



INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

Laboratorio de Ciberseguridad

Persistent Intrusive Evaluation

TESIS

QUE PARA OBTENER EL GRADO DE:
MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN
P R E S E N T A:

Ing. Gabriel Martínez Mendoza

Director de tesis:
Dr. Eleazar Aguirre Anaya

Ciudad de México

Diciembre 2016



Centro de Investigación
en Computación
Instituto Politécnico Nacional



INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de _____ México, _____ siendo las _____ 12:00 _____ horas del día _____ 29 _____ del mes de _____ noviembre _____ de 2016 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

"Persistent Intrusive Evaluation"

Presentada por el alumno:

MARTÍNEZ

Apellido paterno

MENDOZA

Apellido materno

GABRIEL

Nombre(s)

Con registro:


B	1	4	0	4	7	9
---	---	---	---	---	---	---

aspirante de: **MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN**

Después de intercambiar opiniones los miembros de la Comisión manifestaron **APROBAR LA TESIS**, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.


LA COMISIÓN REVISORA

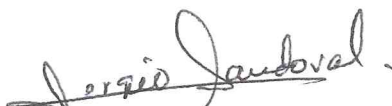
Director de Tesis


Dr. Eleazar Aguirre Anaya


Dr. Ponciano Jorge Escamilla Ambrosio

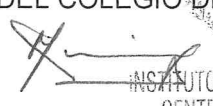

Dra. Nareli Cruz Cortés


Dr. Mario Eduardo Rivero Ángeles


M. en.C. Sergio Sandoval Reyes

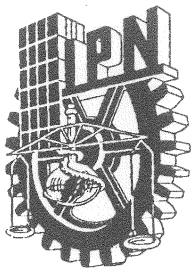

Dr. Moisés Salinas Rosales

PRESIDENTE DEL COLEGIO DE PROFESORES


Dr. Marco Antonio Ramírez Salinas



INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACIÓN
EN COMPUTACIÓN
DIRECCIÓN




INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México el día 5 del mes de Diciembre del año 2016, el que suscribe Gabriel Martinez Mendoza alumno del Programa de Maestría en Ciencias de la Computación con número de registro B140479, adscrito al Centro de Investigación en Computación, manifiesta que es autor intelectual del presente trabajo de Tesis bajo la dirección de Dr. Eleazar Aguirre Anaya y cede los derechos del trabajo intitulado Persistent Intrusive Evaluation, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección gmartinezm0302@gmail.com. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.


Gabriel Martinez Mendoza

Nombre y firma

Resumen

Evaluación Intrusiva Persistente Debido a la importancia que el software, las redes de ordenadores y los sistemas en general han logrado en muchas áreas, es importante asegurarse de que funcionará no sólo correctamente, sino también de manera segura entendiendo que los sistemas, en general, están continuamente expuestos a numerosas amenazas. Para ello, es necesario probar la seguridad de un determinado sistema mediante recursos y experiencia reales de un atacante, además, para representar una evaluación a largo plazo, se necesita que la evaluación persista en la búsqueda de una vulnerabilidad explotable como una contraparte lo haría. En el trabajo actual se aborda un nuevo enfoque para una evaluación de seguridad a largo plazo en busca de posibles ataques explotables; está destinado a representar una vigilancia constante en la manera que una contraparte lo haría sin conocimiento específico de un objetivo en particular, sino una prueba de caja negra. Se diseña una manera de lograr una llamada evaluación intrusiva persistente y se desarrolla un caso de estudio para probar escenarios que ilustran cambios comunes que podrían alterar la exposición a amenazas de un sistema evaluado, como la nueva divulgación pública de una vulnerabilidad y las actualizaciones del sistema, Obteniendo resultados interesantes en el descubrimiento de patrones de ataque, y su explotabilidad de una manera persistente, de una manera que represente la seguridad de un sistema continuamente a través de los cambios mencionados.

Abstract

Persistent Intrusive Evaluation Due to the relevance that software, computer networks and systems in general had achieved in many areas, it becomes important to ensure that it will perform not just correctly, but also securely understanding that systems, generally speaking, are continuously exposed to numerous threats. For this, it is needed to test the security of a given system by means of actual attackers resources and experience, furthermore, to represent a long term evaluation, it is needed for the assessment to persist in the search of an exploitable vulnerability as a counterpart would do. In the current work a novel approach for a long term security evaluation in search of possible exploitable attacks is addressed; it is meant to represent a constant surveillance in a counterpart manner with no specific knowledge of a particular target, but a black box testing. It is designed a way to achieve a so called persistent intrusive evaluation, and a case of study is developed to tests at scenarios illustrating common changes that would alter the threat exposure of an evaluated system, such as new vulnerability public disclosure, and system updates, obtaining interesting results at the attack pattern discovery, and exploitability in a persistent manner, in a way that depicts the security of a system continuously through the mentioned changes.

Summary

The present work is organized as follows.

Chapter 1 Presents and introduction to the topic at hand, and introduces the objectives, justification and boundaries for the instrument proposed within this work to develop automated security testing in a persistent manner.

Chapter 2 Includes the related work on main subjects for the works proposal, it also implies the principal ideas on which the security testing instrument is based to be automated.

Chapter 3 Presents the theoretical framework, introducing the definitions and theory about the subjects addressed within this work, based on the related work.

Chapter 4 Addresses the analysis and design on the instrument to automate security tests including the founding components, as well as the considerations; further it describes the implementation performed for testing.

Chapter 5 Introduces the tests experiments including the tests performed, considerations and results.

Chapter 6 Provides conclusions on the design and the contributions achieved, as well as presents future work.

Bibliography

Thanks

This work was supported by the Consejo Nacional de Ciencia y Tecnología (CONACYT), the Centro de Investigación en Computación (CIC), the Instituto Politécnico Nacional (IPN), from México, and the Universitat Politècnica de Catalunya · BarcelonaTech (UPC).

This work is dedicated to every person involved, in one way or another, in its development and research.

To my family . . .

Contents

Resumen	i
Abstract	iii
Summary	v
Thanks	vii
List of Figures	xiii
List of Tables	xv
1 Security Evaluation Generalities	1
1.1 Problem statement	3
1.2 Objective	4
1.2.1 General objective	4
1.2.2 Specific objectives	4
1.3 Project justification	5
1.3.1 Boundaries	5
1.4 Text organization	6
2 Related Work on Security Evaluation	7
2.1 Classification of security testing models	8
2.2 Fault Injection security testing approach	9
2.3 Vulnerability Detection approach	11
2.4 Vulnerability Scanners comparison	12
2.5 Fuzzy Testing	13
2.6 A wider approach to security evaluation	14
2.7 Proposed model for persistent evaluation	15

3	Theoretical Framework	17
3.1	Penetration Testing	17
3.1.1	Planning and preparation	19
3.1.2	Information gathering and analysis	19
3.1.3	Vulnerability Detection	20
3.1.4	Penetration Attempt	20
3.1.5	Analysis and reporting	21
3.1.6	Clean up	21
3.1.7	Limitations of Penetration testing	22
3.2	Knowledge Representation	22
3.2.1	Knowledge base construction	24
3.3	Evolution strategy	25
3.3.1	Evolutionary strategies principles	25
3.3.2	Mutation and parameter control	26
4	Analysis and Design	27
4.1	Analysis on related work	27
4.2	Proposal Design	29
4.2.1	Enumeration	29
4.2.2	Knowledge representation	31
4.2.3	Search method	35
4.2.4	Design Integration	38
5	Case of Study	41
5.1	Code Development	41
5.1.1	Knowledge Database	41
5.1.2	Enumeration	44
5.1.3	Evolution strategy	45
6	Tests and Results	49
6.1	Test Scenario	49
6.1.1	Expected Results	50
6.1.2	Target selection	50
6.1.3	Results scoring	52
6.2	Testing experiments	53
6.2.1	Testing case one	55
6.2.2	Testing case two	57
6.2.3	Testing case three	59

6.2.4	Testing case four	61
6.3	Result Analysis	63
6.4	Comparison to related work	68
Conclusions and Future Work		71
Bibliography		73

List of Figures

1.1	Vulnerability disclosure not covered by pentesting	4
3.1	Small example knowledge graph. Reprinted from [1]	23
4.1	Flow diagram on enumeration phase	31
4.2	Knowledge graph of a general misconfiguration attack	33
4.3	34
4.4	Knowledge graph on attack patterns	35
4.5	PSO gbest algorithm	37
4.6	Design integration flowchart	40
5.1	Entity Relation model for knowledge database	42
5.2	Enumeration results example	45
5.3	Evolution strategy attack pattern execution example	47
6.1	Attack pattern added to knowledge DB over Vulnix	56
6.2	Attack pattern added to knowledge DB over Metasploitable	56
6.3	Attack scoring testing case one Metasploitable	57
6.4	Attack scoring testing case one Vulnix	58
6.5	Configuration corrected	59
6.6	Attack scoring on testing case two	59
6.7	Testing case three attack pattern fitness	60
6.8	Attack scoring on test case three	61
6.9	Vulnerable version update test	62
6.10	Attack scoring on testing case four	63
6.11	Software update scenario repetition	66
6.12	Software update scenario repetition with search method change	67

List of Tables

3.1	Subject Predicate Object example, Adapted from [1]	23
4.1	Towards automation based on related work	28
5.1	Configuration Vulnerability attack patterns	42
5.2	Implementation Vulnerability attack patterns	43
5.3	Parameter example on CVE-2007-2447	43
6.1	Reported attack patterns by selected vulnerable OS distribution	51
6.2	Vulnerable Distributions	52
6.3	Attack scoring	53
6.4	Number of vulnerabilities discovered for each OS distribution	54
6.5	Comparison with current related work	69

Chapter 1

Security Evaluation Generalities

One of the main security concerns related to organizations are attacks against the organization assets, such as its computer systems; according to Howard and Logstaff[2], an attack can be described as "a series of steps taken by an attacker to achieve an unauthorized result.", according to this definition, an attack, can take several steps to be concluded, and it is commonly aimed to get an unauthorized result in favor of the attackers, which is understood as an intrusion, perpetrated by an intruder, in this case, the attackers. Even when it is a common belief that an attack, or intrusion, may generally be performed by well experienced, expert personnel, some of it can be executed by people without expert knowledge because of the specialized tools that make some of the attacks easier even for non experienced personnel; having this in mind, it is very important for organizations to deal with attacks as a continuous process, understanding that some intrusions can be executed easily and therefore more frequently, furthermore, it is highly desirable to recognize, prior to an actual attack, if the organization assets are exposed to any of these threats, therefore the penetration testing process also referred as pentest comes at hand, which implies gathering information and performing attacks by security expert personnel, aiming to find weaknesses at the organization goods, before an attacker could take advantage of it. A good approach to penetration testing is presented by Chan Tuck[3] published by SANS institute, in the manner Tuck describes the penetration testing is performed for two main reasons, one is to test the intrusion detection and response teams, the second goes in a way to increase the security awareness, by exposing threats or vulnerabilities probably not considered earlier to the administrators and owners; nevertheless the penetration testing process presents some limita-

tions, it is also important to consider, such as the access limitations on the tester side, which may not allow the test to overcome on certain vulnerabilities that would be evident to personnel with a higher level access, in addition the penetration testing process aims to determine vulnerabilities within certain time, if any vulnerability gets discovered after the test this would not be disclosed by it, this means that once done, the constant changes on the organization systems, due to updates, or new vulnerabilities being exposed publicly, could lead an attacker to succeed in attacking the organization assets. Additionally, there are well known, publicly disclosed, vulnerabilities, related to a concrete version of software, these vulnerabilities, being publicly known, can be exploited by any attacker, either an experienced one or not, but there is no certainty that it will be exploitable at a particular given target, furthermore, it is needed to consider vulnerabilities, not in the actual code, but within the configuration, which can lead to information disclosure or further elaborated attacks, this wrong configuration can be difficult to asses for an unexperienced system administrator, and even if the service got wrongly configured, it is not sure whether the attack can take place or not, as those wrong configurations could be exploited in more than one way.

Security Evaluation Terminology

Penetration Test (**pentest**) is "the process of attempting to gain access to resources without knowledge of user-names, passwords and other normal means of access"[4].

Target is the computer system to be subject of the penetration testing.

Black box testing refers to a kind of examination where "...penetration tester does not have complete information about the system being tested"[3].

Persistence from this work's perspective "... means that the adversary is determined (often formally) to accomplish a mission... does not necessarily mean that they need to constantly execute malicious code on victims computer. Rather they maintain the level of interaction needed to fulfill their objectives"[5]

Generation. In the context of an evolutionary algorithm, a generation is a complete iteration of the algorithm process.

1.1 Problem statement

The purpose of this text is to introduce a novel approach for penetration testing, in a persistent manner, performed in an automated fashion, to overcome the persistent problems not covered by the execution of a pentest process, such as new vulnerabilities disclosed, and changes on the system under testing among pentest executions, this comes to avoid uncertainty on attack exploitability between pentest executions and to cover a continuous time of testing vulnerabilities to verify its exploitability before it could an attacker could take advantage of it. Vulnerabilities can be described as "a weakness in a system allowing unauthorized action"[2], and can be from different kinds described as follows. Design vulnerability, a vulnerability inherent in the design or specification of hardware or software whereby even a perfect implementation will result in a vulnerability. Implementation vulnerability, a vulnerability resulting from an error made in the software or hardware implementation of a satisfactory design. Configuration vulnerability, a vulnerability resulting from an error in the configuration of a system, such as having system accounts with default passwords, having "world write" permission for new files, or having vulnerable services enabled [2]. Understanding this could be shown at any time, as indicated by Northcutt et al "It's important to understand that it is very unlikely that a pen-tester will find all the security issues. As an example, if a penetration test was done yesterday, the organization may pass the test. However, today is Microsoft's "patch Tuesday" and now there's a brand new vulnerability in some Exchange mail servers that were previously considered secure, and next month it will be something else. Maintaining a secure network requires constant vigilance."[4] this can be exemplified in figure 1.1, understanding that such changes will be continuous and that security must be assured constantly as described earlier. For such outcome to be achieved, it is needed to come up with a mechanism to both represent and store the actual knowledge of an attacker, as stated in work by Northcutt et al the main difference between an attack and a pentest is the permission, so to automate the test it is needed, represent and store the attacker knowledge in order to be used at the actual analysis in an automated manner, to determine suitable attack patterns and test it, in addition it is needed a way to persist in the attack and find the most suitable time and parameters to take advantage of the attack at a particular target system, and to link it up to the knowledge representation to determine the effectiveness of each attack tested at the target, this can be described as follows: The

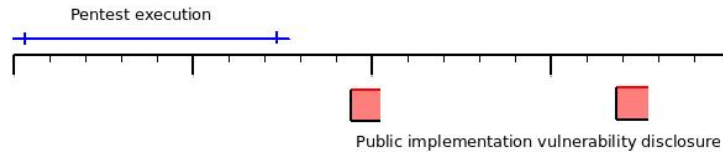


Figure 1.1: Vulnerability disclosure not covered by pentesting

knowledge representation would include both, the attack stages along with its proper parameters to be chosen at the attack execution; to further implement it, being able to determine whether or not it succeeded at the given target. In addition the attack knowledge must be tested to determine the feasibility of it to succeed automatically, which implies a search method to both, test and evaluate each attack.

1.2 Objective

1.2.1 General objective

To design a persistent instrument to evaluate the exploitability of an intrusion attempt on a given target, through the representation of knowledge in combination with an evolutionary strategy as a search method to reduce the possibility of an intrusion prior to an attack by a counterpart

1.2.2 Specific objectives

- Based on previous effort to automate pentesting, consider the objective of bringing persistence to it to develop a persistent evaluation.
- Determine the knowledge representation for storage of attacker's knowledge in a generic manner to craft attack pattern from it, in order to be generic, scalable and portable.
- Determine the search method at use to describe the attack testing in a persistent manner.
- Choose the proper mechanism to correlate the stored knowledge to search method and the attack testing results.

- Determine the threat metrics to be used to evaluate each attack pattern tested in terms of threat metrics and its parameters.
- Determine testing experiment scenarios and considerations to gather persistent evaluation results.

1.3 Project justification

The security assessment is a necessity in any organization, understanding that it is highly desirable to determine the concrete exploitability of a certain attack at a given target system, recognizing that an attacker deals with some limitations on a black box attack, as technical details on the target are not available for the attacker, in such fashion, attackers must carry on with the attack by means of the knowledge and experience of them, this means that any intention for an automated pentest tool, is highly desirable to overcome the security assessment at a given target considering changes along time of the system's production stage. To achieve this, it is needed to represent the expert knowledge concerning attacks, and relate it to a search mechanism to automate the attack as well as determine the suitable parameters targeted to succeed at the system under testing. The relation implied is hard to achieve as both, the knowledge and attack tools, are subject to expertise by the attacker during execution time, most of the time this is achieved by the experience and it could be not well documented for automation purposes, as the knowledge posses multiple variables, in such way of thinking, it is needed a common framework to automate and test each attack in a similar manner as an expert would perform it, this benefits the organizations by showing the possible attacks before it would occur, this aims to aware the system administrators and owners, understanding that such attacks may take place, and that corrective actions must be taken; after which, it is also needed to know if the actions taken were enough to improve the security of the system, which the automated instrument can actually accomplish as well.

1.3.1 Boundaries

This work does propose an instrument intended for long term security assessment with realistic expert knowledge and attack execution, nevertheless, it does not imply directly any sort of defending mechanism at the system

under testing to be evaded by the attack, furthermore it is limited to some representative attack patterns carried on to generic targets, in order to determine the correct instrument functioning, in addition to this the attacks are limited to those prone to be implemented or executed by known tools, understanding that this works approach does not include the attack execution tool development. To take into account security measures at a target, and more advanced attack patterns, further stages in conjunction with the suitable knowledge shall be proposed.

1.4 Text organization

Chapter 1 Begins with an introduction to security evaluation, and its importance, as well as a general vision of what it is needed to achieve it, it continues with a brief description of the proposed instrument given the investigation problem, and continues presenting the objective, justification and boundaries.

Chapter 2 Includes the related work on which the proposal will be based, it recalls knowledge representation models as well as efforts to either automate security evaluations such as pentesting, and some proposed methods to automate and chose from given options.

Chapter 3 Describes the theoretical framework, it does include the references on which the work will be established, considering the references and definitions used during the work explanation.

Chapter 4 Contains the design of the proposed evaluation , it does recall the related work as well as the definitions to accomplish the complete design and its considerations for testing.

Chapter 5 Details a case of study on de development of an instrument to achieve the designed evaluation for testing.

Chapter 6 Introduces the tests experiments including the tests performed, considerations and results.

Conclusions and Future Work Provides conclusions on the design of the evaluation and the testing case ogf study as well as the contributions achieved and presents future work.

Chapter 2

Related Work on Security Evaluation

The security evaluation process corresponds to several single problems, some of them addressed at different scopes under dissimilar circumstances, as the evaluation process includes the actual rating of certain attack patterns, as well as the corresponding representation for the results assessment, this is intended to prevent possible security issues by resemble attack scenarios in a controlled fashion; thereby by maintain control of the attack, it can be measured in some way the level of exposure to potential attacks. As it might be expected, this assessment can be complicated if carried manually as it requires diverse knowledge and tools; in this understanding, work has been done concerning different areas comprising various stages and processes of the security evaluation; such ideas will be retaken for the development of an instrument that can jointly carry out the security evaluation to deliver results that indicate the current status on the security of a system in terms of its exposure to known attack patterns.

In this chapter there will be explored the related efforts developed on behalf of bring some improvement on security evaluations, the main ideas aimed to automate attacks to assess the security as well as approaches closer to the pentesting automation in which this work main proposal will be founded.

2.1 Classification of security testing models

As discussed by G. Tian-yang et. al[6], the main methods on security testing can be organized among several categories. Authors discuss that a formal security testing method, is complex to achieve, due to great variety of possibilities presented on a computer system, therefore, security testing based on models, is a major issue on security testing assessment, as it allows the systems behavior to be represented in a less complex and more general way by a model.

Some of the modeling includes, fault injection, in which vulnerabilities at implementation, it is in the code, are tested and manipulated, fuzzy testing, which allows testing several aspects of security, such as incident response, the so called fuzzy inputs, bring the opportunity to search for abnormal behavior by using random inputs to a system; and vulnerability scanner, that looks for already known public vulnerabilities with help of specialized tools, and compare results to security threats found in each case.

The classification aims for a system representation, as an entity observed as a black box, or from the inside by changing aspects of it and test repeatedly. Some of the most complete related work in terms of automated security assessment include the work by Neha Samant[7] in which the possibility of carrying out a security evaluation automatically is discussed, considering attacks on known protocols as well as Denial of Service ones, taking such attacks as the basis for automated attack execution. The work is presented as a stand alone application which is meant to help on the security evaluation pentest process, it offers a control console to be operated, as it offers a variety of DOS (Denial of Service) attacks, and several protocol abuse attacks, the tool requires the operator to enter the appropriate parameters for each attack; also the results must be interpreted by the operator and the attack scope is limited to those already included in the tool, such coverage is very limited range of possibilities of attack, besides it focuses only on the availability of the evaluated system, although it can be easy to execute a Denial of Service attack, it is necessary to consider more elaborated attacks altogether with publicly known vulnerabilities to have a better approach to full assessment on the security of a system. In the work presented by L. Gen, Wang Bailing et al[8] it is possible to observe as an approach to a more complete tool for testing security in a way more wide, using a database of knowledge, unlike the work of Neha Semat[7], it can be tested among a larger attack database, which allows for greater variety of tests and their results; In addition, simu-

lates more elaborate attacks, which require a series of steps to be completed therefore, the scope is limited with an expert staff to simulate such attacks, however, the work presented still requires an operator to decide the attack parameters and the time to perform it, this limits the ability of the tool consistently deliver information concerning the status of security system without having to repeat the evaluation manually.

On the other hand, there have been focused on solving work only separate parts of the steps in the security assessment process; previous work considers as a good approach described what is required for carry out a security assessment automatically, even though in both cases an operator is necessary, however it is possible to rescue main ideas of each one of these approaches to be resumed in a more complete instrument.

2.2 Fault Injection security testing approach

Fonseca et al[9], discuss a fault injection mechanism, based on fault injection model, to be used in testing the security, in the paper this is limited to a web application, it consists in overriding or removing some functions to sanitize manually typed inputs, the inputs consist mainly in text strings, which subsequently are used to complete a query to a database; this missing sanitization function are considered as vulnerabilities, due to the fact that raw inputs can reach the database queries, this missing input sanitization, considered as a vulnerability, is included intentionally on the code of certain application to test the security under not considered scenarios.

Vulnerabilities are controlled by removing the sanitization functions because the concrete action is known to testers, therefore, automatic attack aims only to SQL injection vulnerabilities which are aimed to be miss from consideration by the missing sanitization functions, the paper presents experimental results with this scope considering specific queries as payload. In a later update to this work [10], the very same method is used to test security mechanisms such as Intruder Detection Systems(IDS) and vulnerability scanners, this is done by reviewing the queries to the database itself, after an attack is performed, as, if the attack was successful, queries must change depending on the payload used, with this information, the false positives reported by the IDS are known explicitly; on the other hand, vulnerability scanners, are tested to find the injected, and therefore, controlled, vulnerabilities; according to these results, tested vulnerability scanners had poor performance

at finding such vulnerabilities, and IDS false positives were not as low as it could be expected. The paper's results seems to have a wider scope on security testing, including code review, by removing some actual code's functions, and security team response, the scope on vulnerabilities and attack is actually pretty limited, attack construction is limited to predefined queries for SQL injection.

Also, considering that the test is done in a white box scenario, full access to code is provided, and database queries are collected in the database itself; this may be useful for security testing on the web app, providing factual information about cases not considered at the app implementation, and the security mechanisms it considers; but having a limited attack scope, some security concerns may not be evaluated, for example other to SQL injection vulnerabilities; nevertheless, the formal methodology may be used to aim for a scenario including several security mechanisms as well as an implementation considering different inputs to be tested for security testing, with a wider scope on attacks, understanding that the methodology used aims to find issues at different security levels, we can think that, narrow this aim, and giving it a wider scope on attacks can be done with similar results.

Morais et al [11], issued the fault injection, by an attack-vulnerability-intrusion model, consisting of the following, a fault is considered, as a result of a successfully attacked vulnerability, to intrude a system. The main difference here is that attacks are generated through an attack tree, which is developed under code's specification, code is analyzed to determine all possible states derived from inputs and actions, the codes possible states are displayed as a tree, then those states leading to unauthorized results are identified, specially those aimed to achieve an intrusion. The methodology to construct such trees is described briefly, first of all, attack intentions must be clear, the attacker capabilities, and the attack models to construct the tree, then attack scenarios are generated and refined. In a later work Moraes et al[12], introduced the concept of threat, identifying and modeling, to assess the known vulnerabilities and generate the attack scenarios based on this information.

The proposed model by Moraes et al[12], includes the vulnerability assessment, and, attack related to it; attack scope may be wider depending on the vulnerabilities found, this wider scope can lead to considering more circumstances and, possibly, determine more potential attacks against a target, if that possibility is what it's looking for, it offers more options for the attack

depending on its final goal, nevertheless, it is presented as a static approach, as if there is any change on the subject of test, the attack tree may not be adequate to this new scenario. On the other hand, we can think about a protocol, as well known steps followed by software, in a more general scope, with this in mind, the general behavior of a piece of software can be modeled, and attacked, making this approach feasible of good results in a less specific scenario.

2.3 Vulnerability Detection approach

So far, the models reviewed consider only the security tests on controlled and well defined environments, as we have discussed, this narrows the search on attack scope and the feasible applications. As part of a different approach to security testing, by attack injection, S. Member and N. Neves[13] presented a much wider approach not requiring vulnerabilities to be known, due to it discovers it.

The main idea on this approach is to generate and find inputs to a system that would not be handled correctly by it, as malformed requests, extremely long inputs, etc., so it is needed to know some aspects of the target system, for example how system inputs are formed and responded, such behavior can be interpreted as a protocol for such system as it can be a known one like SMTP or TCP, by knowing such information it is possible to determine if a particular input was not handled correctly by observing a missing step on the system's protocol for that particular input, which may point to a vulnerability, in such observation consists the security test itself; the main difference with the previously cited papers by Fonseca et al [9][10] is that, in Member, Neves[13] approach, the only thing needed to know is the protocol for that particular system, which in general is standard for certain services such as SMTP or SSH, in the paper, the model is explained with the test case of a service running on a computer, and in Fonseca et al[9][10] papers, full access to code is needed to test.

Neves and Member's approach[13] seems to have a wider attack scope with more emphasis on finding target system's protocol mismatches, nevertheless it lacks of some features presents in previous revisited works, such as the attack main goal, as this just seeks for unusual answers based on the target's system protocol specification, it is not for sure if such behavior can lead to an intrusion, or a Denial of Service; also, it is not considering the already

known vulnerabilities, which can be used, with help of this attack scope, to test more cases and reach the system intrusion, if that was the case.

2.4 Vulnerability Scanners comparison

Up to this point we have issued some methodologies and tools to inject attacks and vulnerabilities, as a way to test the security on a system, nevertheless, there are still some commercial use tools, aimed to detect vulnerabilities on a wide scope of applications and services, as we have seen, the test scope is limited, as it becomes more specific to certain goal, application or vulnerability, even in S. Member and N. Neves[13] papers proposal, it is needed to know the exact protocol to find vulnerabilities.

Commercial use vulnerability scanner tools cope with generic testing problem, by letting the input to be any app or service, testing against well known vulnerabilities, but, as we may expect, this generic testing, has to deal with false positive and false negative cases; this kind of testing is black box access, as it does not require access to anything but the service, this is of course a complex issue and, it is limited to certain restrictions in many cases. J. Bau et al.[14], presented a research conducted among several commercial vulnerability scanner tools, aiming to detect opportunities for future research, based on detection rate for certain vulnerabilities; for the most part, commercial tools perform well for well known and most common vulnerabilities, such as Cross Site Scripting (XSS), and SQL injection, nevertheless, all of this applications had poor to none performance at looking for second order SQL injection, achieved when there is already a malicious record on the database that, when retrieved, can be used to alter further queries, in addition, false positive rate was observed at up to 50% during the comparison among the scanners, and low detection rate for advanced vulnerabilities was observed as well, as the above mentioned second order SQL injection, exemplified in J. Bau et al.[7] results. On the other hand, A. Doup et al.[15], discuss the use of commercial tools aimed explicitly to web applications, together with a web crawler, used to detect state changing on the web application, to test it more deeply, by gathering more complete search on the web application, as it may change its behavior under certain state circumstances, however, this novel approach, has to cope with the problem of false positives, as it only extends the search scope, by running different instances of the very same tool for other states on the web application.

Even with all issues related to commercial tools for generic usage, these are still largely used to conduct security evaluation, such as pentesting, the low detection rate on some vulnerabilities and false positives, are balanced with the adaptability of such tools, as it may be used to conduct test among several distinct kinds of application, services and systems, although, these tools do not inject or generate attacks, these are still the first approach to gather information and possible vulnerabilities when performing a security evaluation, so, such tools must be considered, at the goal of generating attack injections.

2.5 Fuzzy Testing

As most of previously revisited proposals, the main problem here is the test case generation, which leads to attack injection, as part of the testing models, there is the Fuzzy model, which generates test cases, randomly, and test them all, a variation of this is to use genetic algorithms to generate test cases, and prove them, as attacks injected; A. Avancini and M. Cecatto[16], offer an approach to such attack generation.

The main idea here is to find a useful representation, and fitness function, for the genetic algorithm, to assess how effective, in terms of its goal, such attacks can be; in A. Avancini and M. Cecatto[16] proposal's, genetic algorithm inputs are represented by Strings, and they are mixed, in a process called crossover, by splitting the string in parts, and mixing different inputs to obtain a great variety of strings, as the payload changes every time and it can be generated as an attack.

Besides the novel approach to this construction, the fitness function, that evaluates the attack itself, requires a flow graph to be constructed, in order to determine the attack effectiveness, in terms of the attack goal presented on the flow graph; such processing can only be achieved by a full understanding of code, which implies a white box testing. Also, as initial population is created randomly, it is possible that no genetic algorithm input can achieve the flaw state, indicating the attack success, and, it will take longer for the algorithm to find such individuals through genetic algorithm generations.

As fuzzy testing aims to cope with larger attack scopes, this may be complicated to test, as it must be determined the correct representation of individuals in each case as inputs or responses, and, a proper fitness function must be placed. This scope, as genetic algorithms, may be placed for difficult

problems, this is, with an enormous search space, or difficult to evaluate, in terms of computing effort and functions. Nevertheless, in addition with previously stated methodologies and tools, this kind of algorithms may be used, not exactly to obtain test cases, but to choose among sets of them.

2.6 A wider approach to security evaluation

Having considered the security evaluation stated previously it can be recalled the knowledge database that referred to in the Bailing work of Wang et al[8] the idea of a knowledge database adds flexibility to the evaluation as well as their possible results, however, mentioning that knowledge representation as it is described and used, lacks flexibility to generate new attack patterns from those previously stored, in addition, the parameters to use for each pattern of attack must be known by the operator, so it is necessary for the operator to determine the most appropriate according to each case.

There are several options for knowledge representation considering the purpose that is pursued, in the work of Davis et al[17] describes the necessary properties to represent knowledge in general. In the work of Kevin Murphy et al [1] knowledge representation is described by means of graphs, so that each noun can be represented as a node, which relate to other nouns through vertices as verbs, indicating the relationship between the two nouns.

Thus it is possible to relate different attack patterns to form all the ramifications of a single attack, according to properties presented for each evaluating system; this enables not only the flexibility of an attack pattern, if not also the standardization of stored knowledge, as well as its easy interpretation and understanding, in order to make the process easier both to cure graph knowledge as to interpret the results and to add new attack patterns from the general description.

Continuing this train of thought, it is necessary to mention something very important during security assessment, we could say that this is the beginning evaluation process in a black box type, i.e., without knowledge about what is available or installed on the system to be evaluated; the initial step is precisely to determine, with any method, the availability of some specific software in the system being evaluated, so that when an attack gets executed, this has the best chance of being successful.

As mentioned in Neha Semant work[7] there are different ways to get information about evaluating system, one of the simplest is to initiate a connection,

for example by TCP protocol, to expect the response header which must include a successful response in case a service is running and listening at a given port, this information is useful to select the most suitable attack patterns to be simulated on the system under evaluation.

Of course that should be considered various methods of obtaining such information, as this process is extremely important when considering, for example, vulnerabilities on specific versions of software, whose attacks will be able to be successful only in very specific scenarios, allowing more precise selections of attack patterns to evaluate.

On the other hand the Work proposal by Xue Qiu et al[18], proposes an automated approach for the whole pentesting process, including the recognition and vulnerability assessment, to do so they propose a pushdown automata that requires the whole process to be fulfilled, and the acceptance state determines the set of exploitable attacks, the proposal relies on the automata definitions among rules and discoveries to determine suitable attack patterns for a certain goal, nevertheless unlike L. Gen, Wang et al [8] it does not seem to be easily scalable, as knowledge base is highly related to the automata definition and it is, in some way tied to it, in addition, the proposal by Semant Neha [7], does include parameter selection for each available attack, in the case of Xue Qiu et al [18], the parameter are not explicitly mentioned in such understanding, it does not seem to include parameter selection for elaborated attacks, in such fashion, it lacks of flexibility as well as from a generic view of attacks to be easily scalable and understandable for users.

2.7 Proposed model for persistent evaluation

In the current work it is proposed a design to achieve persistent intrusive evaluation, which implies the main ideas carried from related work, first of all, the way of testing by instantiating attacks for different scenarios, here recalled as a long term pentesting, as in the work by G. Tian-yang et. al[6], it is considered discovery on the evaluation of attacks. Unlike the work by Fonseca et al[9] and Member, Neves[13], the aim on the evaluation is to be black box type, understanding that in such works, detailed information is needed, for specific cases and the proposed model is desired to be generic case useful, for this, generic information and general case is needed. The current work's proposal relies on the idea shared by L. Gen[8], Neha Semant[7], and Xue Qui et al [18], about storing information knowledge on how attacks are

performed, such information is in general not well documented and diverse; to overcome such difficulty on store such varied knowledge the work by Kevin Murphy et al. [1] does comes handy by a natural language description of elaborated attacks to be stored.

In addition the work by J. Bau et al.[14] and A. Doup et al.[15], includes the commercial and public tool usage to determine vulnerabilities and its boundaries, this gives the idea of bringing tools already existent for specific tasks at the model with the intention to be generic in usage, but not limited the model for such tools.

The proposal by A. Avancini and M. Cecatto[16], recalls the genetic algorithms to overcome the security testing, the idea would be recalled to develop a search method on evaluating attacks on a long term pentest process.

By recalling such ideas it is proposed an instrument design which be able to determine information about a system, also to include knowledge storage, on how to perform attacks in a general way, as well as a search method to evaluate the suitable attacks at a given system on a pentest fashion.

In addition, the major proposal revisited, does not cover a larger period of time which seems as an opportunity area to be addressed within current work's proposal, also, the recall of main ideas here presented, such as knowledge storage and representation, and parameter selection, to be fulfilled automatically by testing the actual attacks.

In the present chapter, the efforts and models to pursue security evaluation on an automated fashion were recalled as well as it major influences on the current work's proposed model, in addition, the general description on the proposed model was addressed.

In next chapter, the theoretical framework on security evaluation, pentesting, knowledge representation and Evolution strategies as search methods will be discussed to contribute to the model designing.

Chapter 3

Towards Persistence, Theoretical Framework

The current chapter presents the main topics in which the present work's proposal is supported, the chapter is aimed to issue the needed concepts to develop the complete approach for the work objective, which means it would be abridged to some definitions, nevertheless, this can be extended for specific objectives when the intention goes for some very particular issue as some points on this proposal can be discussed in more than one way from different perspectives.

3.1 Penetration Testing

In order to assure the security of a system it is necessary a process to assess it, the penetration testing comes to help with this issue, as it is a manner to identify existing vulnerabilities at a given system, according to a SANS (SysAdmin Audit, Networking and Security Institute) it "usually involves the use of attack methods conducted by trusted individuals that are similarly used by hostile intruders" [3]; this may include the actual execution of a reported vulnerability, it is basically an attempt to breach the security of a network or system, to later on, report the findings to the owner of the system so they can fix the found problems. It must be considered that the penetration testing results do not endure for large periods of time, the pentest is only the vision of a system's security at a given time, during which the vulnerabilities or wrong configurations do not change. The actual difference

between the penetration test and an genuine attack is the permission given by the system owners, it must attempt to gain entry to system's resources without knowledge on any means of access [4]. The pentest is not actually mean to find all the security vulnerabilities, as carried with limitations, also it can report a system as secure one day, and a public vulnerability can expose the system on next day, this means that the security of a system or network must be guarded constantly. For this work we will understand vulnerabilities as "a weakness in a system allowing unauthorized action"[2], and can be from different kinds described as follows.

- Design vulnerability, a vulnerability inherent in the design or specification of hardware or software whereby even a perfect implementation will result in a vulnerability.
- Implementation vulnerability, a vulnerability resulting from an error made in the software or hardware implementation of a satisfactory design.
- Configuration vulnerability, a vulnerability resulting from an error in the configuration of a system, such as having system accounts with default passwords, having "world write" permission for new files, or having vulnerable services enabled [2].

For this text purposes there will be considered the configuration vulnerabilities, also referred as wrong configurations and the implementation vulnerabilities, related to code.

The penetration testing process involves several stages which can be summed up as follows[3].

1. Preparation and planning
2. Information gathering and analysis
3. Vulnerability detection
4. Penetration attempt
5. Analysis and reporting
6. Cleaning up

3.1.1 Planning and preparation

In order to successfully conduct a penetration test it is needed to agree with system administrators and owners, the appropriate goals for the test to be reported, the most general case of the pentest goal is to demonstrate that there exists some executable vulnerabilities at a given target, in this stage it is also important to determine the time frame in which the test will be executed as well as the results and actions taken during the test, this implies a good understanding between the testers and administrators or owners to minimize the risk of any further affectation, as the test does actually carry on with execution of attacks, this must be considered to affect the behavior of the system, in a controlled manner, planning on the test must include legal agreement and protection in case anything goes wrong.

3.1.2 Information gathering and analysis

After the planning it is needed to gather as much information as possible from the target system, it can be seen as the beginning of actual interaction between testers and the system subject of the test. The information gathering can be achieved by the use of several specialized tools. Information can be retrieved from diverse sources and techniques, a network survey can be useful to determine the reachable systems on the network, as well as server's basic information, such as IP addresses, and host names. Thereafter, further information must be acquired, one common way to do this, is to perform a port scanning, which implies retrieving open ports and services running on the target system; the result of the information gathering stage should be a list of systems and IP addresses alongside reported open ports, information about the operating system as well as running services present at the target system. In addition, information can be addressed at public or open sources, such as web pages; public information can also be used to disclose specific details about a system under evaluation.

There exists several methods and tools to perform the enumeration on systems, as this is an interesting task not just for attackers, but for administrators and security auditors or pentesters; one of the best known tools to perform system enumeration is Nmap [19], which includes several methods and options to perform the enumeration it is capable of performing port scanning as well as software versions and Operating System (OS) enumeration to determine the version of it at a black box system to be tested, Bailing et

al [20], used the tool to implement elaborated attacks, from enumeration to attack phase by adding plug-ins to the tool.

3.1.3 Vulnerability Detection

Once the information is gathered it gets required to determine the vulnerabilities existing at the target system related to the information disclosed. To achieve this it is necessary for the testers to have at their disposition a collection of exploits and vulnerabilities to be chosen for each case, in this sense, the knowledge of the testers is the most relevant part of this stage; nevertheless, some of this vulnerability exploit pairs can be correlated by automated tools. The results of this correlation is very important to determine the next steps on attempting an attack on the target system, given the vulnerabilities found.

3.1.4 Penetration Attempt

Having the information collected, it can be determined the suitable targets as ports or services, to be subject of the penetration attempt, maybe not all the information or ports and services may be subject of an attack, but those which are, should be determined in the earlier step and the attack must be executed against it. This attempt execution can be carried out by some specialized tools, commonly requiring customization, this can be used to perform the actual attack attempt, but, it must be clear that, even when a vulnerability gets recognized at a given target, it does not imply that it will be exploitable easily, so it can be understood that the attack attempt may not succeed even when a vulnerability or exploit was discovered. In addition to exploits and vulnerabilities, there can also be attempted to gain access by password cracking, this consist in trying to guess the appropriate user password pair to access a service such as FTP or SSH; the methods to attempt a password cracking can be summed up as follows:

- **Dictionary attack.** This consist in trying word list from a dictionary file.
- **Hybrid attack.** This approach tries variations of words in a dictionary file.
- **Brute force.** Tests all possible combination of characters.

This kind of password cracking attacks can also be executed by some specialized tools, depending on the service prone to this attack. In addition to this method, social engineering can be used to try to retrieve sensible information such as passwords directly from people at the organization, this approach aims to exploit the human resource and can be used for password hijacking, depending on the limitations on the penetration testing that were agreed on the planning stage.

3.1.5 Analysis and reporting

Having conducted the attacks as part of the evaluation, it is needed to report the findings to the system administrators and owners, the report will include a brief description on the process followed as well as the vulnerability findings, this must be separated to give emphasis on those considered more relevant to the organization security, in addition there must be reported the vulnerable scenarios found, the detailed list of any information acquired during this process as well as the vulnerabilities found and its description, to end with suggestions on how to manage such vulnerabilities and fix the scenarios found. For this work's proposal, the evaluation is not meant to have an explicit end, for such reason, the report and analysis will bring up as a threat metric on the successful attack patterns found at a given time, this is reported in a constant manner leaving the corrective action suggestions to the system's administrators.

3.1.6 Clean up

After attempting any attack, it is necessary to clean any trace of the test, this can be achieved by keeping a detailed list on the actions performed, if any change was done to the target system it must be reported and cleaned up, taking into account that this stage must not compromise the normal work on the organization operations. Even though this stage is not explicitly considered for the current's work evaluation proposal, it is needed to have it in mind as the evaluation does consider the attack outcome to be limited by design, this is, it's not considered any change or further damage to the system beyond an unauthorized remote access, for such reason, the clean up stage is dismissed from current work's proposed evaluation.

3.1.7 Limitations of Penetration testing

This kind of testing is not able to identify all the security issues as it is generally carried on as a black box exercise, which means that no information is available for the tester, this implies that any other vulnerability easily discovered by a higher level user may not be achieved by this kind of testing, also, the test is not capable to offer information about new vulnerabilities, specially those disclosed after the test is performed; even if no successful exploit was found it does not mean that the system is secure, as new vulnerabilities can be disclosed in the near future, the configuration may change over time, further updates on existing software or new services activated, such actions imply that the penetration testing can be understood as a limited snapshot of the system's security at a given time. The snapshot is limited to the attacks evaluated and the duration of the test, if any change occurs further to the evaluation, it will not be covered by the assessment and this must be performed again.

On the other hand, the pentester skill must be stored to carry the test for larger periods of time as this work's design proposes, in the next section the knowledge representation considered for the design is briefly explained.

3.2 Knowledge Representation

Knowledge can be understood in terms of the role it plays according to the task at hand, as stated in a work by Davis et al. [17], this can be described as:

- It can be used as a substitution for the thing itself, to take into account consequences related to a word.
- It is a theory of intelligent reasoning.
- A medium for pragmatically efficient computation.
- A set of ontological commitments, which gives context to a word.
- Is a medium of human expression as a language in which things are said about the world.

The knowledge representation by itself is not a data structure, the task for knowledge representation to be worked in this work, can be determined

Table 3.1: Subject Predicate Object example, Adapted from [1]

subject	predicate	object
(LeonardNimoy	profession	Actor)
(LeonardNimoy	starredIn	StarTrek)
(LeonardNimoy	played	Spock)
(Spock	characterIn	StarTrek)
(StarTrek	genre	ScienceFic

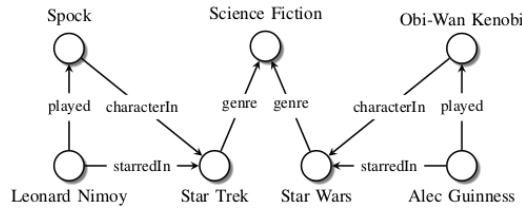


Figure 3.1: Small example knowledge graph. Reprinted from [1]

as a medium of human expression, from which it can be communicated to a machine something about the world. It must be reminded that the knowledge aimed to be represented in this work corresponds to that from attackers experience, this is, knowledge about how to perform an attack given a certain system, in which case, the task to communicate something, as a medium of expression, is adequate as it is being tried to communicate the specialized tools what to do in order to perform an attack, in this way, according to the work by Davis et al. [17], it is needed to overcome the generality and the expression of knowledge represented. In such way of thinking the cognition representation as a knowledge graph can be implemented to describe and expressing ideas. The representation by a knowledge graph is useful to achieve machine readable information, in the work by Nickel et al [1] the knowledge graph is described as simple as a relation between a subject and an object by a predicate, this relation indicates a fact, for example the next relation can represent facts, an example can be found on table 3.1.

Under this assumption the graph gets constructed by adding nodes as subjects and predicates, and edges as predicates, from this understanding the graph does represent knowledge as well as communicating it, being able to be understood with not much effort, a brief example of a knowledge graph can be found at figure 3.1.

3.2.1 Knowledge base construction

The knowledge represented in a graph needs to be complete and accurate according to the relations stored, as this is a complex task to achieve, there exists several approaches to build up a knowledge base, as a graph can be seen as a special case of a knowledge base, the construction can be classified in four main groups:

- Curated. This approach creates the relations manually by a closed group of experts.
- Collaborative . This approach creates relations manually by n open group of volunteers.
- Automated semi-structured. The relations are extracted automatically from semi-structured text by defined rules or regular expressions.
- Automated unstructured. Relations are extracted automatically from unstructured text by natural language processing techniques.

Commonly the more accurate results are achieved by curated approaches, nevertheless, this requires human experts to grow the graph. This kind of knowledge base is aimed to represent a wider approach on general knowledge of any kind, with possible relations among any two nodes.

Having considered the knowledge base construction it is needed to consider the task of creation itself to obtain a knowledge base as a graph. This is the most complicated approach to generate a knowledge base since it must rely on the expert to be curated properly this can be a hard task to achieve, but it can be easily carried out by defining boundaries to which it must communicate, in such way, graph knowledge can be limited by some subjects to be easily understood, this can be seen as a closed world assumption [1], in which only the explicit relations are considered to be fact, the other approach, an open world assumption, can be interpreted in any way, as a relation may be understood to not be on the graph, but to exist in reality. So, for some limited scopes, it can be generated a knowledge graph under a closed world assumption in a curated approach to give specific interpretation on the knowledge stored.

In addition to the knowledge representation it is needed a way to actually test the attacks retrieved and stored by the knowledge representation, in the next sect

3.3 Evolution strategy

It can be understood as an evolution strategy an algorithms that are most commonly applied to black box optimization problems in continuous search spaces[21], from algorithmic point of view this methods sample new candidate solutions stochastically. This search principles are based on those by biological evolution, such search can address an optimization problem by implementing a repeated process of small variations to candidate solutions followed by selection in each iteration, also called a generation, to generate new candidate solutions, this is done based on the best fitness of each candidate solution. This strategies are commonly used to address the problem of optimization as a black box exercise under continuous search spaces, it is considered a function $f(x)$ that is the object of optimization and a candidate solution denoted as x , there is no other assumption over $f(x)$ other that it can be evaluated for every candidate solution x , the main problem is to generate solutions x with values of $f(x)$ prone to the optimization searched.

3.3.1 Evolutionary strategies principles

Evolutionary strategies rely on principles from biological evolution, it is assumed a set of candidate solution also called individuals, composed by the actual parameters for the candidate solution, and the fitness value; in a general way the evolution consist of the procedure:

1. One or several parents are picked from the population (mating selection) and new offspring are generated by duplication and recombination of these parents.
2. The new offspring undergo mutation and become members of the new population.
3. Environmental selection reduces the population to its original size.

From this it can be retrieved the main principles that can be specified according to the task at hand. For this work purposes the most important principle to understand is the mutation and parameter control.

3.3.2 Mutation and parameter control

Mutation on candidate solutions implies adding small random changes to an individual; these changes are typically applied to all variables from the candidate solution. The variation on changes performed can be controlled by parameters related to the context and may vary over time, this is called parameter control, and it serves as a boundary to the mutation on candidate solutions, the parameter control can either be fixed or vary over time. Parameter control is not always directly inspired by biological evolution, but it is indispensable to evolutionary strategies.

While evolution strategies can be a complex issue, due to the many considerations and options prone to the strategy itself, the most important issue to consider within this work's boundaries, is the fact that it can iteratively compute a fitness function and that it would change the parameters on a big search space, which comes handy for this work's proposal. Further detail on the evolution strategy will be addressed on Analysis and design chapter.

As presented during this chapter, the recalled concepts can be matched with the ideas from related work understanding that the relation among this is the object of the proposed design, also, it is needed to understand that the concepts related are mainly aimed to achieve persistence, understood as a long term continuous interaction between tester and the system under evaluation.

In the next chapter the particular details on analysis and design for the current work's proposal will be addressed.

Chapter 4

Persistent Intrusive Evaluation Analysis and Design

The present chapter will disclose the analysis and recall from related work on security evaluation, knowledge representation and evolution strategy to develop the current work's proposed design for a persistent intrusive evaluation, and the specifications on the design proposed will be presented.

4.1 Analysis on related work

The pentesting process is intended to cope with the uncertainty of knowing whether the attack would be exploitable or not, as it actually tries the attack and reports the findings. On the contrary, pentest process requires a security expert, as the only limitation on which attacks to perform is the expert knowledge of those performing the pentest, and as stated previously, this can be expensive. The current work proposes a new approach, without the supervision of an expert, by having the knowledge representation stored, and trying each suitable attack, exploring over some parameters options, over a large period of time, aiming to verify whether or not each suitable attack can be exploited at a system under testing even through the changes and updates that can be applied to it over time, intended to mimic the behavior of a well motivated attacker as persistence. According to Tipton and Nozaki [5] persistence can be understood as "*...means that the adversary is determined (often formally) to accomplish a mission... persistent does not necessarily mean that they need to constantly execute code on victim*

Table 4.1: Towards automation based on related work

Title	Knowledge DB	Elaborated Attacks	Recognition	Parameter Control	Time coverage
Design and Implementation of a Network Attack Platform Based on Plug-in Technology[8]	Plugin	Yes	previously performed	Operated manually	Snapshot
Automated Penetration Testing[7]	Hardcoded	No	previously performed	Operated manually	Snapshot
An Automated method of penetration testing[18]	Hardcoded	Yes	Automated	None	Snapshot

computers ... they execute a "low and slow" approach to continuously gather information through sustained monitoring for long-term presence. They wait patiently for the appropriate time to strike.", understanding that this kind of behavior, from a counterpart is dangerous for an organization, and it aims to be not noticed until the actual attacks takes action, which implies that the design is also aimed to self-test the security of a certain system, additionally this could lead to prevention on exploitable vulnerabilities at the system, by discovering exploitable attack patterns at the system under testing, to be corrected before an attack by a counterpart could take place. Additionally, as this means to have a permanent pentesting process running, it is important for the design to be harmless for the system under testing, understanding that the tests are actually attacking the system, it is expected that such attacks do not alter the usage of any particular service, as this may affect the exploitability of a certain attack pattern, leading to erroneous results as well as unauthorized results, which is not what the design is intended for.

Considering the recalled related work, it can be considered the two main related work proposal for the complete design aim, to determine how to achieve automation on pentesting, as it rely on similar objective, as proposed, the design can be divided in knowledge DB, the search method, and the tests performed, as the aim is automation, the level of interaction with users shall be limited, a comparative on such subjects of interest can be seen at table 4.1.

From the comparative we can recall that, for the most part, parameter selection and search method are complete unaware of automation, which is the main issue to be addressed, understanding that this is the crucial stage to perform an attack, by correlating it and determine the necessary steps and instructions to execute it, as well as needed parameters in such a way that this can be operated without interaction from any personnel. In addition, the knowledge DB, as hard coded and plugin based, can be not so flexible to add new knowledge or to apply new methods to it, which would also be addressed within the design.

First it will be discussed the proposal details and general layout, later it will be discussed the test scenario and results, last there will be the conclusions and further work.

4.2 Proposal Design for a persistent intrusive evaluation

The proposed design integrates basically three main elements, corresponding to the main intrusion phases according to Heberlein et al [22], an attack consists of three phases; the preparation phase, in which the attacker must obtain information about the system target of the attack, it is needed to acquire general and system specific information from various sources. The attack stage, this is where the actual attack takes place, considering the information collected on the previous phase and performing the most suitable attack in each case, this considers the attack as a chain of actions. Finally, the post-attack phase, which implies the attacker goal, it can be completed by stealing or disseminating information, as in spam, or decrease the system capabilities, as in a Denial of Service (DOS) attack. Considering this description of an attack, the proposed design considers two of the mentioned stages, not considering the post-attack phase, as the goal is not to actually affect the system, or disseminate any information, but to prevent an attacker from doing it, the proposed design integrate three main aspects described as follows. The enumeration of the system under testing, to retrieve information about the system, and possible attack points, the knowledge storage, to be used on demand, instead of having an expert performing a pentest every time, and a search method, which allows the proposed design to perform the tests, and persist over this testing as well as respond to changes at the system under testing, and the knowledge stored; this will also allow to search among different parameters for each attack pattern. In next sections it will be explained each main component.

4.2.1 Enumeration

System enumeration, also known as recognition, is a complicated task to achieve, for instance there could be security controls intended to prevent attempts to gather information at a system, and that information itself may

not be easily gathered from the system even when getting access to the system, similarly, OS enumeration, as well as software version are complicated, prone to interest tasks.

Some work has been done in such area, Yarochkin et al. [23], proposed a modification to the Xprobe tool, aimed to reduce the traffic generated by the enumeration process, which can be used to detect and block this kind of efforts on the defensive side, in addition, several works aim to determine the OS of the target system, the most common approach is by Time To Live (TTL) responses, which may vary from OS to OS, an example can be found on work done by Vanauble et al. [24], in which TTL approach is used to determine OS on routers.

For the purpose of this design it is needed to perform a port scan, to determine the open ports, having in mind that the well known ports correspond to known services, therefore it can be known which services are running and try out the misconfiguration attacks, as this attacks does not rely on any particular version, but in the service itself, later, it can be determined the system specific software versions, to match known vulnerabilities, the cited tool offers both solutions and flexibility to extend the enumeration over large periods of time, trying to avoid some security control methods, which is desirable for the design.

In addition, for the design, the enumeration must be repeated over large periods of time, to determine if there is any change either at the system under testing or at the knowledge stored, by any new vulnerability or attack pattern added to it, in either case, the suitable attack patterns must be found and generate the test instances. This can be summed up in the figure 4.1, it can be noticed that this is an ongoing process, designed to find suitable attack vectors at any particular time.

Also, there must be a way for the enumeration to evade security controls, if any, to do so, in a very general way, there would be retry delays on each test, intended to deal with firewall filtering, by decreasing enumeration traffic on large periods of time and gathering information, this intends to imitate the behavior that an attacker could present at a certain system while in the preparation stage of an attack. Such functionality can be described as follows, if at any given enumeration results, there is less information discovered, such as less open ports found, the delay time among tests sent by the tool will be increased pseudo-randomly, this behavior intends to cope with possible

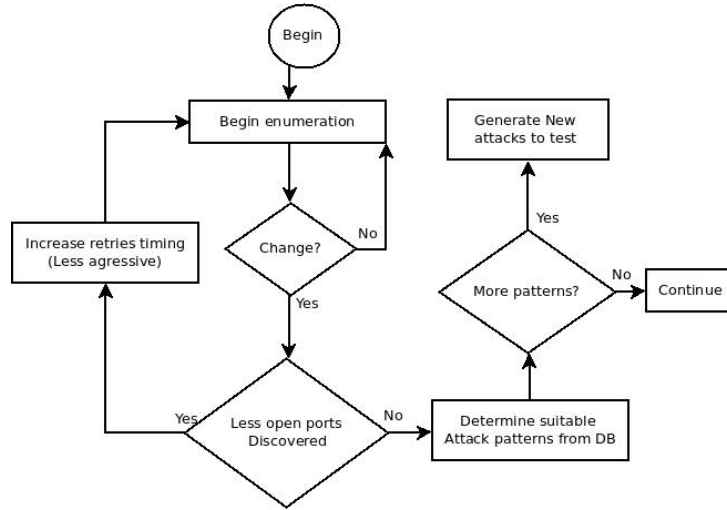


Figure 4.1: Flow diagram on enumeration phase

pattern recognition at the security control in order to evade it and be able to gather information from the system being evaluated.

4.2.2 Knowledge representation

Knowledge is a complicated concept to deal with, it implies description and interpretation, there have been work done on how to use it by representing it through several methods, although knowledge representation has the intention to describe the natural world and its richness according to Davis et al. [17], it can be specialized over certain subjects L. Gen, Bailing et al [20], did use a basic concept of knowledge representation to store attack patterns as dll extensions, so a complete attack pattern would be available for a certain tool; the approach used to store the attack knowledge was to craft the attack patterns, by hand, and later on use the complete attack pattern at once, the attack parameters must be selected by user at runtime being the final user the expert responsible for the attack to succeed by getting the right parameters, this kind of representation does not allow certain patterns to be flexible, as if there is another use for the same attack, a new .dll will be needed to include this new pattern, leading to redundant storage and narrowing the flexibility of the design. On the other hand, a more flexi-

ble solution for knowledge representation is by a knowledge graph, in which knowledge is stored as the relation between nodes which represent nouns, by axis representing verbs relating the nouns, this representation aims to be extensible and easy to understand, as knowledge gets stored as natural language phrases, this was explained by Nickel et al, to be used alongside machine learning techniques [25]. Although it is a good approach to represent knowledge, it is technically complicated to create the knowledge graph, for instance, one node, as any noun, could be linked to many other nouns on different circumstances, moreover, if it is not linked to any particular node it can be assumed that there could be a relation, that was just not stored at that particular knowledge graph, this is called an open world assumption. The interpretation and crafting of knowledge graphs depends highly on the kind of problem it is aimed to solve.

According to Nickel et al. [25], there exists several open projects of general knowledge graphs, created either automatically, or curated by hand, as this is the main technical problem while creating a knowledge graph which depends on the method of creation and the description of the knowledge aimed to be represented, it can imply a bug effort to achieve a knowledge graph to be filled. The proposed design includes the knowledge representation as a knowledge graph, as stated earlier, it could be a complicated task. Nevertheless, it offers flexibility at knowledge storage, and, being represented as a natural language phrase, is much more intuitive for interpretation purposes. In addition, as a graph can be as complicated as the curator decides it; the design aims to store attack patterns in an intuitive yet general way, so it can be further extended to unknown or not previously considered attack patterns. To represent an attack pattern it is needed to describe, as in natural language, in a general form, considering that attacks may be performed over several ways, including many steps, here comes handy to impose some constraints to the knowledge graph; first of all, the extension, as sated earlier the scope aims to represent attack knowledge, nevertheless, attacks are not meant to be performed in an ordered way, as this is more an outlaw activity there are no general rules to perform an attack, in addition, for certain attacks some specialized knowledge is needed in the way of reverse engineering, analysis and reading comprehension.

Understanding that the scope gets limited to attack patterns on the follow criteria, there is no need of analysis or reverse engineering, as this complicated task still needs actual human interaction, the attack pattern is limited to get an unauthorized access, this means no further action is considered once

the unauthorized access is achieved, no privilege escalation is considered, as the attack concerns more about weak spots to access any organization assets, this also means that DOS attacks are not considered. Next, for the general description on attack patterns, it is considered on two specific cases, first, the configuration vulnerability, also called missconfiguration, is considered as elaborated multi stage attacks, including several steps to be executed with different parameters in each step; and the vulnerability exploits, related to an implementation vulnerability which aim to be single step over specific software version discovered through the enumeration stage.

For instance, the misconfiguration attack is proposed to be generally described as a: *Precondition Presenting a Misconfiguration which Cause the Attacks Steps Leading to an attack Outcome*, the graph representation of such description, as in a knowledge graph can be seen at figure 4.2.

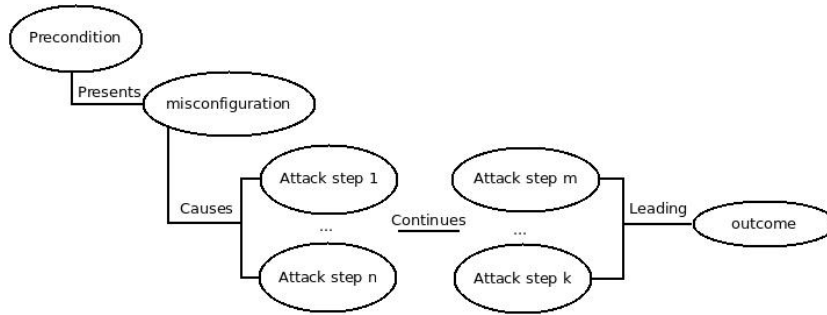


Figure 4.2: Knowledge graph of a general misconfiguration attack

This describes in a general way how misconfiguration attacks can be placed considering that such vulnerabilities may occur at any software version, on the other hand, the implementation vulnerability attacks are more specific on software version exposed to it, additionally, there could be a reported implementation vulnerability, but no available exploit which needs to be considered, because of this, an attack by implementation vulnerability is proposed to be described as a *Precondition matching a CPE showing a CVE vulnerability that Has an Exploit Leading to an attack Outcome*. Such proposal on both attack kinds considered aims to correspond with the attacks at hand for the evaluation proposal to be fulfilled. The graph representation of such description on a general implementation vulnerability attack can be seen at figure 4.3.

As stated, it is needed to include up to date information about implemen-

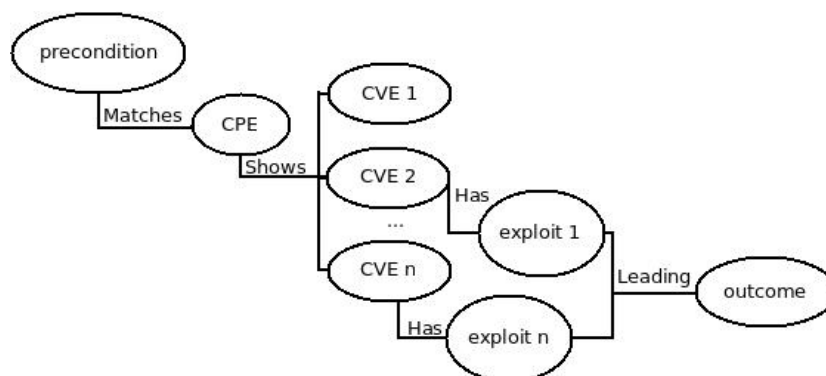


Figure 4.3:

tation vulnerabilities to be retrieved and related against the system under testing, such information is retrieved from the National Institute of Standards and Technology (NIST) from the U.S. department of commerce through the Common Vulnerabilities and Exposures (CVE) feeds, which concerns to and are updated in a constant basis by the same institution.

The second version of this feeds includes the list of all software showing vulnerabilities, giving the chance to retrieve such information, and correlate, more precisely on the enumeration findings, to accomplish this, the use of the CPE, which stands for *Common Platform Enumeration*, a standardized way to describe software and hardware by version and release, comes useful, the use of CPE to match vulnerabilities follows the standards of NIST on CVE data feeds, which allow having recent information about public vulnerabilities, and disclose such information by CPE, as a NIST standard, nevertheless, it is also possible to match information such as product name and version, to match different sources; having in mind that the main vulnerability source would be the NIST CVE data feeds.

Once considering the description as stated before, the attack patterns can be described in such way to craft the knowledge graph, an example of this graph, abridged, can be seen at figure 4.4, it includes the patterns by misconfiguration and implementation vulnerabilities exploits; as the scope states, the attack pattern must look for an unauthorized access, therefore, the outcome is considered as a remote terminal, in the figure can be seen that some attacks can be performed for several misconfiguration as well as the fact that an exploit for an implementation vulnerability could affect several version of the software.

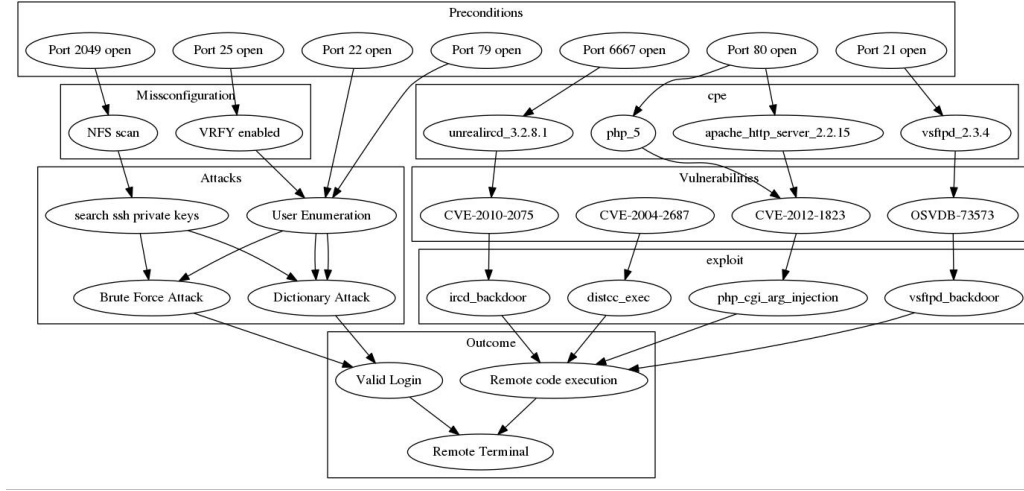


Figure 4.4: Knowledge graph on attack patterns

The knowledge graph is intended to comply with the attack pattern description proposed above, to include several attack patterns from configuration and implementation vulnerabilities to include such patterns at the actual test, it can be seen that the graph can be separated by clusters, such as attacks, vulnerabilities CPE, etc, giving a chance to grow the graph from already known circumstances to develop new attack patterns. It is intended that this graph can be expanded, by collaborative effort, being intuitive to expand as by describing any attack, in the terms already mentioned in natural language terms.

4.2.3 Search method

The main part of the penetration testing process, is to actually try an attack to verify if it can be exploited against certain system, to do so, several specialized tools are used by security experts during the pentesting process; understanding that each attack can be challenging in terms of analysis or reverse engineering, several tools, for different scopes are needed during the process, including forensic and reverse engineering tools; for the purpose of this work, as stated before, it is not considered the analysis or further exploitation, but only instantiation on specific attacks over a network. Nmap

[19] offers flexibility to instantiate elaborated attacks, including dictionary attacks and user enumeration, as shown by Bailing et al[20], nevertheless, it is also needed to include the parameter choosing on each suitable attack, to test it at the system being evaluated, done in an automated fashion. The test are meant to handle changes, either at the system under testing considered internal, such as updates, or at the knowledge database, as external changes such as new vulnerabilities disclosure; to accomplish such goal it is needed to persist both, at the enumeration and the attack evaluation, testing both continuously at the system being evaluated.

Evolution approach

Srivastava et al [26], worked on security test for software, using an evolution approach, by a genetic algorithm, this is, software inputs were altered, by genetic mechanisms, and tested over the software flow graph, in which, the insecure paths were marked, this allowed the testers to find suitable inputs for insecure behaviors on the software. In general evolution heuristics deal with the problem of finding most suitable combination by given outcomes, as a minimization iterative problem; on the same way, this kind of heuristics can be used to search on a changing search space, in such case, the new minimal will be approximated over the algorithm long term execution. In the same way of thinking, the Particle Swarm Optimization (PSO) algorithm comes handy, as it is simple and complies with evolution heuristics, by testing iteratively a collection of candidate solutions, called population of individuals, by mutating each one individually, and keeping a global best record (gbest), on each individual best result during the algorithm execution, if the result improves, the global best record gets updated, any other case, it is just skipped, each iteration on this algorithm is called a generation, this means that each possible solution, called an individual, was tested once. Figure 4.5 shows the general flowchart of the particle swarm optimization algorithm.

Fitness function

For evolution heuristics, it is very important the result on the problem tried to be solved, as this represents what the algorithm is looking for, for evolution heuristics, this is called the fitness function, for instance evolution heuristics

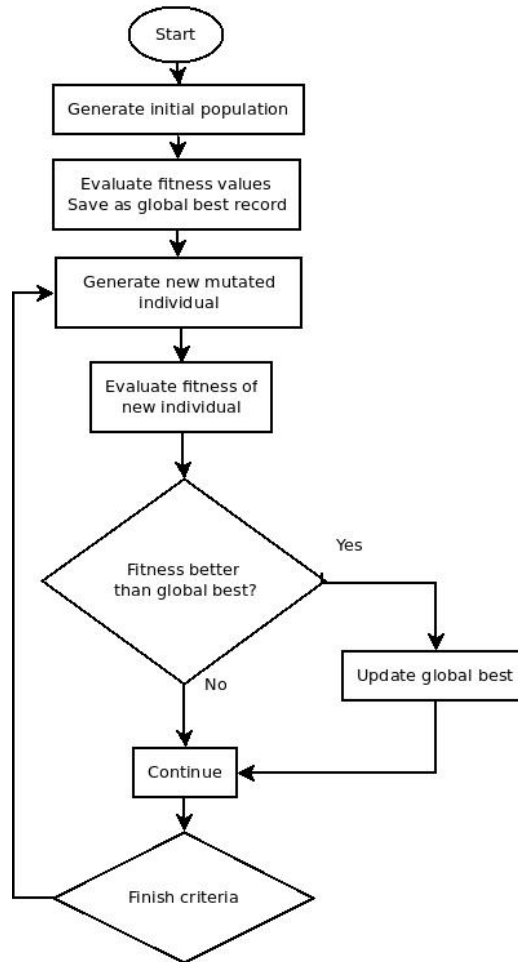


Figure 4.5: PSO gbest algorithm

are commonly used in minimization problems difficult to calculate by standard methods, the main technical problem at using an evolution strategy is the usage of a *fitness function* which allows the quantification of the problem variable aimed to be optimized.

For this design, the function is actually related to attack patterns previously determined to be suitable to the system under testing and its feasibility to be exploited at it considering the following; as an attack can take multiple steps to be performed, and those steps are meant to achieve an unauthorized access, the degree of advance over such path must be represented, and as

stated before, the main considered goal is to achieve an unauthorized access as a remote terminal, for which all paths must have the same goal, and therefore, a maximum common result. With this in mind, it is proposed a way to calculate the fitness function on each attack pattern shown in equation (4.1).

$$fitness = (Nsuccess)/(Ntot) \quad (4.1)$$

where $Nsuccess$, represents the number of successfully performed steps over an attack pattern on the system under testing, and $Ntot$ describe the total number of steps of the attack pattern tested.

As an example, lets consider the following attack pattern:

precondition-> user enumeration-> dictionary attack-> unauthorized access

Which includes 4 steps, as $Ntot$, and, assuming, that the dictionary attack could not be performed successfully, but previous step could, only the first two steps on this attack pattern would be exploitable; so the fitness on this example is presented in equation (4.2).

$$\begin{aligned} fitness &= 2/4 \\ fitness &= 0.5 \end{aligned} \quad (4.2)$$

Indicating that over such particular attack pattern, just half of the necessary steps, to gain an unauthorized access, can be placed, at the system under testing.

As this is an exhaustive search method, it will try eventually, all possible combinations on candidate solutions, the main purpose for this heuristics is to deal with huge search spaces. For this design purposes, the search space is actually quite limited on the candidate solutions, which are the attack themselves, on the other hand, considering every possible change available either at knowledge database or the system under testing, the search space is actually quite huge considering the time as a variable to fit the attack within time on the suitable change at the right time.

4.2.4 Design Integration

Having considered the three main aspects to be fulfilled on the design, next it is presented the complete design, in which each main phase interacts as

follows; first of all, the enumeration stage gathers information about the system under testing, as stated above, the enumeration must be performed repeatedly, and retrieve information related to services running at the system being evaluated the specific software version for such services as well as the kind of service interpreted as the port number opened, this is later related to information stored in the knowledge graph, as preconditions of an attack pattern that could be suitable for the system being evaluated relating also software versions through CPE (Common Platform Enumeration) information retrieved from enumeration.

Once it is related, attack patterns, suitable for the system under testing, are generated and passed to the evolution strategy algorithm, to be tested over large periods of time and among changes, in fact, there are two major actions occurring simultaneously, the enumeration process, which gathers information and correlates changes at the knowledge database, derived from knowledge representation as stated before, to form the attack patterns to be tested in concordance with such knowledge information; and the evolution strategy algorithm, which tests the attack patterns and modify its parameters aimed to get the best result on each attack pattern.

The complete flowchart on the design is presented in figure 4.6, enumeration stage and the evolution strategy algorithm are separated yet related process, each one executed individually, but associated by the enumeration discovery, as this is the main issue to determine information for which attack patterns would be suitable for the system under testing, according to the enumeration performed.

The integrated design intends to be representative on the behavior that a persistent attacker could met while trying to achieve an unauthorized access against a particular system.

During current chapter the details on proposal design to achieve a persistent intrusive evaluation was addressed, including the corresponding stages such as enumeration and search method through evolution strategy, from such design it can be recalled the main subjects brought to mind by the related work, also it is needed to consider the design application as black box testing which comes at hand when talking about the knowledge representation, to overcome the generic use of this design on security evaluation.

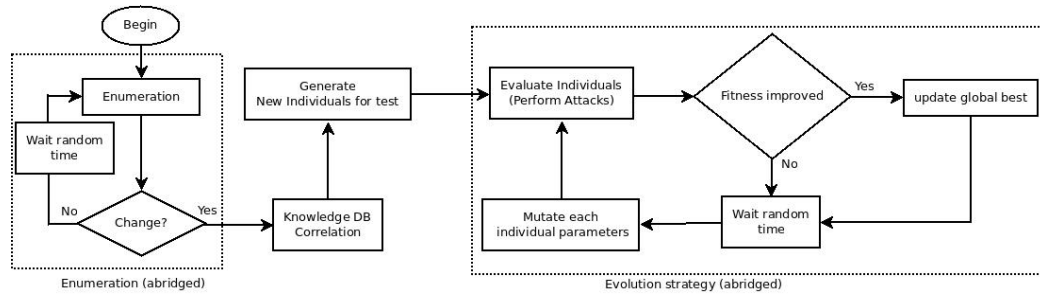


Figure 4.6: Design integration flowchart

For next chapter a case of study is addressed for implementing the evaluation on a selected scenario to determine the results on persistence that the design aims to achieve.

Chapter 5

Case of Study

In the current chapter a case of study on a brief development to achieve the persistent intrusive evaluation, based on the proposed design to achieve it is presented as well as the details on code integration in addition to the data and tools employed. The development was performed using Python version 2.7, the tools at hand are NMAP[19] for enumeration, and Metasploit framework[27] version 4.12 for attack injection.

5.1 Code Development

5.1.1 Knowledge Database

By this point, it must be clear that, knowledge graph approach, aimed to describe attack patterns, goes under graph data structure capabilities, understanding that, by design, there will be just a couple of queries from time to time over the data structure, since it is more meant as a knowledge database, provided by the knowledge graph description process creation. In this way, the knowledge storage is performed in a traditional relational database, instead of an adjacent matrix, or a linked list, as this graph representations are more meant to perform several operations over the graph as a whole, and, for this design purpose, it is just needed to store the knowledge, and to query it from time to time; in this manner, knowledge database gets constructed and stored as a relational database over a file, and queried when required, being able to be modified at any time, leaving the concurrence over writing and reading to the DB management system understanding that the

most common operation would be the query, keeping in mind that no more concurrence beyond the enumeration would be needed. Therefore, in figure 5.1 it is presented the entity relation model of the knowledge database, constructed based on the knowledge graph representation described above; as stated earlier, the main goal here is to represent the knowledge in an intuitive yet general way, and to be able to query it. For the purpose of the current work, the only outcome considered is the unauthorized access, so it will not be considered during querying the database.

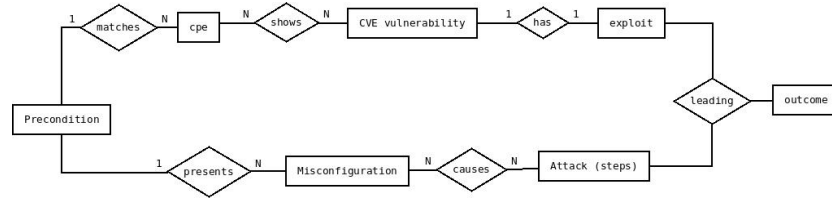


Figure 5.1: Entity Relation model for knowledge database

The knowledge database is stored using the sqlite database management system, as this does not require a running server instance or further configurations, it offers the SQL functionality with few limitations and flexibility to implementation, this allows the instrument to be easily implemented. An example of the configuration vulnerability attack patterns stored at the knowledge database can be seen at the table 5.1 the information does include the command to execute the actual attack, at the Metasploit framework, as well as the next step on the attack pattern; it can be noticed that intermediate steps can be shared by different attack patterns.

Table 5.1: Configuration Vulnerability attack patterns

Configuration Vulnerability	port	Attack name	Command module	Next Step	Score
VERFY enabled	25	User enumeration smtp	auxiliary/scanner/smtp/smtp_enum	Dictionary attack	10
Rservices enabled	513	Rlogin	auxiliary/scanner/rservices/rlogin_login		12
Rexec enabled	512	Rexec	auxiliary/scanner/rservices/rexec_login		12
Rsh login	514	Rsh login	auxiliary/scanner/rservices/rsh_login		12
Finger	79	Finger enumeration	auxiliary/scanner/finger/finger_users	Dictionary attack	10
Java rmi	1099	Java rmi	auxiliary/scanner/misc/java_rmi_server	Java rmi exploit	40
NFS scan	2049	NFS scan	auxiliary/scanner/nfs/nfsmount	search_private_ssh_keys	12

As the attack patterns also consider the implementation vulnerability, the knowledge on how to execute such attacks, and the specific software version

and CPE, the information on such attack patterns can be seen at table 5.2, it is noticed that some vulnerabilities does not include exploit information, as it is not available by the tool. In addition, there can be noticed that the same vulnerability may apply to different software versions.

Table 5.2: Implementation Vulnerability attack patterns

CVE	product	version	port	exploit	cpe
CVE-2004-2687	distccd	v1	3632	exploit/unix/misc/distcc_exec	cpe:/a:apple:xcode:1.5
CVE-2004-2687	distccd	v1	3632	exploit/unix/misc/distcc_exec	cpe:/a:samba:samba:2.18.3
CVE-2007-2446	Samba smbd	3.X	139	exploit/linux/samba/lsa_transnames_heap	cpe:/a:samba:samba:3
CVE-2007-2447	Samba smbd	3.X	139	exploit/multi/samba/usermap_script	cpe:/a:samba:samba:3
CVE-2007-2447	Samba smbd	3.X	139	exploit/multi/samba/usermap_script	cpe:/a:samba:samba:3.0.0
CVE-2007-2447	Samba smbd	3.X	139	exploit/multi/samba/usermap_script	cpe:/a:samba:samba:3.0.4.rc1
CVE-2010-2063	Samba smbd	3.X	139		cpe:/a:samba:samba:3
CVE-2010-2075	Unreal ircd		6667	exploit/unix/irc/unreal_ircd_3281_backdoor	cpe:/a:unrealircd:unrealircd:3.2.8.1

On the other hand, the parameter control works by the specific parameters met at each attack pattern, such parameters must include the necessary information for the attack to be executed, and the acceptable values by the attack injection tool, in the current work it is used Metasploit, an example on such parameters for the CVE-2007-2447 can be seen at table 5.3, it can be noticed that each parameter can be select within a range of values, also included at the knowledge database, such values are meant to be selected each execution by the evolution strategy algorithm. According to the design,

Table 5.3: Parameter example on CVE-2007-2447

Parameter	Min Value	Max Value
ConnectTimeout	5	15
SMB::VerifySignature	0	1
SMB::VerifySignature	0	1
SMB::ChunkSize	400	600
SMB::VerifySignature	0	1
SMB::VerifySignature	0	1

the knowledge DB needs a feed to retrieve new vulnerability entries disclosed and published by NIST as CVEs, these are commonly updated and retrieved as XML files from the NIST data feed[28], from which it can be acquired such information about new vulnerabilities, nevertheless, the information about available working exploits for such vulnerabilities must be curated manu-

ally, as it relies on the Metasploit framework available information, which is limited; this can be a subject for further work additions.

5.1.2 Enumeration

The proposed design does consider the enumeration to retrieve information from the system at evaluation, to do so it is needed to consider the appropriate tools to achieve such development, as stated previously, the pentesting process rely on some specialized tools, for each stage of the test, for instance as recommended on the document by Chan Wai [3] from SANS institute, NMAP [19], includes several, if not all, the previously related methods to gather information about a certain target, from which it can be mentioned the running services at the given system that can be interpreted from the ports open at it and the software versions needed to recall implementation vulnerabilities, such information needs to be compared against the stored knowledge to retrieve the suitable attack patterns and its parameters, the developed code does the request for the tool and retrieve the information, to correlate the suitable attack patterns to the knowledge DB, the development pseudocode can be seen at listing 5.1.

```

1  class Recon(threading.Thread):
2      def run(self):
3          while True:
4              Execute NMAP enumeration
5              for service in target:
6                  Determine suitable attack patterns
7                  #including implementation and configuration vulnerability
8                  if Attack patterns not in Current tested:
9                      Acquire flag
10                     Update Current Tested
11                     Release flag
12                     Wait pseudo random time
13                     if less services found:
14                         Increase delay randomly
15                     else:
16                         Wait pseudo random time

```

Listing 5.1: Enumeration code example

The connection with the NMAP tool, is achieved by the python library correspondent to the NMAP project, this allows the instrument to both, perform enumeration requests, and to receive detailed information about

a particular target, the correlation with the stored knowledge previously presented will bring up the individuals to test at the evolution strategy search method.

The enumeration is executed as a thread related to evolution strategy by the flag as a semaphore and the current tested attack patterns to determine those not being tested by the evolution strategy at a given time and being able to include it at the search method.

The correlation is pretty straightforward from the enumeration information gathered, from services name versions and CPE information related to the preconditions stored at the knowledge database as presented earlier. An example of the disclosed information by enumeration can be seen at figure 5.2, it can be seen that enumeration gets enough information to determine the suitable attack patterns according to the data stored in the knowledge database.

```
{21: {'conf': '10',
      'cpe': 'cpe:/a:vsftpd:vsftpd:2.3.4',
      'extrainfo': '',
      'name': 'ftp',
      'product': 'vsftpd',
      'reason': 'syn-ack',
      'state': 'open',
      'version': '2.3.4'},
 22: {'conf': '10',
      'cpe': 'cpe:/o:linux:linux_kernel',
      'extrainfo': 'protocol 2.0',
      'name': 'ssh',
      'product': 'OpenSSH',
      'reason': 'syn-ack',
      'state': 'open',
      'version': '4.7p1 Debian 8ubuntu1'},
 23: {'conf': '10',
      'cpe': 'cpe:/o:linux:linux_kernel'}}
```

Figure 5.2: Enumeration results example

5.1.3 Evolution strategy

The main aspect about the evolution strategy in the fitness function calculation, as stated in the design, it is totally related to the successfulness of a certain attack pattern, such information can be retrieved and interpreted from the Metasploit framework console output once the attack pattern step

gets tested by it; the attack step parameters are stored alongside the knowledge information in the DB as explained earlier, so it can be retrieved at each time and mutated within the limits stated at the knowledge DB, once the attacks are tested, each result is stored in the results DB for further analysis as the actual result of a given attack pattern at a certain time against the target system. The pseudocode on the evolution strategy development code is shown at listing 5.2, the current attack patterns are updated from enumeration, and the evolution strategy runs as a separated thread communicated through global variables for attack patterns and a flag as a semaphore to update the attacks if needed.

```

1  class Attack_test():
2      Wait first enumeration results
3      Connect to Metasploit Framework
4      while True:
5          Obtain current attack patterns
6          if flag not acquired:
7              for attack pattern in Current tested:
8                  Mutate attack parameters
9                  Send attack pattern to Metasploit framework
10                 Calculate attack pattern fitness
11                 if new fitness > Current tested:
12                     Update Global best
13                 elif new fitness < Current tested fitness:
14                     Increase worsen counter
15                 if worsen counter > treshold:
16                     Update Global best
17             Store Results on DB
18             Wait pseudo random time
19         else:
20             #if enumeration finds a new suitable attack vector
21                 Acquire flag
22                 Update Current Tested Attack patterns
23                 Release flag

```

Listing 5.2: Evolution strategy pseudocode (abridged)

The evolution strategy does implement pauses between each generation, in the understanding that, changes will occur within large periods of time, the aim of the search method is to maintain a low profile interaction, yet it does persist in the search of an exploitable vulnerability, either at the enumeration or the exploitation stage.

To actually exploit the attacks, it is used the Metasploit Framework connection library for python, this allows the interaction with the tool to perform

the attacks and retrieve the outcome of such attacks, bringing the ability to determine if it succeed or not and for fitness calculation. The commands to execute at each particular attack are included as part of each individual retrieved from knowledge database as explained earlier, as well as the parameters prone to it, which are mutated during the evolution strategy execution.

As a result of the search method the attack steps, the parameters used, the start and finish time as well as the attack execution outcome are retrieved as part of the current generation attacks tested, an example of it can be seen at figure 5.3, it can be noticed that the actual code string tested at the Metasploit framework is included as well as the parameter values, the fitness, outcome and needed information for the evolution strategy execution.

```
[[u'auxiliary/scanner/smtp/smtp_enum', [[u'UNIXONLY', None, 0, 1, 0, -2]],
[u'auxiliary/scanner/ssh/ssh_login',
[u'USER_FILE', u'Dictionaries/us', 1, 2, 1, 0],
[u'BLANK_PASSWORDS', None, 0, 1, 0, 0],
[u'THREADS', None, 1, 50, 1, -74],
[u'USER_AS_PASS', None, 0, 1, 0, 0],
[u'PASS_FILE', u'Dictionaries/cpas', 1, 2, 1, 0]],
1.0,
'2016-12-09 15:57:30.677568',
'2016-12-09 16:02:48.989177',
[[[+] 192.168.172.137:25 - 192.168.172.137:25 Users found: , backup, bin, daemon, distccd, ftp, games, gnats, irc, libuu
d, list, lp, mail, man, news, nobody, postgres, postmaster, proxy, service, sshd, sync, sys, syslog, user, uuwp, www-data'],
["[+] SSH - Success: 'sys:batman' 'uid=3(sys) gid=3(sys) groups=3(sys) Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr
10 13:58:00 UTC 2008 i686 GNU/Linux "],
"[+] SSH - Success: 'klog:123456789' 'Could not chdir to home directory /home/klog: No such file or directory Could not chd
ir to home directory /home/klog: No such file or directory "]]],
0]
```

Figure 5.3: Evolution strategy attack pattern execution example

On the current chapter the development code for a case of study on the proposed design as a developed instrument was addressed to determine the feasibility of it to be implemented and perform evaluation on a persistent manner; it is needed to consider the limitations on the developed instrument as the selected tools are mean to be used over a limited scenario, and the knowledge graph was manually curated bringing boundaries to the attacks the developed instrument would perform, nevertheless the behavior , as persistence may be achieved to determine the design feasibility on such long term evaluation.

Within next chapter, the detailed test scenario as well as the results achieved would be presented as well as some analysis on such results.

Chapter 6

Tests and Results

In the present chapter it is presented a test scenario for the case of study developed code from previous chapter, considering the design and case of study code to achieve a persistent intrusive evaluation, it is needed to consider a test scenario related to the considerations made for the evaluation to be persistent, the details on such scenario and the results obtained are shown.

6.1 Test Scenario

Having considered the details about the design, it is needed to plan how to test if the design and its developed code would actually work on a functional environment, for this, it is considered that the main issue about the design is to actually perform attacks over a system, as a matter of fact, it is needed to test a system, previously known to be vulnerable, understanding that prior to performing the attacks it is needed to know if certain attack is actually exploitable, to determine if the design does actually find and exploit successfully a suitable attack and gather the correct outcome of it. Lets remember that it does propose a sort of automated pentesting in a black box exercise for this, the scenario must be well known to overcome the results as successful or not during each generation execution given that no information about the target is available to the developed instrument, and it just relies on the knowledge DB and the information disclosed by the enumeration stage of the instrument.

As being part of a pentesting, it is convenient to think about known scenarios, this is, systems vulnerable by design, which allow penetration testers

to try their skills, or to follow established steps as in "capture the flag" exercises, this publicly known systems can help to know whether the design would place and exploit a suitable attack pattern, considering, it is already reported to be possible to do it. For the current work purposes, three of these systems were chosen as testing scenarios, the details on such scenarios and vulnerable OS distributions is presented next.

6.1.1 Expected Results

The current work is intended to come up with the design and proof of concept concerning an instrument to perform automated penetration testing by means of stored knowledge, including the actual examination of the attack and its corresponding results as a manner of long term security assessment in an active fashion, including new vulnerability disclosure as well as misconfiguration and some other attacks, showing the exploitability changes under variable conditions of the target system, such as updates or new vulnerabilities disclosed by trusted organisms.

6.1.2 Target selection

Due to the nature of a penetration test, a system and its information can get compromised, it is performed in agreement with system administrators, regarding of the confidential information that can be disclosed in the process, understanding this, it is clear that a common approach for pen-testers to practice, is to have popular distributions prone to vulnerabilities, in such fashion, several OS distributions are available for testing as virtual machines. For instance several of this vulnerable OS distributions include not only software vulnerabilities, but also, vulnerable configurations on known services, in fact, as this OS distributions are prone to be exploited, there are several exploitable attack patterns over it. On the current work three examples of this vulnerable OS distributions are considered given that such OS distributions represent well reported vulnerability exploitation and different representative case of the available options.

Metasploitable, Vulnix, and Fristileaks, include wrongly configured services, as well as software vulnerabilities, in the case of Fristileaks it represents a "capture the flag" exercise, for which some analysis is needed to exploit the presented vulnerabilities, on the other hand, some suitable attacks, over these vulnerable distributions, do need some analysis or reverse engineering to be

exploited and, as stated earlier, such attack patterns are not considered for the current work, in such case, the identifiable and prone to be tested, attack patterns over each vulnerable OS distributions, are presented in table 6.1.

Table 6.1: Reported attack patterns by selected vulnerable OS distribution

Target	Known Attack Vectors (included on Knowledge DB)	Search spaces
Fristileaks	Web admin enumeration -> Dictionary Attack -> Shell PHP	Threads, Dictionaries, Attempts, Time
Vulnix	User enumeration (smtp, finger) -> Dictionary attack over SSH	Threads, Dictionaries, Attempts, Time
	User enumeration (smtp, finger) -> Dictionary attack over rlogin	Threads, Dictionaries, Attempts, Time
	Mount smb share -> Generate ssh keys -> save .ssh authorized key -> SSH login	Time
Metasploitable	User enumeration (smtp, finger) -> Dictionary attack over SSH	Threads, Dictionaries, Attempts, Time
	User enumeration (smtp, finger) -> Dictionary attack over rlogin	Threads, Dictionaries, Attempts, Time
	Mount smb share -> Generate ssh keys -> save .ssh authorized key -> SSH login	Time
	Telnet port 1524 -> Backdoor login	Time
	Metasploit exploit modules -> remote terminal	Threads, Dictionaries, Attempts, Time

Here it is referred as search space the applicable parameters on each attack pattern, over which the evolution strategy algorithm can search in addition to the time and changes over the system or the knowledge database, such behavior is interpreted as persistence in terms of the definition at use by the current work[5].

The main features for each OS distribution are described as follows. Fristileaks consists in a capture the flag scenario, this allows the practice of analysis and reverse engineering over vulnerable configurations and implementation at a web page, the "capture the flag" exercise implies that there is a way to intrude such target, but the vulnerabilities to do so, rely on some human interpretation of code comments and analysis. The Fristileaks distro allows testing the instrument on a known vulnerable target, yet complex to attack, it is a representative, well documented example of a "capture the flag" exercise, it is expected that the instrument development obtain poor results at the intrusion as this requires some analysis beyond the reach of the knowledge representation here proposed. This aims to represent a portion of the available vulnerable distributions, prone to be attacked, and the actual potential vulnerability disclosure limitations on the proposed instrument.

The Vulnix distribution is a well documented example of vulnerable distribution to perform testing, in this is represented some configuration and implementation vulnerabilities. The aim is to determine the successfulness on the developed instrument to both, identify and exploit such vulnerabilities, this distribution does not allow any change or sort of administration, so this can be seen as a fixed scenario, on the target side.

The Metasploitable distribution includes several configuration and implementation vulnerabilities, this is also well documented by its developers

and it does allow all sort of changes and administration. In this case it is aimed to serve as a live scenario, to perform both, changes at the knowledge DB, understood as external threats, such as vulnerability disclosure, and to perform corrective actions, to allow the system to determine the changing conditions on the security at this target over a period of time. The main features for each distribution can be seen at table 6.2

Table 6.2: Vulnerable Distributions

Distro	Features	Fixed
Fristileaks	Capture the flag, needed of some analysis	Yes
Vulnix	Multiple implementation and configuration vulnerabilities	Yes
Metasploitable	Multiple implementation and configuration vulnerabilities	No

As the system under testing will be each of the vulnerable OS, it is implicit that, the developed instrument would have a way to reach the system under testing; because of that, the network topology is not considered for the testing.

6.1.3 Results scoring

With a view to obtain results easily understandable, it is needed to implement a scoring to the successful results, as different attack patterns may not represent the same impact on the system under testing, in terms of scoring several considerations can be done about the attack itself, according to Mark Mateski et al. in a work by Sandia Report [29], some aspects of the threat can be useful to measure the potential impact on the system under testing, as for giving a score based on how can those vulnerabilities actually be exploited and what is needed for a person to accomplish it, for instance, it can be recalled that the most important condition about threat metrics can be narrowed to the time it takes to accomplish a successful attack attempt, the needed knowledge from an attacker, and the stealth to carry on the attempt. Taking into account such factors, the scoring gets constructed for each attack, in the mentioned terms, to determine the security level at a given time as a report for further analysis and to take corrective measures according to administrators criteria. The scoring table can be seen at table 6.3. The attacks scoring on each attack attempt is determined by the average on each attack individual steps scoring and the total number of steps as shown in equation (6.1).

Table 6.3: Attack scoring

Time	score	Knowledge	score	Intensity	score	Stealth (delay)	score
<i>Months</i>	1	<i>Specilized</i>	1	<i>High</i>	1	<i>High</i>	1
<i>Weeks</i>	3	<i>Technical advanced</i>	3	<i>Medium</i>	3	<i>Mediium</i>	3
<i>Days</i>	5	<i>Technical basic</i>	5	<i>Low</i>	5	<i>Low</i>	5
<i>Hours or less</i>	25	<i>Basic/none</i>	25	<i>One shot</i>	25	<i>None</i>	25

$$\frac{\sum_{i=1}^n (Sc_t + Sc_K + Sc_I + Sc_{St})}{n} \quad (6.1)$$

Where n is the total number of steps on each attack, and Sc_t, Sc_K, Sc_I and Sc_{St} are the score of each attack step on terms of time, knowledge, Intensity and Stealth, each score is the sum of each category according to the step's qualitative categorization. For example, an attack pattern carried within hours, needed of basic knowledge, with low level of intensity, understood as few requests needed to accomplish the attack, and no stealth needed would be scored as presented in equation (6.2).

$$\begin{aligned} score &= 25 + 25 + 5 + 25 \\ score &= 80 \end{aligned} \quad (6.2)$$

6.2 Testing experiments

Having proposed a way to represent and store attack patterns knowledge and later a search method to test them, there are two main issues to test at this design, first of all, the attack pattern actual exploitability, this is, that it gets exploited and that it could be retrieved actual true information about its success or failure on the system under testing; and, in the other hand, whether or not the design will include changes, either at the knowledge database, or at the system under testing, corresponding to an exploitability change on an attack pattern or a new one not previously tested. Because of this, there are some experiments of interest.

At first we have the vulnerable distributions, prone to be attacked, as the developed instrument is intended for black box testing, it does not include knowledge or details about any target in particular, for instance, it just includes generic knowledge similar to attackers knowledge to be implemented during an actual attack, for this, it is first necessary to sum up the results in

terms of how did the instrument performed at discovering possible vulnerabilities and its later exploitation, in this case it was referred to developers information, on each distribution case, to rely on the available vulnerabilities according to them, the number of vulnerabilities discovered for each distribution is shown on table 6.4.

Table 6.4: Number of vulnerabilities discovered for each OS distribution

Distro	Exploitable vulnerabilities (reported)	Potential vulnerabilities found	Vulnerabilities exploited (avg)
Fristileaks [30]	2	1	0
Vulnix [31]	4	6	1.5
Metasploitable [32]	6	11	6

As expected, the Fristileaks distribution requires actual reverse engineering, and further analysis, which is out of the scope on the knowledge representation postulated, therefore, it is expected the no exploitation available for such distribution, proving the knowledge boundaries on the instrument developed. On the other hand, the Vulnix and Metasploitable distributions offer either implementation and configuration vulnerabilities, because of this, there are several not reported by their developers, vulnerabilities were found as potentially exploitable, nevertheless, just some of them were actually exploited by the instrument, proving that some vulnerabilities need analysis or related knowledge not available at the instrument's knowledge DB.

According to the reported vulnerabilities the attack patterns applicable to it, the instrument did find and exploit those not needed of analysis or reverse engineering but not those needed of such capabilities not present at the design or the instrument developed, this behavior was expected, as the knowledge representation is not aimed to represent further more advanced knowledge needed for this, as analysis and reverse engineering are much more elaborated to be automated, nevertheless, the instrument includes vulnerabilities prone to real world and, as results show, it was able to determine and exploit most of them, automatically, without any knowledge from the actual target, such results are presented later for each case. The show vulnerability assessment done automatically, which can be understood as part of the pen-testing process, for the next, the persistence is tested as part of the complete design and developed instrument, which is accomplished by testing changing scenarios as a real target may experience, this will be described as follows.

First of all, how the design will deal with changes at the knowledge database, as stated vulnerabilities can be made public at any time, and the proposed design must allow testing over this new vulnerability disclosures, next, it will need to cope with changes at the system under testing, this includes updates, or configuration upgrades to correct any vulnerability showed earlier; this has to be made while the testing is still in progress as the actual testing will take long time to be concluded; and both conditions tested over larger period of time, which aims to test if the exploitability and system availability are affected by this tests.

The tests will be presented as follows, first tests done by adding a vulnerability to the DB to be added for testing, later a vulnerability added that is not exploitable even when it is found suitable for the system under testing by the enumeration and knowledge DB correlation, and finally, the complete test regarding changes at both sides, the DB and later the system under testing for a large period of time. In each case, the exploitability of an attack pattern is implicit, as this attack patterns are considered from known reported attacks on the vulnerable OS distributions, and, furthermore, the fitness reported for each attack vector, correspond to each attack vector exploitability.

6.2.1 Testing case one

The first case test is to start the complete designed process, against a vulnerable OS distribution, and later add , to the knowledge database, a vulnerability and its corresponding exploit; it is expected that, as the enumeration process is running constantly, it will eventually find the new suitable attack pattern, and this will be added for testing at the evolution strategy algorithm. On this way, there was added a new vulnerability and its exploit for two distributions, as shown on figure 6.1.

As seen, the NFS scan, which corresponds to a vulnerability added to the knowledge database during the first generation of the evolution strategy algorithm, and it was added to it at 2nd generation with a fitness of 1 for that particular attack pattern, indicating that this vulnerability can be successfully exploited at the given system being tested, in this case the Vulnix

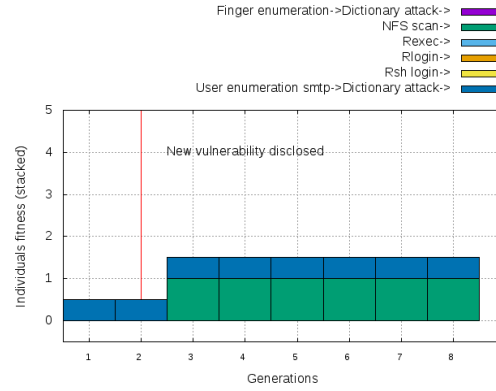


Figure 6.1: Attack pattern added to knowledge DB over Vulnix

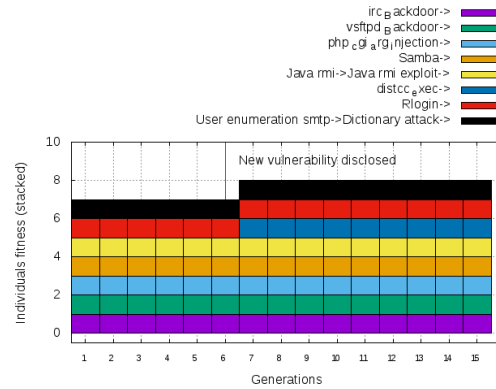


Figure 6.2: Attack pattern added to knowledge DB over Metasploitable

OS distribution. The test was also done over the Metasploitable distribution, in which, the attack pattern for a vulnerability and its exploit were added to the knowledge database, during the fifth generation of the evolution strategy algorithm, and it was added and tested by the sixth generation of it, this can be seen at the figure 6.2. At figure 6.3 can be seen that, as for this case the new vulnerability added its scored highly in terms described at table 6.3, the new attack pattern gets a higher threat metric, this is important to be considered on security awareness.

In both cases the vulnerability was added for testing a few generations after it was added to the knowledge database, this is the expected behavior on the design, nevertheless, this could represent a disadvantage, as any

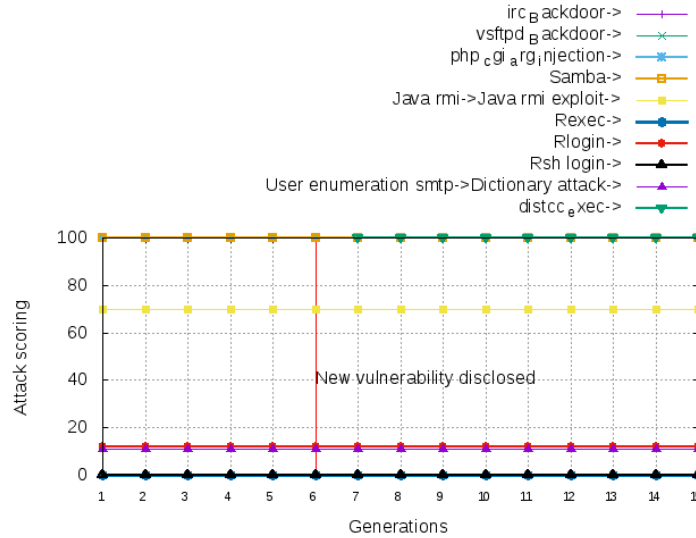


Figure 6.3: Attack scoring testing case one Metasploitable

vulnerability publicly exposed can be tried to be exploited as soon as it gets publicly known, nevertheless, it is considered that not all reported vulnerabilities could have a working exploit by the time they are made public. On the other hand, the scoring shows the same behavior, but, understanding that it does not get a full unauthorized access, it is only scored as information gathering as shown on figure 6.4.

6.2.2 Testing case two

For the second case, it is considered an update over the system under testing, not to add a new attack pattern, as this was partially considered on the first case, but to correct a configuration vulnerability to show the exploitability results changing over such corrective action, for this, the vulnerable SMTP configuration, prone to user enumeration, was corrected to avoid information gathering techniques, this was done only on the Metasploitable OS distribution, because this is the only selected OS distribution that allows system administration, understanding that the other distributions are just for a "capture the flag" exercise and not a wider solution to perform pentesting. For the Samba configuration correctness, it can be seen that the fitness of the attack pattern diminishes for the next generation of the evolution strategy

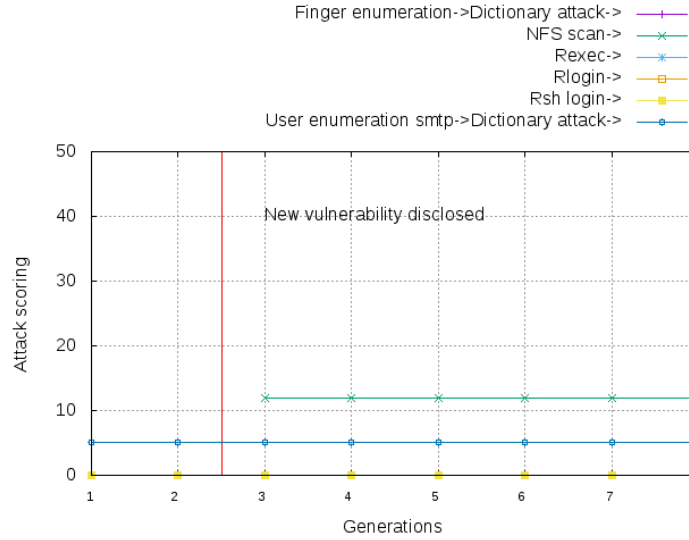


Figure 6.4: Attack scoring testing case one Vulnix

algorithm after the correction gets applied, nevertheless, as the algorithm just keeps record of the global best, meaning that if no improvement is got on fitness, the results would not be altered, the behavior is altered to give a penalty on the results, to accomplish this, it is considered a penalty for each fitness value minor to the global best, if three of this are reach, the global best has to be updated to reflect that the individual being tested met the worsen threshold on fitness, so it can recall the results over next generations, the test done on this kind of update are shown on figure 6.5, it can be seen that the Samba vulnerability gets unexploited by the sixth generation, in addition, the VSFTP service related exploit is not always exploitable, as it can be seen on the same figure.

In addition, after each attack pattern gets scored, it can be seen that not all attack patterns can be considered equally threatening for the system under testing, as some of them, require less effort and are executed successfully, the scoring on this test case can be seen in figure 6.6, it can be noticed that the vulnerability related attacks are better scored and, because of that, would need actions to overcome the threatening situation.

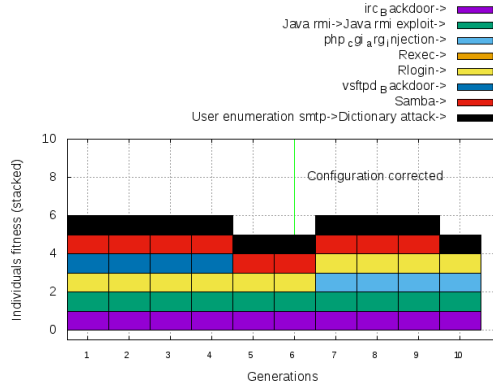


Figure 6.5: Configuration corrected

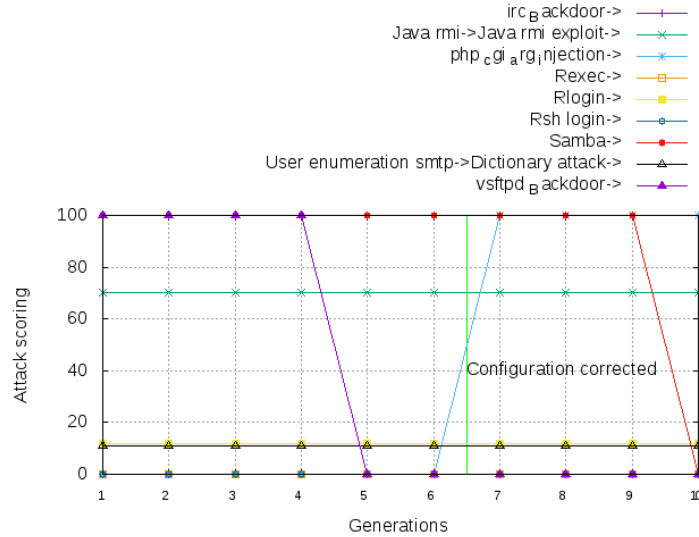


Figure 6.6: Attack scoring on testing case two

6.2.3 Testing case three

It is considered an example of the previous cases shown before over a larger number of generations and experiencing both changes at given times, for instance, it is first submitted an update to database, and later on, the SMTP misconfiguration gets corrected to verify the results on a composed scenario by a larger number of generations being able to carry on the attacks for a larger amount of time and having a reduced impact on the system tested.

It can not be assured that the tests will have no repercussion on the system tested, as it is in fact an attack, in fact, at this point it is expected some minor impact on the system, nevertheless, it is expected the the impact would not be significant for the system availability, in such case , it is contemplated to test it for larger periods of time to determine the system behavior at, first , not to induce a system outage, being it partial, or any service attacked, or to the whole system, and second to determine that exploitability results will hold for large periods of time, this means that the attack considerations can take place at any instance with no interaction from previously exploited attacks, being able to obtain realistic information of an attack exploitability each time.

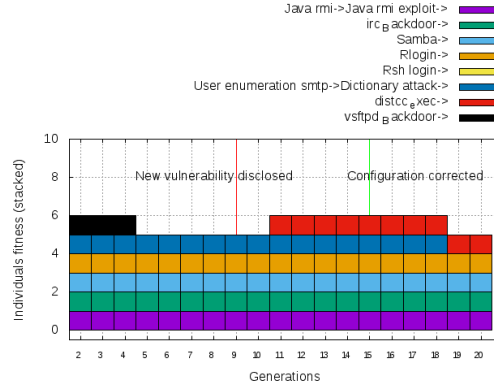


Figure 6.7: Testing case three attack pattern fitness

The results are shown on figure 6.7, it can be seen that attacks can still be exploited over larger number of generations, this can be understood as the service still being available, and attacks being exploitable independently from previous generations, it can also be noticed the SMTP configuration corrected during the last generations and the addition of the DISTCC service related exploit by the tenth generation, understanding that this results can be carried on for large periods of time, responding to changes and proving not to have a significant impact to the system under testing.

In this case it can be noticed that the vulnerability added gets a higher threat measurement impact on the target system as shown at figure 6.8, in

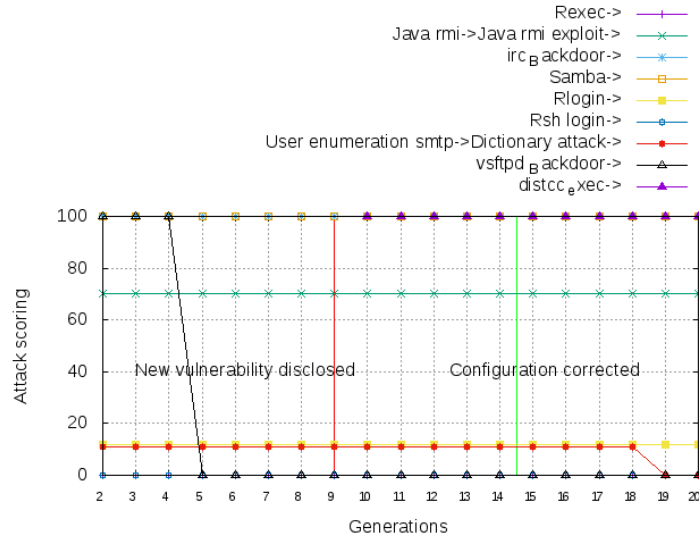


Figure 6.8: Attack scoring on test case three

addition, the misconfiguration correctness can be seen as a low level action, taking into account the scoring as presented in figure 6.8, this results show the importance of having a scoring system on the attacks tested, as some of them represent a higher threat in the terms mentioned at table 6.3, like the implementation vulnerability exploits such as the IRC Backdoor attack or the DISTCC related exploit.

6.2.4 Testing case four

For the last testing case, the intention is to show a more complex scenario, as it corresponds to the case where it can be seen the security assessment against a known vulnerability, over certain software version, moreover, as the version gets updated the vulnerability shall be corrected, however the security would only be assured for a limited time, as new vulnerabilities are discovered in a continuous basis, for instance, after few generations, new vulnerability over the updated version is added to the knowledge database, showing the common behavior of vulnerabilities and giving the idea of the importance of security assessment, as it is not the same security status as times goes by, and new vulnerabilities are discovered, and exploits are made public. For this test, the vulnerable software is samba on old releases, in this

case the 3.0.20 and the 3.0.26 to include already known working exploits on both version, aiming to simulate the response to a vulnerability correctness by an update, this shows the process from the vulnerability being added to the exploitability on the target system, and the eventual correction due to a system update. As seen in figure 6.9, the Samba related exploit does work, from its discovery, simulated by adding the information to the knowledge DB, to its correctness by a system update on samba version, nevertheless, the exploit does work again after few generations.

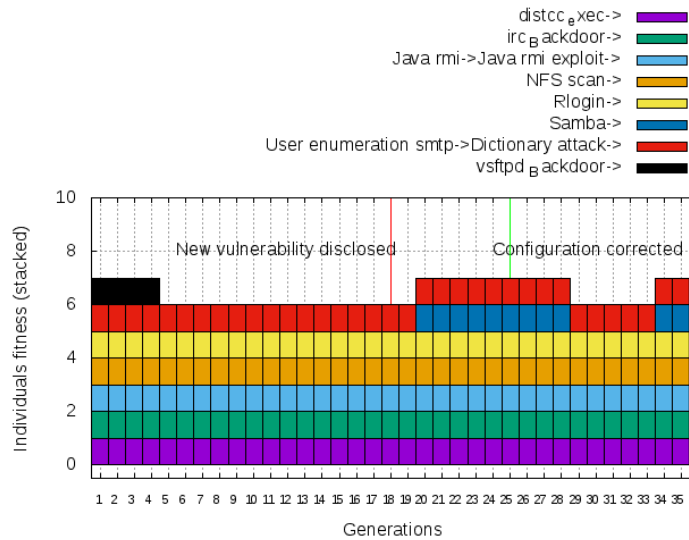


Figure 6.9: Vulnerable version update test

On the other hand, by scoring this kind of attacks it can be seen the major impact on the security of the given system, as it is easily exploitable and gets a higher score, however, the update does impide the attack to succeed for a period of time as shown on figure 6.9, after which it gets exploitable again, this changes can be seen on this test, understanding the importance of knowing that regular tests may not came up with the new exploitability after the update. To complete the results the attack scoring is evaluated for this scenario, it is noticed that the exploitability, even after the update gets a higher value, indicating the threat measurement for such vulnerability, such scoring is presented at figure 6.10.

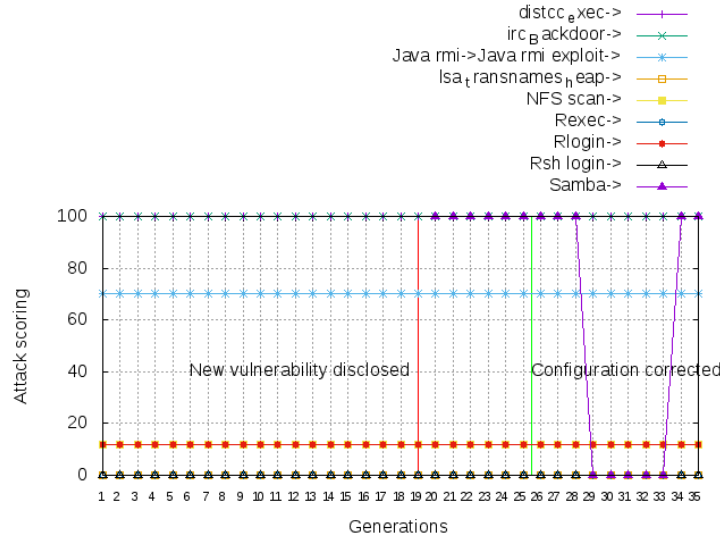


Figure 6.10: Attack scoring on testing case four

6.3 Result Analysis

As shown earlier, the results' main purpose is to improve the security awareness, and to reflect the changing conditions on system's susceptibility to attacks, on either corrective actions, or new discoveries on knowledge. The main results are in the changing conditions under different circumstances for the previously mentioned selected vulnerable distributions, according to the limitations and vulnerabilities prone to each vulnerable target.

Knowledge DB update

For the first testing case the susceptible targets tested were the Metasploitable and Vulnix distributions, due to its multiple vulnerabilities it was able to demonstrate the exploitability of each new vulnerability added to the knowledge DB, and the awareness brought by the instrument by the actual exploitation of these vulnerabilities, as it can be seen in figure 6.1 during first generation, the NFS scan vulnerability was added to the knowledge DB, as it was suitable for the Vulnix distribution, this was successfully exploited and therefore reported with fitness of one. It can also be seen that the User enumeration by SMTP is possible, but the dictionary attack was not successful on this case, which can be interpreted as that passwords are not as weak, for

being included in a common dictionary, this can be seen by the fitness of 0.5 on this vector at trying the attack on the Vulnix distribution shown stacked to be differentiable from each attack pattern.

On the other hand, in the Metasploitable distribution, the vulnerability added was the distcc exploit which is an exploit related to the CVE-2004-2687, exploitable at targets running the distccd daemon, which allows the attacker to an unauthorized remote access, as it can be seen at figure 6.2 the attack gets exploited successfully by the sixth generation, given that the vulnerability was added during the fifth generation execution and that it was not known to the instrument before that time.

In addition, for both cases the scoring gets different impact on the target machine, as it can be interpreted, an NFS scan just avails some information, when the distcc exploit does actually bring unauthorized access, for which the scoring must be higher, the attack scoring on each case can be seen at figures 6.3 and 6.4. From such results it can be seen that the Vulnix distribution, related to 6.4, its less prone to high scored attacks, and therefore, slightly more difficult to get accessed by unauthorized means, which corresponds to reality, as Metasploitable is more easy to intrude that the Vulnix distribution, this proof of concept represents the conditions of the changing environment in which systems are actually used, as new vulnerabilities may become available, and there could not be awareness on how secure are the systems until an actual attack takes place taking advantage of some of these.

Corrective actions

On the third testing case it must be observed that the test scenario includes both actions to be determined by the instrument, first of all the addition on the knowledge database of a new vulnerability publicly disclosed, and its correspondent execution, and later a configuration correctness, which includes the execution update due to the configuration corrected, both actions can be understood as the changing conditions over large periods of time during which exploitability of attacks its expected, this fluctuation can be seen in figure 6.6 and 6.5 at the 7th generation as the php argument injection exploit was not known previously to it, and that the samba misconfiguration got corrected by the 10th generation, this implies that the system was, in the first case able to determine the new vulnerability to be suitable at the target system, and that it exploited such vulnerability successfully then, also, it must be understood that, from the point of view of an attacker, whit no

knowledge about the target system, this is implicit by the black box testing assumption, the samba exploit is no longer available, which diminishes the potential unauthorized actions performed by an attacker against the system.

Software update scenario

By the forth testing scenario it must consider an update on the target system, to be determined by the system given that this case was not considered earlier and that such case corresponds to a common approach on security at organizations, understanding that, it must be clear that the update action must impact the exploitability and in available exploits to be used by the search method if any. For instance, this scenario is intended to cover the whole process of a vulnerability in the implementation, this is, the discovery, exploitability and the correction of that by an update, as seen by an attacker with no knowledge of the target system, this behavior can be seen at figure 6.9 and 6.10, the implementation vulnerability is determined to be suitable and executed by the 20th generation, and the update occurs at the generation 25, but, for the threshold implementation, this is visible at the results by 28th generation, being unexploited by the 29th generation.

Nevertheless as shown in figure 6.9, the vulnerability related to Samba gets exploitable again after few generations, such scenario gets repeated in order to determine the cause of such behavior, the results on such repetitions are shown at figure 6.11, from it it can be seen that the behavior is not replicated, as the update affects the attack exploitability and it does not get exploited again.

In addition, to try to determine the causes of such results at figure 6.9; it is presented an update on the search method, this update was performed by adding the velocity calculation for each parameter value on each attack pattern, in a similar manner as in the PSO algorithm, to try to give more emphasis on the exploitation over high fitness results, the velocity calculation pseudocode is shown at listing 6.1.

```

1 for attack params in current attack pattern
2   Vel_max=(Max value - Min value)/2
3   New_Vel = Vel_prev + (2*(Random(0,1))*(Current param value -
   Global best param value))+(2*(Random(0,1))*(1.0-Global best
   fitness))
4
5   if(New_Vel < Vel_max):
6     New_Vel= Vel_prev + New_Vel

```

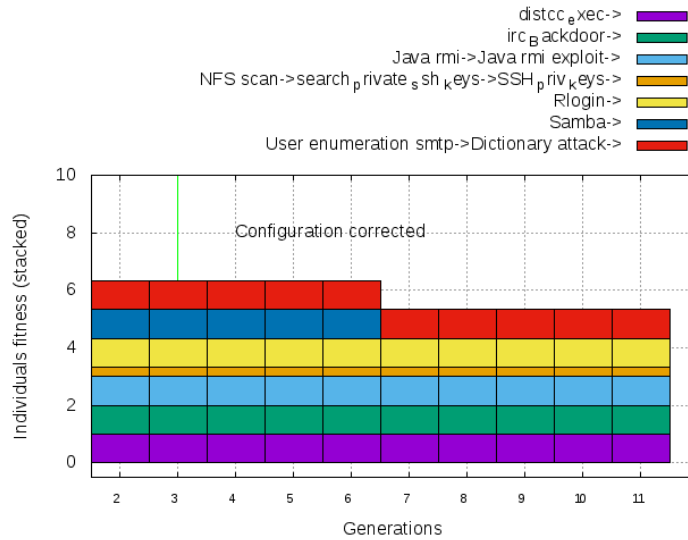


Figure 6.11: Software update scenario repetition

```

7  else :
8      Vel_new= Vel_max
9      New param value = Current value + Vel_new
10
11  if (New param value < Min value):
12      New param value = Min value
13
14  if (New param value > Max value):
15      New param value = Max val

```

Listing 6.1: Velocity calculation for attack parameters (abridged)

With this update at the search method, the scenario test was repeated for software update at Samba, the results are shown in figure 6.12

It can be seen at the figure that the behavior presented could not be reproduced; this could mean either of two cases, an error at the developed instrument, or a circumstance not considered derived from persistence. For the first case, an error at the developed instrument is likely to happen, nevertheless, the behavior was not shown at any other of the tests performed, furthermore, the results obtained are highly related to the Metasploit framework responses, so this results, if mistaken, could be related to such tool, which seems very unlikely. On the other hand, persistence, as an objective from the design and the developed instrument, can carry out such circum-

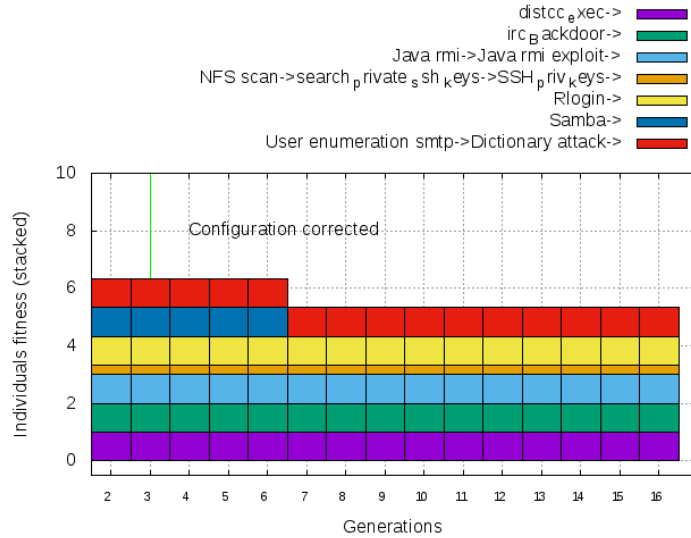


Figure 6.12: Software update scenario repetition with search method change

stance over long enough periods of time, as presented in figure 6.9, the 35 generations test is long enough to consider constant interaction, as well as persistent attack attempts, which could cause the abnormal exploitation, that could not be reproduced, even at later manual attempts at the particular target; this may imply time, interaction and parameter search on a limited scenario, which comes at hand as persistence. It can be assured that such behavior was derived from persistence owing to developed instrument design, but it seems likely, as it shows an unexpected, hard to replicate result, that may not be achieved in any other way, giving that an error was not achieved in any other scenario, in similar conditions.

The results show the behavior and vision of a well motivated attacker without any known particulars at the system under testing, implying the changes due to corrections made to the target and new knowledge determined due to public sources. In this case as the test universe is declared, the results lie within the declared vulnerabilities, already reported over the vulnerable system prone to the testing, this helps to determine that the instrument was able to determine such available knowledge, from public sources, and apply it to attack a system, and persist, given that it is already known whether or not such attacks can be exploited at the given system, it was also determined that the instrument could correctly determine so, given that, from the instrument

implementation point of view, there was no specific information about the target itself, in this understanding the instrument was able to build and test attack vectors and test it against the system determining its exploitability on each case, without any interaction from human operators.

6.4 Comparison to related work

Results shown can be either described as security awareness, understood in terms defined at table 6.3, or as proof of flaws present at the system under testing, furthermore, the tests aim to prove the exploitability of an attack under changing conditions, improving the awareness of security in a persistent manner. Based on the results provided by Bailing et al.[8] it can be interpreted the contribution on automation, and persistence among changes, the expected contribution was on the engagement on the knowledge representation, and the search method, which allowed the instrument to implement the attacks without external interaction, this results, on multi steps attacks exploitation, and recognition were included by Bailing et al [8], nevertheless, this includes human interaction on the correlation over information discovered, and the actual attacks; on the other hand, the work proposed by Semant Neha [7], includes some attack , but the relation and actual execution is always performed by an operator, as in the case of Bailing [8]; the persistence would only be achieved by a person testing the security of a given system continuously. The work by Semant Neha[7] includes attacks for Denial of Service; mainly aimed to made a service unavailable, from different methods, from this work we can recall the collection of available attacks, nevertheless, whether the attacks can actually take place at a certain system or which to be used, is determined entirely by the operator, in such understanding, the proposal includes the persistence as automation by the instrument achieved by the knowledge storage, flexible and prone to be scalable in a generic manner and the correlation with the proposed search method to determine at first the suitable vulnerability, and the exploitability of it at a given target. As previously observed, the work proposed by Xue Qiu [18], recalls a pushdown automata to overcome with automation, nevertheless, the effort to cover more widely the variety of attacks lacks of multi step ones, as well as from parameter selection done explicitly, as it is not shown on their work. The main contribution achieved by the current work's design lies in the fact that the automation actually goes in away to determine the evaluation of

the security on a system in a constant way executed by the persistence of the actual instrument and the parameter selection provided by the search method by evolution approach, and the ease of scalability understood by the knowledge representation and attack graph curating from diverse sources.

The current proposal compares to the related work by the results on persistence understood as the time it covers by the actual evaluation, it is also possible to compare the different techniques used for recognition and knowledge representation, which this proposal addresses to overcome the persistence in time, the comparison can be seen at table 6.5 for each of the major related work.

Table 6.5: Comparison with current related work

Title	Knowledge DB	Elaborated Attacks	Recognition	Parameter Control	Time coverage
Design and Implementation of a Network Attack Platform Based on Plug-in Technology[8]	Plugin	Yes	previously performed	Operated manually	Snapshot
Automated Penetration Testing[7]	Hardcoded	No	previously performed	Operated manually	Snapshot
An Automated method of penetration testing[18]	Hardcoded	Yes	Automated	None	Snapshot
Current work's proposal	Knowledge Graph	Yes	Automated persistent	Evolutive approach	Constant

In this chapter the details on tests scenarios, results obtained and analysis on it were addressed to determine suitable tests to overcome the design and development considerations to achieve a persistent intrusive evaluation, it was determined that such design and the corresponding developed instrument as a case of study, could carry on over major scenarios contemplated giving that such scenarios would represent real changes presented at system, that are relevant to be evaluated at the security, over changing conditions.

In the next chapter, the conclusions, contributions and future work are presented given the results obtained.

Conclusions and Future Work

The current work presents a novel approach to achieve security penetration testing in an automated fashion, to provide real factual feedback on a system's security related to its vulnerabilities at a given time continuously; it lacks some aspects of the actual pentesting process such as detailed reporting, nevertheless, it is a basic approach to be worked on by adding further recognizance and exploit assurance to cover more specialized cases and deal with more complicated scenarios and security controls, this aims to be a tool to help, on first instance, at the specialized security tester, by performing simple, yet compound test over certain system, leaving the further analysis to the expert; additionally it would support the system administrator unexperienced at security issues, which as the knowledge representation allows it, can intuitively check where are the security flaws at the system under his or hers administration are placed. One of the main limits at this work is the feasibility to avoid security controls, it was added some very basic approaches to achieve this, nevertheless, as security controls can be implemented in several ways, it may not successfully avoid some of them, for this, additional work must be placed on security controls and ways to avoid them, and to add such knowledge to the database for this further work is needed.

Contributions

During the present work it was presented a novel design based on related approaches for pentest automation, with such base, it was added a new way to store the knowledge as recalled from related work, it aims to be scalable and flexible to allow easier scalability and a wider approach on the attacker knowledge, as well as a search method to automate the actual testing alongside the parameter selection for each attack attempt; it was possible to represent and store attack patterns as well as determine whether or not it succeed by the

search method and to vary the parameters over time to overcome changes at the target system. It was determined a way to engage information at a certain knowledge representation, and the search method, proposed as a evolution strategy, to comply the persistence on pentesting security evaluation, this implies, that the instrument can act as an external attacker, without known of the target specifications, and that this behavior on the attacker boundaries can be determined by the systems owners and administrators in a continuous manner, availing the capability to determine more promptly a new threat or whether or not the actions taken towards security were enough to stop attacks, from attackers' point of view.

Future Work

On the other hand, the designed instrument lacks of automation on the knowledge feed, as it must be curated manually, this is an opportunity area to develop an automated stage on this feed, to provide a stronger instrument, capable of giving aid to security assessment with less effort and trustworthy information about the security of a system at a given time continuously as a persistent attacker could do it. In addition, the enumeration stage, in which testing mainly relies, is implemented at a very basic level, it would be needed some extra effort to overcome further analysis on information gathered and its authenticity, as it may be the first line of defense to provide fake information as a security control, further, information gathered may be compared to determine, more precisely, software versions and services provided by the target system , such work can be placed separately, as the design allows modular work on each stage.

Bibliography

- [1] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich, “A review of relational machine learning for knowledge graphs,” *arXiv preprint arXiv:1503.00759*, 2015.
- [2] J. D. Howard and T. A. Longstaff, “A common language for computer security incidents,” *Sandia National Laboratories*, 1998.
- [3] C. T. Wai, “Conducting a penetration test on an organization,” *SANS Institute. Retrieved January*, vol. 11, p. 2004, 2002.
- [4] S. Northcutt, J. Shenk, D. Shackleford, T. Rosenberg, R. Siles, and S. Mancini, “Penetration testing: Assessing your overall security before attackers do,” *Sponsored by Core Impact, SANS Analyst Program*, vol. 3, no. 6, p. 22, 2006.
- [5] H. Tipton and M. Krause Nozaki, *Information Security Management Handbook, Sixth Edition, Volume 6*. CRC Press, 2012. [Online]. Available: <http://www.amazon.com/Information-Security-Management-Handbook-Edition/dp/1439893136>
- [6] G. Tian-yang, S. Yin-sheng, and F. You-yuan, “Research on software security testing,” *World Academy of science, engineering and Technology*, vol. 70, pp. 647–651, 2010.
- [7] N. Samant, “Automated penetration testing,” Ph.D. dissertation, San Jose State University, 2011.
- [8] L. Gen, W. Bailing, L. Yang, B. Xuefeng, and Y. Xinling, “Design and Implementation of a Network Attack Platform Based on Plug-in Technology,” *network*, vol. 7, no. 3, pp. 195–204, 2013.

- [9] J. Fonseca, M. Vieira, and H. Madeira, “Vulnerability & attack injection for web applications,” in *Proceedings of the International Conference on Dependable Systems and Networks*, 2009, pp. 93–102.
- [10] —, “Evaluation of Web Security Mechanisms Using Vulnerability & Attack Injection,” *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 5, pp. 440–453, 2014.
- [11] A. Morais, E. Martins, A. Cavalli, and W. Jimenez, “Security protocol testing using attack trees,” *Proceedings - 12th IEEE International Conference on Computational Science and Engineering, CSE 2009*, vol. 2, pp. 690–697, 2009.
- [12] A. Morais, I. Hwang, A. Cavalli, and E. Martins, “Generating attack scenarios for the system security validation,” *Networking Science*, vol. 2, no. 3-4, pp. 69–80, 2012. [Online]. Available: <http://link.springer.com/10.1007/s13119-012-0012-0>
- [13] S. Member and N. Neves, “Vulnerability Discovery with Attack Injection,” *IEEE Transactions on Software Engineering*, vol. 36, no. 3, pp. 357–370, 2010.
- [14] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, “State of the art: Automated black-box web application vulnerability testing,” *Proceedings - IEEE Symposium on Security and Privacy*, pp. 332–345, 2010.
- [15] A. Doupé, L. Cavedon, C. Kruegel, and G. Vigna, “Enemy of the state: a state-aware black-box web vulnerability scanner,” in *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, 2012, pp. 523–538.
- [16] A. Avancini and M. Ceccato, “Towards security testing with taint analysis and genetic algorithms,” in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems*. ACM, 2010, pp. 65–71.
- [17] R. Davis, H. Shrobe, and P. Szolovits, “What is a knowledge representation?” *AI magazine*, vol. 14, no. 1, p. 17, 1993.
- [18] X. Qiu, S. Wang, Q. Jia, C. Xia, and Q. Xia, “An automated method of penetration testing,” in *Computing, Communications and IT Applications Conference (ComComAp), 2014 IEEE*. IEEE, 2014, pp. 211–216.

- [19] G. F. Lyon, *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.
- [20] L. Gen, W. Bailing, L. Yang, B. Xuefeng, and Y. Xinling, “Design and implementation of a network attack platform based on plug-in technology,” *network*, vol. 7, no. 3, 2013.
- [21] N. Hansen, D. V. Arnold, and A. Auger, “Evolution strategies,” in *Springer Handbook of Computational Intelligence*. Springer, 2015, pp. 871–898.
- [22] L. T. Heberlein, G. V. Dias, K. N. Levitt, B. Mukherjee, J. Wood, and D. Wolber, “A network security monitor,” in *Research in Security and Privacy, 1990. Proceedings., 1990 IEEE Computer Society Symposium on*. IEEE, 1990, pp. 296–304.
- [23] F. V. Yarochkin, O. Arkin, M. Kydyraliev, S.-Y. Dai, Y. Huang, and S.-Y. Kuo, “Xprobe2++: Low volume remote network information gathering tool,” in *Dependable Systems & Networks, 2009. DSN’09. IEEE/I-FIP International Conference on*. IEEE, 2009, pp. 205–210.
- [24] Y. Vanaubel, J.-J. Pansiot, P. Mérindol, and B. Donnet, “Network fingerprinting: Ttl-based router signatures,” in *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 2013, pp. 369–376.
- [25] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich, “A review of relational machine learning for knowledge graphs,” *Proceedings of the IEEE*, vol. 104, no. 1, pp. 11–33, 2016.
- [26] P. R. Srivastava and T.-h. Kim, “Application of genetic algorithm in software testing,” *International Journal of software Engineering and its Applications*, vol. 3, no. 4, pp. 87–96, 2009.
- [27] Rapid7. (2016, July) Metasploit framework. [Online]. Available: <https://www.metasploit.com/>
- [28] NIST. (2016, September) Cve data feed. [Last accessed 30 September 2016]. [Online]. Available: https://nvd.nist.gov/download.cfm#CVE_FEED

- [29] M. Mateski, C. M. Trevino, C. K. Veitch, J. Michalski, J. M. Harris, S. Maruoka, and J. Frye, “Cyber threat metrics,” *Sandia National Laboratories*, 2012.
- [30] Ar0xA. (2015, December) Fristileaks capture the flag. [Online]. Available: <https://www.vulnhub.com/entry/fristileaks-13,133/>
- [31] R. User. (2012, sep) Vulnix. [Online]. Available: <https://www.rebootuser.com/?p=933>
- [32] Rapid7. (2012, may) Metasploitable 2. [Online]. Available: <https://community.rapid7.com/docs/DOC-1875>