of the SARSOP algorithm is easy to compile and use, and has reasonable performance. The solver is run without time horizon limit, until a target precision $\epsilon = 0.001$ is reached. Since we are solving a stochastic shortest path problem, a discount factor is not required, however we use $\gamma = 0.95$ to improve performance. We will briefly discuss below the effect of changing ϵ and γ.

Our problem generator is implemented in Python. It has 3 parameters:

- number of machines M in the target network,
- number of exploits E in the pentesting tool, that are applicable in the target network,
- time delay T since the last pentest, measured in days.

For simplicity we assume that, at time $T = 0$, the information about the network is perfect, i.e., there is no uncertainty. As T grows, uncertainty increases as described in the previous section, where the parameters of the underlying model, cf. Fig. 7, are estimated by hand. The network topology consists of 1 outside machine and $M - 1$ other machines in a fully connected network. The configuration details are scaled along with E, i.e., details are added as relevant for the exploits (note that irrelevant configuration details would not serve any purpose in this application). As indicated, the exploits are taken from a Core Security database which contains the supported systems for each exploit (specific OS and application versions that are vulnerable). The E exploits are distributed evenly over the M machines. We require that $E \geq M$ so that each machine gets at least one exploit (otherwise the machine could be removed from the encoding).

[5] APPL 0.93 at http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/

Combined Scaling

We discuss performance—solver runtime—as a function of M, E, and T. To make data presentation feasible, at any one time we scale only 2 of the parameters.

Consider first Figure 8 (a), which scales M and T. E is fixed to the minimum value, i.e., each machine has a fixed OS version and one target application. In this setting, there are 3^M states. For $M = 8$, the generated POMDP file has 6562 states and occupies 71 MB on disk; the APPL solver runs out of memory when attempting to parse it. Thus, in this and all experiments to follow, $M \leq 7$.

Naturally, runtime grows exponentially with M—after all, even the solver input does. As for T, interestingly this exhibits a very pronounced easy-hard-easy pattern. Investigating the reasons for this, we found that it is due to a low-high-low pattern of the "amount of uncertainty" as a function of T. Intuitively, as T increases, the probability distribution in the initial belief state first becomes "broader" because more application updates are possible. Then, after a certain point, the probability mass accumulates more and more "at the end", i.e., at the latest application versions, and the uncertainty decreases again. Formally, this can be captured in terms of the entropy of b_0, which exhibits a low-high-low pattern reflecting that of Figure 8 (a).

In Figure 8 (b), scaling the number E of exploits as well as T, the number of machines is fixed to 2 (the localhost of the pentester, and one target machine). We observe the same easy-hard-easy pattern over T. As with M, runtime grows exponentially with E (and must do so since the solver input does). However, with small or large T, the exponential behavior does not kick in until the maximum number of exploits, 10, that we consider here. This is important for practice since small values of T (up to $T = 50$) are rather realistic in regular pentesting. In the next sub-section, we will examine this in more detail to see how far we can scale E, in the 2-machines case, with small T.

Figure 8 (c) and (d) show the combined scaling over machines and exploits, for a favorable value of T ($T = 10$, (c)) and an unfavorable one ($T = 80$, (d)). Here the behavior is rather regular. By all appearances, it grows exponentially in both parameters. An interesting observation is that, in (c), the growth in M kicks in earlier, and rather more steeply, for M. This is not as much the case for the larger T value in (d). Note though that, there, the curve over E flattens around $T = 10$. It is not clear to us what is causing this behavior; cf. the next sub-section.

To give an impression on the effect of the discount factor on solver performance, with $M = 2, E = 11, T = 40$, solver runtime goes from 17.77 s (with $\gamma = 0.95$) to 279.65 s (with $\gamma = 0.99$). APPL explicitly checks that $\gamma < 1$, so $\gamma = 1$ could not be tried. With our choice $\gamma = 0.95$ we still get good policies (cf. further below).

The 2-Machines Case

As hinted, the 2-machines case is relevant because it may serve as the "atomic building block" in an industrial-scale solution, cf. also the discussion in the outlook below. The question then is whether or not we can scale the number of

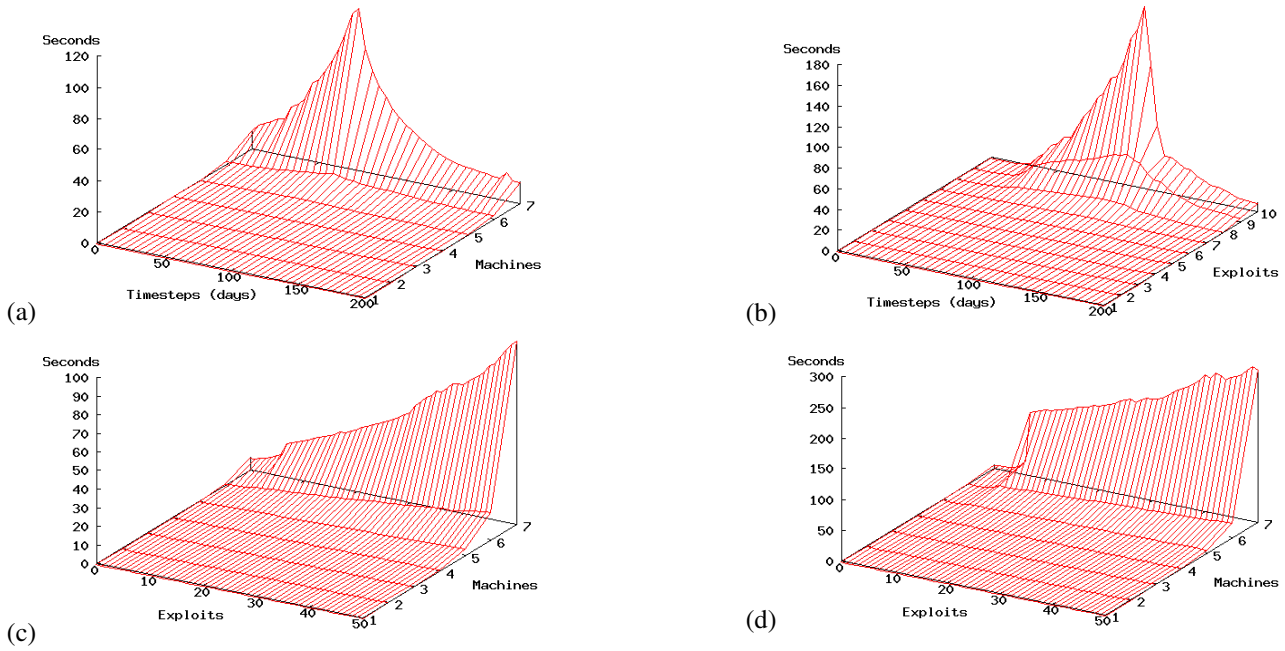


Figure 8: POMDP solver runtime (z axis) when scaling: (a) time delay vs. the number of machines, (b) time delay vs. the number of exploits, (c) machines vs. exploits with time delay 10, and (d) machines vs. exploits with time delay 80.

exploits into a realistic region. We have seen above already that this is not possible for unfavorable values of T. However, are these values to be expected in practice? As far as Core Security's "Core Insight Enterprise" tool goes, the answer is "no". In security aware environments, pentesting should be performed at regular intervals of at most 1 month. Consequently, Figure 9 shows data for $T \leq 50$.

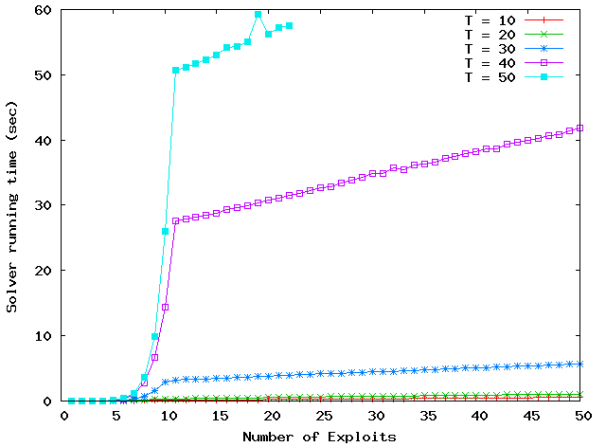


Figure 9: POMDP solver runtime when scaling the number of exploits, for different realistically small settings of the time delay, in the 2-machines case.

For the larger values of T, the data shows a very steep incline between $E = 5$ and $E = 10$, followed by what appears to be linear growth. This behavior is caused by an unwanted bias in our current generator.[6] Ignoring this phenomenon, what matters to us here is that, for the most realistic values of T ($T = 10, 20$), scaling is very good indeed, showing no sign of hitting a barrier even at $E = 50$. Of course, this result must be qualified against the realism of the current generator. It remains an open question whether similar scaling will be achieved for more realistic simulations of network development.

POMDPs make Better Hackers

As an illustration of the policies found by the POMDP solver, consider a simple example wherein the pentester has 4 exploits: an SSH exploit (on OpenBSD, port 22), a wu-ftpd exploit (on Linux, port 21), an IIS exploit (on Windows, port 80), and an Apache exploit (on Linux, port 80). The probability of the target machine being Windows is higher than the probability of the other OSes.

Previous automated pentesting methods, e.g. Lucangeli *et al.* (2010), proceed by first performing a port scan on common ports, then executing OS detection module(s), and finally launching exploits for potentially vulnerable services.

With our POMDP model, the policy obtained is to first test whether port 80 is open, because the expected reward is greater for the two exploits which target port 80, than for each of the exploits for port 21 or 22. If port 80 is open, the next action is to launch the IIS exploit for port 80, skipping the OS detection because Windows is more probable than Linux, and the additional information that OS Detect can provide doesn't justify its cost (additional running time). If the exploit is successful, terminate. Otherwise, continue with the Apache exploit (not probing port 80 since that was already done), and if that fails then probe port 21, etc.

[6]The exploits to be added are ordered in a way so that their likelihood of succeeding decreases monotonically with $|E|$. After a certain point, they are too unlikely to affect the policy quality by more than the target precision ϵ. The POMDP solver appears to determine this effectively.

In summary, the policy orders exploits by promise, and executes port probe and OS detection actions on demand where they are cost-effective. This improves on Sarraute et al. (2011), whose technique is capable only of ordering exploits by promise. What's more, practical cases typically involve exploits whose outcome delivers information about the success probability of other exploits, due to common reasons for failure—exploitation prevention techniques. Then the best ordering of exploits depends on previous exploits' outcome. POMDP policies handle this naturally, however it is well beyond the capabilities of Sarraute et al.'s approach. We omit the details for space reasons.

Discussion

POMDPs can model pentesting more naturally and accurately than previously proposed planning-based models (Lucangeli et al. 2010; Sarraute et al. 2011). While, in general, scaling is limited, we have seen that it appears reasonable in the 2-machines case where we are considering only how to get from one machine to another. An idea to use POMDP reasoning in practice is thus to perform it for all connected pairs of machines in the network, and thereafter use these solutions as the input for a high-level planning procedure. That procedure would consider the pairwise solutions to be atomic, i.e., no backtracking over these decisions would be made. Indeed, this is one of the abstractions made—successfully, as far as runtime performance is concerned—by Sarraute et al. (2011). Our immediate future work will be to explore whether a POMDP-based solution of this type is useful, the question being how large the overhead for planning all pairs is, and how much of the solution quality gets retained at the global level.

A line of basic research highlighted by our work is the exploitation of special structures in POMDPs. First, in our model, all actions are deterministic. Second, some of the uncertain parts of the state (e.g. the operating systems) are static, for the purpose of pentesting, in the sense that none of the actions affect them. Third, unless one models possible detrimental side-effects of exploits (cf. directly below), pentesting is "monotonic": accessibility, and thus the set of actions applicable, can only grow. Fourth, any optimal policy will apply each action at most once. Finally, some aspects of the state—in particular, which computers are controlled and reachable—are directly visible and could be separately modeled as being such. To our knowledge, this last property alone has been exploited in POMDP solvers (e.g., (Araya-López et al. 2010)), and the only other property mentioned in the literature appears to be the first one (e.g., (Bonet 2009)).

While accurate, our current model is of course not "the final word" on modeling pentesting with POMDPs. As already mentioned, we currently do not explicitly model the detrimental side-effects exploits may have, i.e., the cases where they are detected (spawning a reaction of the network defense) or where they crash a machine/application. Another important aspect that could be modeled in the POMDP framework is that machines are not independent. Knowing the configuration of some computers in the network provides information about the configuration of other computers in the same network. This can be modeled in terms of the probability distribution given in the initial belief. An interesting question for future research then is how to generate these dependencies—and thus the initial belief—in a realistic way. Answering this question could go hand in hand with more realistically simulating the effect of the "time delay" in pentesting. Both could potentially be adressed by learning appropriate graphical models (Koller and Friedman 2009), based on up-to-date real-world statistics.

To close the paper, it must be admitted that, in general, "pentesting $\neq$ POMDP solving", by contrast to our paper title (hence the question mark). Computer security is always evolving, so that the probability of meeting certain computer configurations changes with time. An ideal attacker should continuously learn the probability distributions describing the network and computer configurations it can encounter. This kind of learning can be done outside the POMDP model, but there may be better solutions doing it more natively. Furthermore, if the administrator of a target network reacts to an attack, running specific counter-attacks, then the problem turns into an adversarial game.

References

Araya-López, M.; Thomas, V.; Buffet, O.; and Charpillet, F. 2010. A closer look at MOMDPs. In *Proc. of ICTAI-10*.

Arce, I., and McGraw, G. 2004. Why attacking systems is a good idea. *IEEE Security & Privacy Magazine* 2(4).

Bellman, R. 1954. The theory of dynamic programming. *Bull. Amer. Math. Soc.* 60:503–516.

Bilar, D. 2003. *Quantitative Risk Analysis of Computer Networks*. Ph.D. Dissertation, Dartmouth College.

Boddy, M. S.; Gohde, J.; Haigh, T.; and Harp, S. A. 2005. Course of action generation for cyber security using classical planning. In *Proc. of ICAPS'05*.

Bonet, B. 2009. Deterministic POMDPs revisited. In *Proc. of UAI'09*.

Cassandra, A. R. 1998. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. Ph.D. Dissertation, Brown University, Dept of Computer Science.

Dawkins, J., and Hale, J. 2003. A systematic approach to multi-stage network attack analysis. In *Proc. of DISCEX III*.

Koller, D., and Friedman, N. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Kurniawati, H.; Hsu, D.; and Lee, W. 2008. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *RSS IV*.

Lucangeli, J.; Sarraute, C.; and Richarte, G. 2010. Attack planning in the real world. In *SecArt'10*.

Lyon, G. F. 1998. Remote OS detection via TCP/IP stack fingerprinting. *Phrack Magazine* 8(54).

Monahan, G. 1982. A survey of partially observable Markov decision processes. *Management Science* 28:1–16.

Sarraute, C.; Richarte, G.; and Lucangeli, J. 2011. An algorithm to find optimal attack paths in nondeterministic scenarios. Available at http://corelabs.coresecurity.com/index.php?module=Wiki&action=list&type=publication.

Security through Sandboxing? - Towards more secure smartphone platforms

Marcus-Sebastian Schröder and **Florian Junge** and **Jonas Heer**
Center for Computing Technologies (TZI)
Am Fallturm 1, 28359 Bremen, Germany

Abstract

Smartphones are becoming an integral part of modern communication and thus a potential target for attacks. The multitude of interfaces and apps makes it hard to keep an overview of potential threats. Nevertheless, these devices are increasingly used even in safety-critical domains. Even though typical operating systems for smartphones include features intended to prevent malicious applications from running, hackers have found ways to circumvent these security mechanisms. Results from our research show that the situation worsens once the attacker has system-level (or root) privileges. Two major smartphone platforms (iOS and Android) do not provide any means to indicate to the user that her phone has been compromised in such a way. To illustrate this point, we wrote an undetectable data-collecting piece of malware. We argue that the security mechanisms of these operating systems thus do not only fail to fully prevent unauthorized access, but also hinder security tools that could detect such access. Finally, we suggest possible improvements to smartphone security that might mitigate this situation.

Introduction

Smartphones changed the way in which mobile communication devices are used. Calling and texting are no longer the primary purposes of these phones. Quite contrary: since a user may install additional applications, they have turned into small personal computers. As with those, a modern phone is used to access very personal data, such as emails, photos or web sites. However, in contrast to a personal computer, the user only has limited access to the internals of the hardware or the operating system. The choice of additional software suffers from this approach. Usually, there is a predefined source from which software can be installed (App store or Android market). Even if other sources are available (Android offers the opportunity to install software from other markets), some functionality may be missing if the provider does not offer it.

This has not hindered the uptake of smartphone usage. More and more users replace their old feature phone with a smartphone. However, this sometimes happens without considering the security implications that arise through such a device. We argue that upon such a change, people need to reconsider the data that could be obtained through access to their phone. This is especially true if they use it to access sensitive data.

Vulnerability demonstration

To illustrate this problem, we have created a software wiretap for iOS devices. This application accesses a number of data sources within the phone and makes them available to other interested parties. Currently, it can read the data stores for the user's address book, browser history, keyboard cache, captured photos and movies, text messages, phone history, bookmarks and emails. It also can access the camera and audio hardware as well as the GPS module. Finally, some additional data such as serial number, OS version and battery level is read. When the device is connected to the internet, it is possible to transmit this information to another system.

To install this application on a machine, any security exploit can be used that allows the execution of random code. Currently, we use a customized version of the redsn0w jailbreak.[1] This enables us to jailbreak the phone and install our wiretap with a single click. The whole process takes about two minutes and can conceivable to be done while a device is left unattended for a short time. Depending on the exploit used, other attack vectors may become available. To further emphasize how easily data can be accessed once the wiretap is installed on a device, we have created an iPad application that serves as a client to display the gathered information (see figure 1). All functionality of the wiretap can easily be controlled through a touch-based interface. A demonstration of this has been documented in (Schwirblat 2011).

The impossibility of anti-malware apps

Both Android and iOS currently limit the rights of the account that is running user-installed programs. Such a program is saved to a reserved part of the system that acts as a sandbox. A program is only given permission to read and write data within this location. Additionally, an iOS application installed through the AppStore may not use the full Application Programming Interface (API) provided for the system, but just a "public" subset of it. These limitations are supposed to increase the users security. As Ap-

[1] http://blog.iphone-dev.org

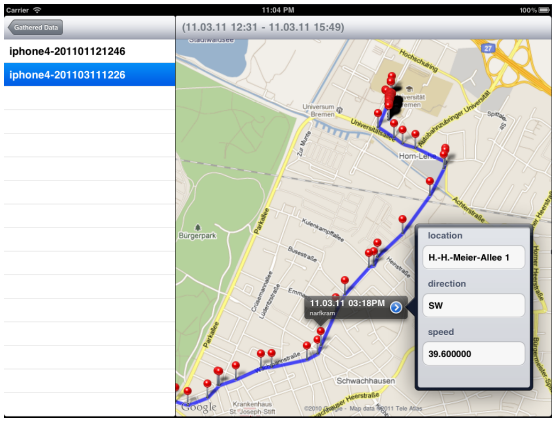


Figure 1: The iPad client, displaying a GPS trace created by our wiretap

ple states[2], "Applications on the device are 'sandboxed' so they cannot access data stored by other applications. In addition, system files, resources, and the kernel are shielded from the user's application space." During a court hearing to establish the legality of jailbreak software, Apples Greg Joswiak stated: "Why do we have these security mechanisms in [the iPhone]? Well, one is we want to protect the product from modification and copying, protect our software from being modified and copied, because these OS modifications can cause damage. And we want to secure the user's data. Again, their E-mail, their context, their pictures, et cetera."(Leonard 2009)
As we demonstrated above, however, there are ways to run software on a smartphone that bypasses these restrictions. In this regard, the security mechanisms intended to protect the device from malicious software turn into limitations that prevent the detection and elimination of such software.

Countermeasures

Since our wiretap does not alter the behaviour of the phone or the appearance of its software, it is not easily detectable by the user. For this reason, the user has to reinstall the operating system of the phone to make sure that such an application is removed from the device. This can be a lengthy process, depending on the amount of data stored on the system, and may take upwards of 30 minutes. It is thus impractical to perform as a daily routine.
To prevent infection with software comparable to ours, we suggest to make sure that control over the physical device is maintained at all times. Furthermore, it is advisable to update to new versions of the operating system as they become available. Usually, these fix security flaws and require the tools used to install unauthorized software to be updated before they can be used again. However, even doing so cannot guarantee that a system will never be compromised by malware. As it is generally considered impossible to produce error-free software, exploits will continue to allow the execution of unsigned code on these platforms. For example, a bug in a previous version of iOS used a flaw in the handling of PDF files to inject software into the runtime.[3] Just visiting a website could execute arbitary code. It would thus have been possible to install our wiretap through this mechanism.
To adequately scan for and act against such attacks, the limitations on 3rd party software as stated in the previous section would need to be removed. Whether this may ever happen is questionable at this time. At this moment, these developments have not encouraged Apple to open up their system any further.

Conclusion and Outlook

Our evaluated smartphone platforms are designed to secure the system and its content by limiting the access to library functions that can be used by 3rd party programmers. Today, however, it can be seen that this limitations have not prevented those with sufficient motivation and resources from accessing data on the device that should be unaccessible to them. Furthermore, the lack of complete system access prevents the creation and usage of tools which would be able to detect code running out of its designated sandbox.
This demonstrates that the attempt to create a secure platform through closing down the system has proven unsuccessful. Therefore, we propose to remove these limitations in order to allow security tools to be created. While this may change the perceived security of a platform, we feel that it would benefit the customer and allow for more secure use of smartphones in sensitive areas.
Another possible approach is the inclusion of a trusted hardware module that maintains control over the system. One might conceive a smartphone platform which deeply integrates a Trusted Platform Module (TPM) to ensure only software is executed which is included on the device's whitelist. Currently, research is trying to determine ways in which such a module could, for example, be integrated into business processes.[4]

References

Barrera, D., and Oorschot, P. V. 2010. Secure software installation on smartphones. *IEEE Security and Privacy* 99(PrePrints).

Hogben, D. G., and Dekker, D. M. 2010. Smartphones: Information security risks, opportunities and recommendations for users. Technical report, European Network and Information Security Agency (ENISA).

Leonard, K. 2009. Section 1201 rulemaking hearing before the copyright office panel.

Schwirblat, C. 2011. Virus im Mobiltelefon.

Traynor, P.; Rao, V.; Jaeger, T.; Porta, T. L.; and McDaniel, P. 2007. From mobile phones to responsible devices. Technical report, Department of Computer Science and Engineering, Pennsylvania State University.

[2] http://images.apple.com/iphone/business/docs/iPhone_Security.pdf

[3] https://github.com/comex/star

[4] http://www.vogue-project.de