

Customizing MulVAL for Attack Graph Generation

*Project report submitted in partial fulfillment of the requirements
for the degree of*
Master of Computer Application

by

Tusher Mondal

Class Roll No.: 002210503039

Examination Roll No.: MCA244038

Registration No.: 163631 of 2022-2023

under the supervision of

Dr. Mridul Sankar Barik

Assistant Professor

Department of Computer Science and Engineering
Faculty of Engineering and Technology
Jadavpur University
2024

Faculty of Engineering and Technology Jadavpur University

Certificate of Recommendation

To Whom It May Concern

I hereby recommend that the project report entitled “**Customizing MulVAL for Attack Graph Generation**” has been carried out by **Tusher Mondal** (Registration No.: 163631 of 2022-2023, Class Roll No.: 002210503039, Examination Roll No.: MCA244038), under my guidance and supervision and be accepted in partial fulfilment of the requirements for the degree of Master of Computer Application in the Department of Computer Science & Engineering, Jadavpur University, during the academic year 2022-2024.

Dr. Mridul Sankar Barik
Assistant Professor
Dept. of Comp. Sc. & Engg.
Jadavpur University
(**Project Supervisor**)

Prof. Nandini Mukhopadhyay
Head of the Department
Dept. of Comp. Sc. & Engg.
Jadavpur University

Prof. Dipak Laha
Dean,
Faculty of Engineering & Technology
Jadavpur University

Faculty of Engineering and Technology

Jadavpur University

Certificate of Approval

This is to certify that the project report entitled “**Customizing MulVAL for Attack Graph Generation**” is a bonafide record of work carried out by **Tusher Mondal**, in fulfillment of the requirements for the award of the degree of Master of Computer Application in the Department of Computer Science and Engineering, Jadavpur University. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but approve the project report only for the purpose for which it has been submitted.

Signature of Examiner 1

Date:

Signature of Examiner 2

Date:

Faculty of Engineering and Technology Jadavpur University

Declaration of Compliance with Academic Ethics

I hereby declare that this project report entitled “**Customizing MulVAL for Attack Graph Generation**” is an outcome of my genuine and independent work, as part of my Master of Computer Application program. All information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, all materials and results that are not original to this work have been fully cited and referenced.

Name: Tusher Mondal

Registration No.: 163631 of 2022-2023

Examination Roll No.: MCA244038

Thesis Title: Customizing MulVAL for Attack Graph Generation

Signature with Date

Acknowledgement

I am pleased to express my gratitude and regards towards my Project Guide **Dr. Mridul Sankar Barik**, Assistant Professor, Department of Computer Science and Engineering, Jadavpur University, without whose valuable guidance, inspiration and attention towards me, pursuing my project would have been impossible.

Last but not the least, I express my regards towards my friends and family for bearing with me and for being a source of constant motivation during the entire term of the work.

Tusher Mondal

MCA Final Year

Exam Roll No. - MCA244038

Regn. No. - 163631 of 2022-2023

Department of Computer Science and Engineering,
Jadavpur University.

Contents

1	Introduction	1
1.1	Background	1
1.2	Contribution of the Project	2
1.3	Outline of the Project Report	3
2	Related Works	4
2.1	Attack Graph	4
2.2	Logical Attack Graph	5
3	The MulVAL Tool	7
3.1	History and Introduction	7
3.2	Modules and Dependencies	7
3.3	Core	8
3.4	Process Overview	9
3.5	Input Generator	10
3.6	Attack Graph Generation	11
3.7	Extension	13
4	AutoPentest-DRL	14
4.1	Motivation	14
4.2	About	14
4.3	Dependencies	15
4.4	How It Functions	15
4.5	DRL - Deep Reinforcement Learning	16
4.6	Autopentest Topology	17
4.7	Interests	18
4.8	Resolution	18
5	Customization of MulVAL	19
5.1	Goal Orientation	19
5.2	Outdated Dependencies	19
5.3	Customization of MulVAL Input Generation	21
5.4	User Interface (UI) and Technological Stack	23
5.5	Roadmap	26
5.6	Summary	28

6	Conclusion and Future Work	29
6.1	Conclusion	29
6.2	Future Work	29

List of Figures

2.1	Sample Attack Graph[17]	5
2.2	Sample Logical Attack Graph [9]	6
3.1	Flow Chart of MulVAL's Internal Process[20]	9
3.2	Sample input of readily available 3host test case from MulVal repository	11
3.3	AttackGraph.dot file Generated by MulVAL	12
3.4	Attack Graph Generated from readily available 3host test case from the MulVAL repository	13
4.1	Flow Chart of Autopentest-DRL Logic[4]	16
4.2	DRL Logic[15]	16
4.3	Sample Topology from AutoPentest Repository[2]	17
5.1	Nmap Scanner Workflow	21
5.2	Tool Logic Diagram.	22
5.3	The Main Tkinter Window.	23
5.4	Web UI landing page.	24
5.5	Nessus and Nmap Scan Results Upload Section.	24
5.6	Example of Nmap Scanning Implementation.	25
5.7	Generated Attack Graph Preview.	26
5.8	A Sample Attack Graph	26
5.9	Sample MulVAL input.	27

Abstract

This report focuses on the enhancement of the MulVAL tool, which is used to create attack graphs that highlight potential security vulnerabilities in network systems. The primary aim of this project was to modernize MulVAL by updating outdated components and making it more user-friendly with current technologies. The switch from the old XML format to the more modern JSON format for handling vulnerability data was implemented. Additionally, the MySQL controllers were updated to be compatible with newer versions of the database, and the old Java modules were replaced with more efficient Python scripts.

To further improve MulVAL, the Autopentest-DRL repository was integrated. This integration aimed to enhance the process of generating inputs for MulVAL by using Nmap, a powerful tool for detailed network scanning and vulnerability assessment.

A user-friendly interface was developed as a key part of this project. A web server and scanner using Flask for backend processing, along with a simple launcher using Tkinter, were created. This new interface allows for the easy uploading of scan results, the generation of attack graphs, and their viewing through a web browser, making complex security analyses more accessible and practical.

These improvements have significantly modernized MulVAL, ensuring its relevance and effectiveness for contemporary cybersecurity challenges. Detailed and automated security analyses can now be performed by the updated tool, providing robust and versatile resources for cybersecurity professionals. Future work will focus on further automating the input generation process, incorporating additional security tools, and enhancing the user interface to cater to a wider range of use cases and user skill levels.

Overall, the necessity of continuously updating cybersecurity tools to keep pace with evolving threats is emphasized by this project. The enhancements made to MulVAL demonstrate its increased robustness and versatility, making it a valuable tool for advanced security analysis.

Chapter 1

Introduction

1.1 Background

Network security is the cornerstone of a healthy digital world. It's the shield that protects computer networks from unauthorized access, ensuring the confidentiality, integrity, and availability of data. Strong network security safeguards sensitive information like financial records, personal details, and intellectual property. Network breaches can be devastating, costing businesses significant sums in data recovery, legal fees, and reputational damage. Robust security minimizes these risks. Cyber attacks can cripple a network, hindering access to critical resources and applications. Network security keeps operations running smoothly, preventing costly disruptions. Consumers value companies that take data privacy seriously. Effective network security demonstrates a commitment to protecting customer information and builds trust. Network security is the cornerstone of a healthy digital world. To achieve this security, organizations adopt a layered defense strategy, a combination of various tools and techniques working together. This strategy includes access control, which acts as a gatekeeper restricting access to network resources based on user permissions. Strong passwords, multi-factor authentication, and user access controls are key components of this access control layer. Firewalls act as a barrier between the trusted internal network and the wild west of the internet. These firewalls filter incoming and outgoing traffic, allowing only authorized communication to flow through. Software protectors, including antivirus and anti-malware software, safeguard devices from malicious software that can steal data or disrupt operations. Encryption scrambles data with a secret code, protecting sensitive information during transmission and storage. This ensures that only authorized users can access the information. Clear guidelines are crucial for user behavior. Security policies and procedures educate users on password management, data handling practices, and reporting security incidents. Network segmentation strengthens security by dividing the network into smaller zones. If one segment is breached, the damage is contained, minimizing the impact on the entire network. Regularly identifying and addressing weaknesses in network devices, operating systems, and applications is vital for vulnerability management. This includes installing security patches and updates promptly.

1.2 Contribution of the Project

The project involves improving and customizing the MulVAL (Multi-host, Multi-stage Vulnerability Analysis) framework to better meet today's cybersecurity needs. The contributions are varied and have greatly enhanced the tool in several important ways

1. Transition to Modern Data Formats

- One of the primary contributions was addressing the transition from the discontinued NVD (National Vulnerability Database) Vulnerability XML Feed to the JSON format. This required a comprehensive overhaul of data handling processes to ensure compatibility and efficiency in vulnerability data processing.

2. Adaptation of Controllers and Frameworks

- The project adapted MySQL controllers and XML frameworks to accommodate newer versions, ensuring that the system remains compatible with the latest technological standards and maintains its performance.

3. Python Integration

- Legacy Java modules were replaced with Python scripts, which streamlined the input generation process. This not only enhanced the efficiency of the system but also simplified the maintenance and further development of the tool.

4. Integration with Autopentest-DRL

- The integration of the Autopentest-DRL repository revolutionized the approach to input generation by utilizing Nmap for comprehensive network scanning and vulnerability assessment. This integration significantly reduced dependencies and facilitated the automation of input template generation, improving the overall workflow.

5. Development of a User-Friendly Tool

- A key contribution was the development of a user-friendly tool combining a web server and scanner. The tool, which incorporates a Tkinter-based launcher and debugging console along with Flask for backend processing, allows users to upload scan results, generate attack graphs, and visualize them through a web interface. This development made complex security analysis more accessible and practical for users.

6. Enhancement of MulVAL's Usability and Efficiency

- The project added the Autopentest-DRL repository's topology generator for input templates and created a user-friendly interface. These improvements allow users to perform detailed network scans, create comprehensive attack graphs, and easily visualize vulnerabilities, significantly enhancing the tool's usability and efficiency.

7. Laying a Foundation for Future Development

- By updating older systems and integrating modern technologies, the project has established a strong base for future enhancements of MulVAL. The groundwork laid out offers numerous opportunities for further development, such as better automation of input generation, addition of more security tools, and continuous improvement of the user interface to support a broader range of use cases and user skill levels.

Overall, this project has not only ensured the continued relevance and effectiveness of MulVAL in the rapidly evolving field of cybersecurity but also demonstrated the importance of continuous improvement and adaptation of cybersecurity tools to meet modern challenges.

1.3 Outline of the Project Report

- **Chapter 1 - Introduction** : The introduction briefly outlines the project's goals, objectives, and provides a roadmap for the content covered in subsequent chapters.
- **Chapter 2 - Related Works** : A review of existing research and relevant projects is presented to support the project and establish its context.
- **Chapter 3 - The MulVAL Tool** : A detailed explanation of the MulVAL tool, its functionalities, and its importance to the project is provided.
- **Chapter 4 - Autopentest-DRL** : An in-depth examination of Autopentest-DRL and its role in recovering input generation for MulVAL is offered.
- **Chapter 5 - Customization of MulVAL** : This chapter highlights the project's advancements, from generating inputs for MulVAL to creating attack graphs with MulVAL, all accomplished through an all-in-one tool.
- **Chapter 6 - Conclusion and Future Work** : The concluding chapter provides a summary of the project's achievements, emphasizing how effectively the solution addresses the initial problem statement. It delves into potential areas for future development, and identifies a range of opportunities for further advancement within the domain.

Chapter 2

Related Works

2.1 Attack Graph

Concept and Importance

Attack graphs are crucial in cybersecurity for visualizing and understanding potential attack paths within a network. These graphs represent the relationships between different network components and vulnerabilities, allowing security professionals to identify weak points and prioritize defenses. An attack graph typically comprises nodes and edges. [7]

- **Nodes:**
Represent system elements such as devices, hosts, or applications. They also include information about vulnerabilities and network services.
- **Edges:**
Depict potential attack paths that an attacker could exploit to move from one node to another, ultimately achieving their malicious objectives.

Applications and Benefits

Attack graphs are instrumental in various cybersecurity tasks, including[7] -

- **Vulnerability Assessment:** Identifying and visualizing vulnerabilities in a network.
- **Risk Analysis:** Assessing the potential impact and likelihood of different attack paths.
- **Security Planning:** Informing the design and implementation of security measures.

Examples and Tools

Several tools and frameworks utilize attack graphs for security analysis [8]

- **MulVAL (Multi-host, Multi-stage Vulnerability Analysis):** A widely used tool for generating attack graphs, which is the focus of this project. MulVAL uses logical rules to deduce possible attack paths based on input data about network configuration and known vulnerabilities.
- **NetSPA (Network Security Planning Architecture):** Another tool that generates attack graphs to evaluate network security.

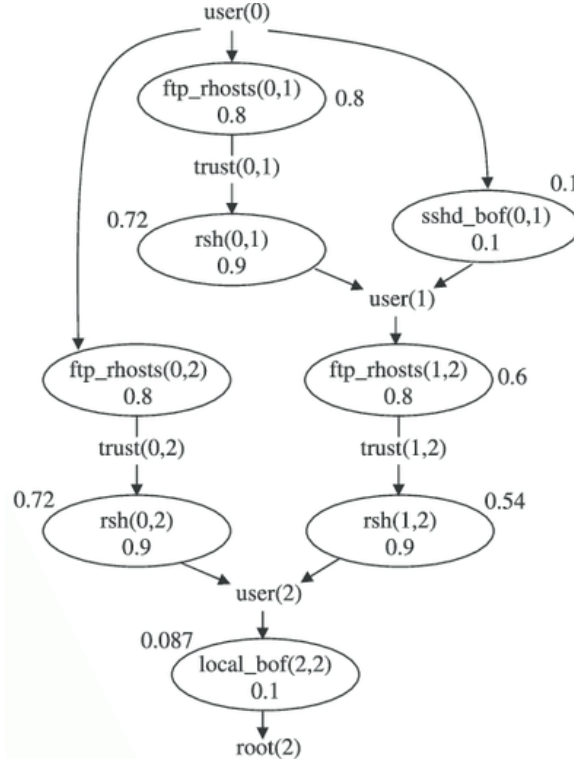


Figure 2.1: Sample Attack Graph[17]

2.2 Logical Attack Graph

Logically speaking, attack graphs have a straightforward structure. They begin with actuators (A1, A2, A3 in Figure 2.2). These actuators establish various conditions (c1, c2, c3, c4 in Figure 2.2). When these conditions are satisfied, they open up exploit regions (e1, e2 in Figure 2.2). These exploits can either be the attacker's ultimate goal or can generate new conditions, leading to additional, more complex attack paths. [16]

- **Configuration Nodes (Rectangles)** : Represent different states or configurations of the system. These states can include both secure and compromised conditions, reflecting the various configurations of software, hardware, and network settings.
Example : A configuration node might represent a state where a particular software version is installed on a server.
- **Exploit Nodes (Ovals)** : Indicate the specific actions or exploits that can transition the system from one state to another. These actions usually represent known vulnerabilities and the methods used to exploit them.
Example : An exploit node could represent the execution of a buffer overflow attack on a specific service.
- **Privilege Nodes (Diamonds)** : Depict the levels of access or privilege that an attacker can

gain by exploiting vulnerabilities. These nodes show how an attacker's privileges escalate as they move through the attack graph.

Example : A privilege node might represent gaining administrative access to a database.

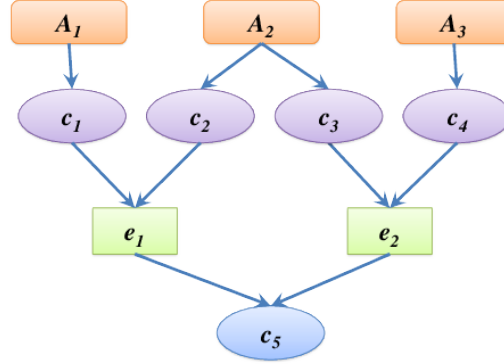


Figure 2.2: Sample Logical Attack Graph [9]

Applications of Logical Attack Graph

- **Security Posture Assessment :** Evaluate the overall security posture of a network by identifying potential weaknesses and vulnerabilities.
- **Incident Response :** Use the graph to understand attack patterns and develop effective incident response strategies.
- **Penetration Testing :** Aid penetration testers in identifying and prioritizing potential attack vectors.
- **Security Training :** Use as a tool for training security professionals by providing realistic attack scenarios.

Benefits of Logical Attack Graphs

- **Clarity and Structure:** Logical attack graphs provide a clear, step-by-step representation of attack sequences, making it easier to understand complex attack strategies.
- **Predictive Analysis:** By modeling potential attack paths, security professionals can predict possible future attacks and take proactive measures to mitigate them.
- **Comprehensive Security Planning:** Logical attack graphs help in identifying all possible attack vectors and the conditions required for each, facilitating thorough security assessments and planning.
- **Prioritization of Defenses:** Understanding the logical flow of attacks allows for prioritizing defense mechanisms where they are most needed, focusing on breaking critical attack paths.

Chapter 3

The MulVAL Tool

3.1 History and Introduction

The MulVAL tool is developed at Princeton university by Xinming Ou and Sudhakar Govindavajhala and Andrew W. Appel.

It is been in development since late 2010 and last updated on 2015. We can see this information in risksense's(The original MulVAL repository) github account.[5]

Primarily the MulVAL is designed to take observation of Nessus / Oval Scan. Both the scans provide in detail information about any given network, about hosts and all the ports, application connecting to the internet, if any known vulnerabilities is there exists, it also reports.

MulVAL is primarily written in Prolog, a logical programming language. The core component responsible for generating attack graphs functions independently and does not require additional utilities. However, this component relies on an input file formatted in a specific language. This file contains crucial information about the network, including hosts, applications, and vulnerabilities.

To generate the curated input file required by MulVAL for attack graph generation, additional utilities are necessary. The currently listed tools include MySQL 5.1 (an older version of a MySQL database management system), dom4j XML framework for Java 1.6.1 (an older version of a Java library for processing XML data), and National Vulnerability Database XML vulnerability feed 1.2.1 (an older version of an XML data feed used for input generation).

The development of the tool stopped in early 2015 and has not been updated since.[5] Consequently, the component responsible for generating input for MulVAL needs to be updated or customized to leverage current industry-standard utilities.

3.2 Modules and Dependencies

MulVAL isn't a monolithic tool. Instead, it leverages a collection of interoperable modules to achieve its functionality. Understanding these modules and their dependencies is crucial for effectively using and potentially extending MulVAL.

- **Modules**

- **Input Generator**

- This module creates input templates specifically designed for MulVAL, based on Nes-

sus/Open Vulnerability Assessment Language (Oval) security scan results of a network.[5]
nessus-translate.sh, *oval-translate.sh* - scripts are used for Input Generation

– **Attack Graph Generator**

Core module responsible for constructing an attack graph based on the refined input file produced by the Input Generator.

graph-gen.sh - scripts are used for Attack Graph Generation[5]

- **Dependencies**

- **Internal Dependencies**

- * *G++ Compiler* - for running C++ scripts.
 - * *JDK* - for running pre-compiled Java scripts.
 - * *Flex and Bison* - for parsing text data into structured data.
 - * *dom4j XML Framework* - for parsing XML data and gathering information.
 - * *Git* - for cloning and patching MulVAL repositories.
 - * *texlive-font-utils* - for enabling Graphviz to utilize various fonts.
 - * *xutils-dev* - for general-purpose operations, especially on Debian distributions.

- **External Dependencies**

- * *XSB Logic Framework* - for logic programming and inference.
 - * *NVD Vulnerability Feed* - for Getting new vulnerability information.
 - * *Graphviz* - for generating visual graph.
 - * *MySQL* - for storing all the updated vulnerability information locally.

3.3 Core

MulVAL combines a Datalog engine for modeling network element interactions with XSB Prolog, a logic programming environment.[10] This allows MulVAL to reason about security rules and generate potential attack paths. Those attack paths information are generated into log data (i.e dot file, txt file). Those logs are further processed to generate a attack graph in PDF format. An open source graph visualization software called Graphviz is used.[18]

The logic engine cares about the following clauses:

- **Network Topology** : How hosts are interconnected within the network.
- **Host Profile** : Information about IP(s), Subnet Mask, Default Gateway, Open Ports (protocol, and number), Services (on each port), Operating System (version, and distribution).
- **Vulnerability Definitions** : Vulnerabilities are typically identified from the National Vulnerability Database (NVD) and supplemented by network scans using tools like Nessus, Open Vulnerability Assessment Language (OpenVAS), and Nmap.
- **Security policies** : It defines the conditions for vulnerability exploitation based on current network configuration, connectivity, and other security factors.[18]

- **Interaction Rules** Interaction rules are defined in the logic engine. These Datalog-based rules act like a condition check for attack scenarios. Instead of focusing on specific vulnerabilities, they describe general attack methods like Trojan horses. This adaptability allows MulVAL to handle new vulnerabilities without constant updates. The rules also consider factors like privileges, network access, and exploited services to reason about how different exploit types can be chained together into a successful attack sequence.[18]

3.4 Process Overview

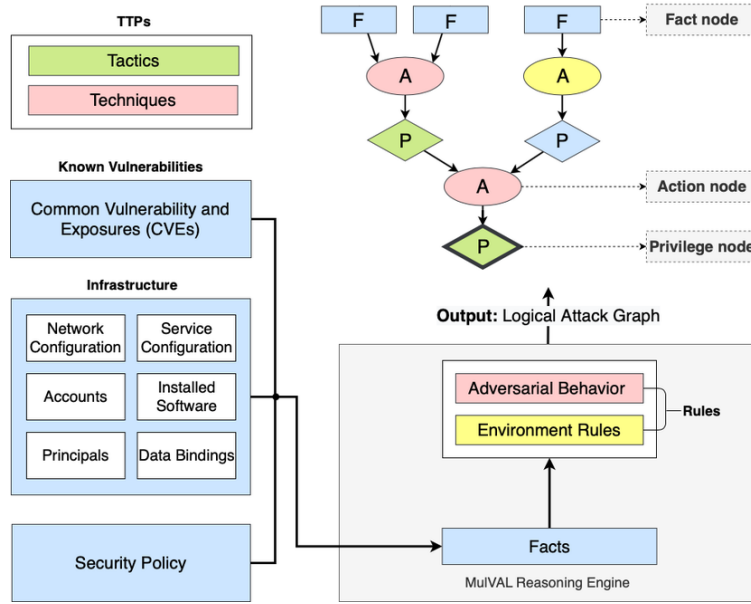


Figure 3.1: Flow Chart of MulVAL's Internal Process[20]

MulVAL's internal Process flow is briefly described below.

- **TTPs (Tactics, Techniques, and Procedures)**
 - **Tactics** : The high-level descriptions of the behaviors or strategies used by adversaries.[19]
 - **Techniques** : Specific methods used by adversaries to achieve their tactical goals. [19]
- **Known Vulnerabilities**
 - **Common Vulnerability and Exposures (CVEs)** : A list of publicly known cyber-security vulnerabilities and exposures, which are used as input for identifying potential security issues in the infrastructure.[11]
- **Infrastructure Components**
 - **Network Configuration** : The setup of network devices and connections within an organization's infrastructure.

- **Service Configuration** : The settings and options for various services running on the network.
- **Accounts** : User and service accounts that can be potential targets for adversaries.
- **Installed Software** : Software applications and systems installed in the infrastructure, which may contain vulnerabilities.
- **Principals** : Entities (users, systems, or processes) with certain permissions within the network.
- **Data Bindings** : Relationships and interactions between different data points and components in the system.
- **Security Policy** : Defines rules and guidelines to protect the organization’s assets and manage security risks.
- **MulVAL Reasoning Engine**
 - **Input - Facts**
 - * Collects data from the infrastructure, known vulnerabilities, and security policies to create a knowledge base.[14]
 - **Processes**
 - * **Adversarial Behavior**: Rules that simulate potential attacker behaviors and actions based on TTPs.[14]
 - * **Environment Rules**: Rules that describe how different components of the infrastructure interact and how vulnerabilities might be exploited.[14]
- **Logical Attack Graph**
 - **Components** :
 - * **Fact Node (F)** : Represents known facts about the system, such as existing vulnerabilities or configurations.
 - * **Action Node (A)** : Represents actions taken by an adversary, such as exploiting a vulnerability or gaining access to a system.
 - * **Privilege Node (P)** : Represents the level of access or control gained by an adversary through their actions.
 - **Output** :
 - * The MulVAL Reasoning Engine processes the input facts using the defined rules to generate a Logical Attack Graph, which visualizes potential attack paths and the progression of adversarial actions through the infrastructure.

3.5 Input Generator

While the core component of MulVAL focuses on attack graph generation, a crucial companion tool exists: the Input Generator. This tool analyzes scan results from tools like Nessus, OpenVAS, and Nmap to generate input templates specifically tailored for MulVAL. The generation process involves a sequence of processing steps and actions.[5]

The Input Generator has two way processing of Scan results (different for Nmap)

- **Step 1: Orchestration with Bash Script**

An internal Bash script acts as the conductor. It initiates the data collection process, by triggering dedicated Java modules to do the following.

- **Step 2: Data Collection and Processing with Java Modules**

The dedicated Java module serves as the workhorse. It connects to two key data sources:

- **Internal MySQL Database:** This database presumably stores vulnerability data relevant to the specific environment.
- **National Vulnerability Database (NVD):** This external database provides access to a broader range of publicly known vulnerabilities.

- **Step 3: Generated Clauses for MulVAL**

After Scans are done and collected

- The system parses scan results stored in XML format into a structure that the architecture can understand.
- After extracting vulnerability information, the system maps it to the National Vulnerability Database to retrieve additional details about the vulnerabilities through MySQL.
- The cumulative information is used to generate the input file in Datalog format.

```
input.P
1  attackerLocated(internet).
2  attackGoal(execCode(workStation,_)).
3
4  hacl(internet, webServer, tcp, 80).
5  hacl(webServer, _ , _ , _).
6  hacl(fileServer, _ , _ , _).
7  hacl(workStation, _ , _ , _).
8  hacl(H,H,_ ,_).
9
10 /* configuration information of fileServer */
11 networkServiceInfo(fileServer, mountd, rpc, 100005, root).
12 nfsExportInfo(fileServer, '/export', _anyAccess, workStation).
13 nfsExportInfo(fileServer, '/export', _anyAccess, webServer).
14 vulExists(fileServer, vulID, mountd).
15 vulProperty(vulID, remoteExploit, privEscalation).
16 localFileProtection(fileServer, root, _ , _).
17
18 /* configuration information of webServer */
19 vulExists(webServer, 'CAN-2002-0392', httpd).
20 vulProperty('CAN-2002-0392', remoteExploit, privEscalation).
21 networkServiceInfo(webServer, httpd, tcp, 80, apache).
22
23 /* configuration information of workStation */
24 nfsMounted(workStation, '/usr/local/share', fileServer, '/export', read).
25
```

Figure 3.2: Sample input of readily available 3host test case from MulVal repository

3.6 Attack Graph Generation

The system operates using a critical bash script called *"graph-gen.sh"* as its core functionality. This script is designed to analyze input files according to predefined interaction rules and subsequently produce an AttackGraph.txt file. Within this generated file, users can find comprehensive

details regarding various system components such as nodes, encompassing attacker information, host specifications, network services, vulnerabilities, and rules, along with delineated attack paths.

Additionally, users have the option to visualize the attack graph by specifying certain arguments (-v -p) when executing the script. This visualization process entails the conversion of the AttackGraph.txt file into an AttackGraph.dot file. Subsequently, the GraphViz extension within MulVAL utilizes this dot file to generate a PDF document illustrating the visual representation of the attack graph. It's worth noting that MulVAL primarily generates the AttackGraph.dot file which contains relevant information for generating a visual, which is then utilized by Graphviz for the creation of the visual attack graph.[5]

```

AttackGraph.dot
1 digraph G {
2     1 [label="1:execCode(server_192_168_1_1,dnsmasq):0",shape=diamond];
3     2 [label="2:RULE 2 (remote exploit of a server program):0",shape=ellipse];
4     3 [label="3:netAccess(server_192_168_1_1,tcp,53):0",shape=diamond];
5     4 [label="4:RULE 6 (direct network access):0",shape=ellipse];
6     5 [label="5:hacl(internet,server_192_168_1_1,tcp,53):1",shape=box];
7     6 [label="6:attackerLocated(internet):1",shape=box];
8     7 [label="7:networkServiceInfo(server_192_168_1_1,domain,tcp,53,dnsmasq):1",shape=box];
9     8 [label="8:vulExists(server_192_168_1_1,'CVE-2020-25681',domain,remoteExploit,privEscalation):1",shape=box];
10    9 [label="9:RULE 2 (remote exploit of a server program):0",shape=ellipse];
11    10 [label="10:vulExists(server_192_168_1_1,'CVE-2020-25682',domain,remoteExploit,privEscalation):1",shape=box];
12    5 -> 4;
13    6 -> 4;
14    4 -> 3;
15    3 -> 2;
16    7 -> 2;
17    8 -> 2;
18    2 -> 1;
19    3 -> 9;
20    7 -> 9;
21    10 -> 9;
22    9 -> 1;
23
24
25

```

Figure 3.3: AttackGraph.dot file Generated by MulVAL

Description of Nodes : In real network a lot of node structures are used, but via the interaction rules present, the following are the most found in Graph Data.

- **hacl(location, host, protocol, port) :** The function defines a host access control list within a network for a specific device's location. The location parameter specifies where the device is active on the network. The host parameter identifies the device using its IP address or hostname. The protocol and port parameters further define the type of application and communication channel allowed for access. More information regarding the permitted network services is provided in the subsequent networkserviceinfo clause.
- **attackerLocated(location) :** The function defines the current location of the attacker.
- **RULE number (Rule Name) :** This clause depicts the exploitation rule to define paths (if it exists) and descends to the local child node. Rules are integral part of interaction rules written in the Datalog model of MulVAL.
- **netAccess(host, protocol, port) :** Information about the host if expoNsed to the internet via net access. Application causing this phenomenon is using the mentioned protocol and port.
- **networkServiceInfo(host, service, protocol, port, application) :** This clause defines hosts with application that is running on given protocol and port utilizing the service.

- **vulExists(host, vulnerability-id, service, attack type, accesstype)** : This is the most important clause in stating which vulnerability exists with which attack type and access type on what service.
- **execCode(host, permission/account)** : This defines if custom code can be executed using the account permission on the host.

All nodes are not described.

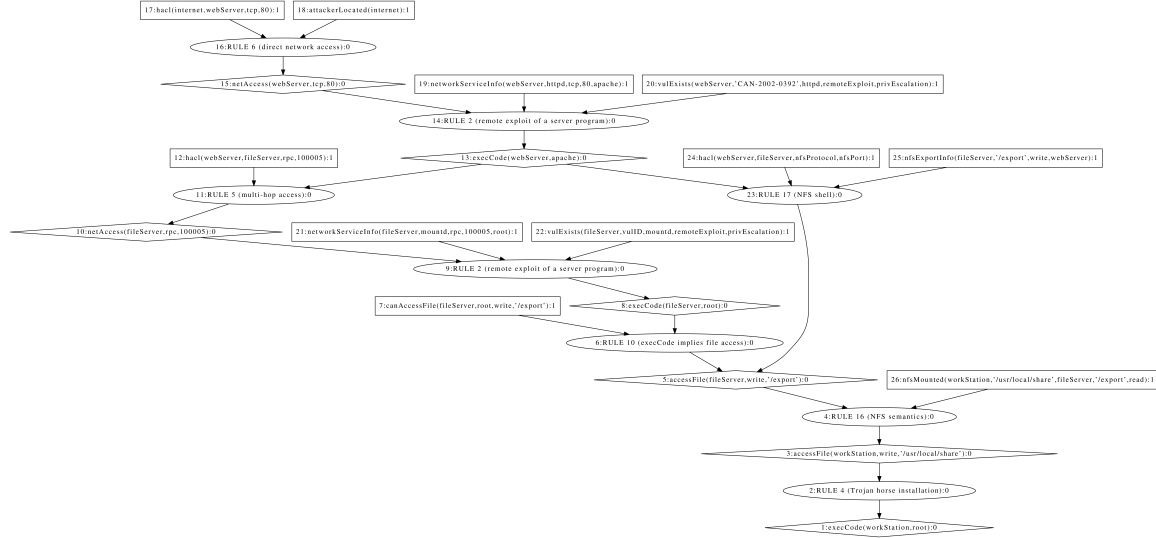


Figure 3.4: Attack Graph Generated from readily available 3host test case from the MulVAL repository

3.7 Extension

Expanding MulVAL's reach is crucial for enhancing cybersecurity. By adapting it to more areas, we can identify and address vulnerabilities across diverse systems. MulVAL's robust structure and intelligent programs make it essential for protecting our digital assets from cyber threats.

Adapting MulVAL to handle evolving threats ensures we remain prepared against increasingly sophisticated attacks. This makes MulVAL invaluable in the cybersecurity field, aiding in problem detection and informed decision-making.

In essence, MulVAL is a vital tool for cybersecurity professionals, helping maintain vigilance and resilience against cyber dangers.

Chapter 4

AutoPentest-DRL

4.1 Motivation

In the early phase of researching MulVAL, significant challenges with input generation due to outdated dependencies were encountered. It became evident that a new approach was necessary. During an exploration of GitHub forks, issues, and threads, AutoPentest-DRL was discovered, a project where a group of developers devised their own method for generating input templates for MulVAL.

Recognizing the potential of their approach, efforts were made to integrate their methodology into the application. However, given that their objectives differ somewhat, extensive rework of their solution is required. Despite these differences, their research has already provided significant insights, and it is believed that it will propel the research forward by a considerable margin. The remaining tasks are being meticulously planned to ensure a robust and efficient solution for generating attack graphs.

4.2 About

AutoPentest-DRL is an advanced automated penetration testing framework that leverages Deep Reinforcement Learning (DRL) techniques. It is designed to identify the most effective attack paths for a given logical network and execute penetration testing attacks on real networks using tools such as Nmap and Metasploit. Developed for educational purposes, AutoPentest-DRL allows users to study penetration testing mechanisms and understand various attack strategies.[1]

This innovative framework is developed by the Cyber Range Organization and Design (CROND) NEC-endowed chair at the Japan Advanced Institute of Science and Technology (JAIST) in Ishikawa, Japan. The framework receives user input regarding a logical target network, including vulnerability information. Alternatively, it can use Nmap for network scanning to detect actual vulnerabilities in a real target network with a known topology. The MulVAL attack-graph generator is utilized to determine potential attack trees, which are then simplified and fed into the DRL engine. The resulting attack path can be studied to understand attack mechanisms across various logical networks. Additionally, this attack path can be executed on a real target network using penetration testing tools such as Metasploit.[12]

4.3 Dependencies

AutoPentest-DRL leverages several key dependencies to ensure robust and efficient automated penetration testing.

- **Deep Q-learning Network (DQN)**[13] : Central to AutoPentest-DRL's architecture, the DQN algorithm enables the framework to identify and prioritize optimal attack paths based on the highest expected rewards. This reinforcement learning approach is particularly suitable for navigating complex attack scenarios.
- **Nmap** : This powerful network scanning tool automatically detects vulnerabilities in the target network. Nmap's scanning results provide crucial data on open ports, available services, and potential vulnerabilities, which are essential inputs for the attack planning process.
- **MulVAL** : A security analysis tool that constructs attack trees from the vulnerability data obtained by Nmap. These attack trees are then converted into a matrix form that can be processed by the DQN Decision Engine. This engine calculates the optimal attack path based on the provided network and vulnerability information.
- **Metasploit Framework**[13]: Used for executing the calculated attack paths. Through its Remote Procedure Call (RPC) API, Metasploit receives commands from the DQN Decision Engine and performs the actual penetration testing on the target systems. This integration allows for seamless execution of exploits, payload delivery, and post-exploitation activities.
- **Topology Generator**[13] : Creates diverse network topologies, enhancing the training model's adaptability. This component ensures that the DQN model is trained on a variety of network structures, improving its performance in different environments.
- **Depth-First Search (DFS) Algorithm**[13] : Used to simplify the matrix representation of attack paths. This optimization makes the data more manageable for input into the DQN model, enhancing the efficiency of pathfinding.

4.4 How It Functions

Autopentest-DRL offers users the flexibility to conduct testing in two distinct environments:

- **Real Network**
- **Logical Network**

For the purposes of the problem statement, the focus will be solely on **Real Network** operations.

- **Steps :**
 - Network Scanning and Consolidation Results Using Nmap (Autopentest-DRL Scripts)
 - Decoding and Vulnerability Extraction (Autopentest-DRL Scripts)
 - Topology Generation (Deep Reinforcement Learning)
 - Vulnerability Integration into Topology (Metasploit Framework)
 - Processing for MulVAL Input (Autopentest-DRL Scripts)

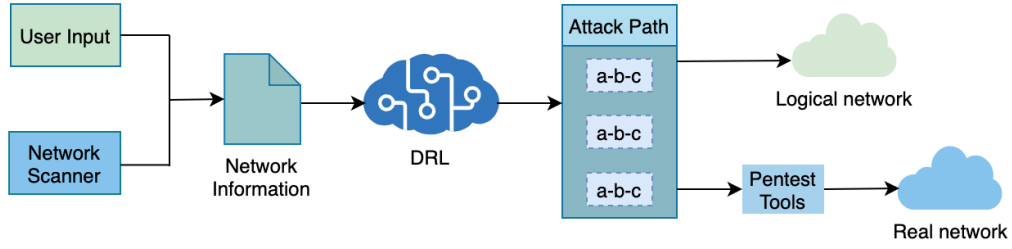


Figure 4.1: Flow Chart of Autopentest-DRL Logic[4]

4.5 DRL - Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) combines the strengths of deep learning and reinforcement learning to handle complex decision-making tasks. In reinforcement learning, an agent learns to make decisions by interacting with an environment to maximize cumulative rewards. Deep learning uses neural networks to process large amounts of data and recognize patterns. When combined, DRL enables the agent to manage high-dimensional state spaces and learn sophisticated policies.[3]

Autopentest-DRL applies DRL to automate penetration testing, which is the process of simulating cyberattacks to identify vulnerabilities in a system. By modeling the pentesting environment, including network configurations and potential vulnerabilities, a DRL agent can learn various attack strategies through simulation. The agent receives rewards for successful exploits and penalties for detection, refining its techniques over time. This automation enhances efficiency, scalability, and adaptability, allowing for continuous and effective vulnerability assessment in complex networks.[4]

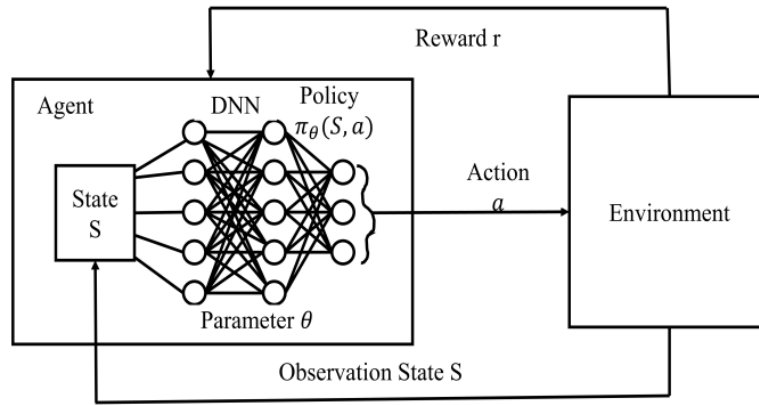


Figure 4.2: DRL Logic[15]

4.6 Autopentest Topology

Autopentest topology involves the organized setup of network components and configurations designed for automated penetration testing (pentesting). This topology replicates real-world network environments, including servers, routers, firewalls, and workstations, to create a realistic testing ground for automated pentesting tools powered by Deep Reinforcement Learning (DRL).[2]

Key elements of the topology include systems with known vulnerabilities and misconfigurations to serve as targets. Simulating common attack vectors such as SQL injection and cross-site scripting (XSS) helps the tools refine their strategies. Additionally, incorporating security defenses like intrusion detection systems (IDS) and firewalls tests the tools' ability to evade detection.

Centralized control and monitoring systems manage the deployment of attack strategies, monitor network responses, and collect data on the effectiveness of different attacks. This ensures efficient and thorough testing, providing valuable insights into network security.[2]

Overall, an effective autopentest topology offers a scalable and realistic environment for automated pentesting, enhancing efficiency, reducing manual effort, and continuously improving as new vulnerabilities and defenses emerge. The benefits of an effective autopentest topology are manifold.

```
attackerLocated(internet).|
attackGoal(execCode(workStation2,_)).

hacl(internet, fireWall, _, _).
hacl(fireWall, webServer, _, _).
hacl(webServer, fileServer, _, _).
hacl(workStation3, mailServer, _, _).
hacl(webServer, mailServer, _, _).
hacl(fileServer, workStation2, _, _).

/* configuration information of webServer */
vulExists(webServer, '{{CVE_Id_Web}}', _).
vulProperty('{{CVE_Id_Web}}', remoteExploit, privEscalation).
networkServiceInfo(webServer, _, _, _).
nfsExportInfo(webServer, '/export', _anyAccess, fireWall).

/* configuration information of workStation2 */
nfsMounted(workStation2, '/usr/local/share', fileServer, '/export', read).

/* configuration information of fireWall */
vulExists(fireWall, '{{CVE_Id_Fire}}', _).
vulProperty('{{CVE_Id_Fire}}', remoteExploit, privEscalation).
networkServiceInfo(fireWall, _, _, _).

/* configuration information of fileServer */
networkServiceInfo(fileServer, _, _, root).
nfsExportInfo(fileServer, '/export', _anyAccess, webServer).
vulExists(fileServer, '{{CVE_Id_File}}', _).
vulProperty('{{CVE_Id_File}}', remoteExploit, privEscalation).
localFileProtection(fileServer, root, _, _).
```

Figure 4.3: Sample Topology from AutoPentest Repository[2]

It offers a realistic testing environment, ensuring that findings are applicable to real-world scenar-

ios. The topology’s scalability allows it to handle large, complex networks, making it suitable for enterprise settings. Additionally, the efficiency of automated tools significantly reduces the time and resources required for manual testing. By supporting continuous learning and adaptation, the topology helps maintain the relevance and effectiveness of automated tools as new vulnerabilities and security measures emerge.

4.7 Interests

The research goal focuses on generating the input template to be used by MulVAL for attack graph generation. Specifically, our interest lies in using Nmap as a new set of scan results for generating this input. Additionally, we aim to leverage and customize the Python scripts used in AutoPentest to integrate them into our tool.

Key dependencies include:

- **Using Nmap for a New Set of Scan Results** - We employ Nmap to perform network scans, which provide detailed data on open ports, services, and vulnerabilities within the target network.
- **Replacing and Re-incorporating Nessus Scan Results** - The process is adapted to parse Nmap scan results instead of Nessus scan results, aligning with the new scripts utilized by AutoPentest-DRL.
- **Customizing Existing Python Scripts from AutoPentest** - We tailor the existing Python scripts from AutoPentest to suit our specific requirements. These modified scripts process Nmap scan results and format them for MulVAL, ensuring smooth integration between the tools.

4.8 Resolution

In the ongoing research to revamp MulVAL’s input generation module, AutoPentest plays a pivotal role. Leveraging AutoPentest-DRL’s insights, we gain valuable understanding of automated penetration testing methods, particularly in network scanning, vulnerability detection, and attack path optimization.

The existing scripts in AutoPentest serve as a solid foundation for understanding the essential mechanisms in generating accurate input templates for MulVAL. By analyzing and customizing these scripts, we seamlessly adapt and integrate them into our research framework. This allows us to utilize tools like Nmap for network scanning and Metasploit for executing attack paths, leading to a more comprehensive and efficient approach to vulnerability analysis and attack graph generation.

Overall, AutoPentest serves as a guiding force in our efforts to enhance MulVAL’s input generation module. Through the utilization of its scripts and methodologies, we aim to uncover the complexities of network security analysis and develop more robust cybersecurity solutions.

Chapter 5

Customization of MulVAL

5.1 Goal Orientation

MulVAL, a powerful framework for generating attack graphs, has historically provided valuable insights into potential system vulnerabilities. However, the project's limited maintenance and outdated dependencies have unfortunately relegated it to niche use cases. In this chapter, we detail our efforts to breathe new life into MulVAL, addressing these challenges and enhancing its capabilities for modern security analysis. We'll discuss the specific problems encountered, such as lack of updates for compatibility with contemporary systems and security standards, as well as difficulties arising from reliance on unsupported libraries. We'll then delve into the solutions we implemented to overcome these limitations, ensuring MulVAL's continued relevance in the ever-evolving security landscape.

5.2 Outdated Dependencies

MulVAL generates an Attack Graph by accepting logic input in Datalog format. Its core functionality operates independently of external libraries, relying only on Graphviz for graph visualization and utilizing bash and C for backend file processing to generate the Attack Graph. From logic interpretation to final output processing, MulVAL operates seamlessly without dependencies on additional foreign libraries. This can be confirmed by testing the 3host input.P that comes with the MulVAL repository in attack graph generation[Chapter 3].

The **Input Generation** process for MulVAL relies on various software and systems, each with its specific challenges. Those challenges were faced one by one and solutions are proposed accordingly:

- **Problem 1 : NVD Vulnerability XML Feed v1.2.1**

The 'nvd-sync.sh' script in MulVAL is crafted to gather vulnerability data from the NVD website. It then parses this information and transfers it into a local MySQL database. Java modules can subsequently access this database to retrieve the collected data whenever necessary.[5]

However, it expects the data to conform to the NVD Vulnerability XML Feed v1.2.1 format, which was discontinued after 2019. Instead, it receives data in the newer NVD Vulnerability

JSON Feed v1.0 format. This disruption impacts the retrieval of crucial information by MulVAL in later stages of operation.[6]

Solution - A Python script has been created to download relevant JSON feeds and interpret their contents to extract pertinent information. This information is then stored in the current directory, ready to be utilized later for additional processing.

- **Problem 2 : MySQL 5.1 Controller**

The pre-compiled Java class files are optimized for MySQL 5.1, utilizing connection parameters tailored to its specifications. However, the newer MySQL 8.3 version employs significantly different connection parameters. Consequently, the Java module, configured for the 5.1 MySQL controller, proves ineffective with the newer version, rendering it unusable in this context.

Solution - A Python script has been developed to mirror the functionality of the existing Java module, but with adaptations to accommodate MySQL 8.3. This script parses the stored XML feed data and inserts it into the local NVD database running on MySQL.

- **Problem 3 : dom4j XML Framework for Java v1.6.1**

The Java modules rely on the dom4j controller, specifically version 1.6.1, to parse XML-formatted scan results. However, this version lacks compatibility with the latest scan result formats, particularly those from Nessus/Oval, which have evolved over time. This issue is particularly prevalent in parsing Nessus results, necessitating the replacement of the module with a more up-to-date alternative.

Solution - To address the compatibility issue with the outdated dom4j 1.6.1 version, the CLASSPATH has been adjusted to simulate the newer dom4j 2.1.4 version. Additionally, a Python script has been developed to fully replicate the functionalities of the Java Module, specifically focusing on parsing Nessus results at the time precedence. This ensures seamless operation while accommodating the evolving format of Nessus results.

- **Problem 4 : The XSB Logic framework**

After significant preparation involving Java Modules and the MySQL NVD Database, another roadblock emerged during the generation phase. Despite overcoming the previous challenge, the execution of the Prolog code failed during input generation. Examination of the XSB log revealed that the Logic framework failed to recognize the commands set for input generation.

Solution - Despite attempts to hot plug the Prolog code and adjust it to fit the current environment, it continued to malfunction with XSB. Subsequent investigation revealed that certain rule definitions were missing from the primitive rules, which likely contributed to the failure.

Conclusions : Achieving functionality in input generation required extensive effort and restructuring of old functionalities to align with newer technologies. However, navigating the challenges posed by XSB and the absence of certain definitions in MulVAL proved to be particularly arduous. As a result, exploration of Autopentest-DRL ensued, leading to the discovery of viable workarounds to address the issues encountered.

5.3 Customization of MulVAL Input Generation

The Autopentest-DRL repository has been instrumental in revolutionizing the approach to leveraging topology-related mechanisms for MulVAL input generation. It has been discovered that utilizing Nmap with custom scripts can obviate the need for reliance on the NVD database. This not only reduces dependencies but also eliminates the necessity for XML and JSON parsers, as Nmap results inherently contain vulnerability information, including CVSS scores when used with tools like 'vulners.cse' or other vulnerability database scripting engines. Similarly, the dependency on MulVAL's outdated bash script and the reliance on XSB for input generation have been terminated, marking a significant advancement in the input generation process.

The application of Autopentest-DRL primarily focuses on stress testing artificial networks, leveraging Deep Reinforcement Learning (DRL) to regenerate topologies in response to failed attacks while utilizing existing vulnerability information. While this application is indeed impressive, it diverges slightly from the original goal, which was to automate end-to-end input template generation. While the Autopentest-DRL uses topology centric input generation, sometimes it fails to meet the requirements during real network

To achieve automation in input template generation, three consecutive steps must be undertaken. Following the Autopentest-DRL project, the Nmap tool was extensively leveraged in Input Generation Automation.

- To address the challenge of scanning one host at a time with Nmap and subsequently running vulners on each of them, a Python script has been developed. This script is designed to scan the network for alive hosts, execute vulners on each of them sequentially, and consolidate all resulting information into a single output format. This consolidated data will then be stored in one central location for further processing and analysis.
- With Nmap integrated with 'vulners.cse' effectively supplying crucial vulnerability information, encompassing details on hosts, open ports, services, and applications, the next phase involves developing an Nmap Parser. This parser will be responsible for extracting the essential information required for input generation, streamlining the automation process. The parsed information is then stored in a mapper for subsequent use in input generation.

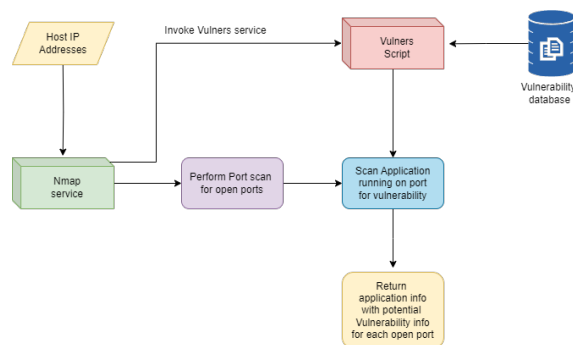


Figure 5.1: Nmap Scanner Workflow

- Another script has been developed to utilize the information stored inside the mapper for input generation. The logic of this script involves verifying the hosts and categorizing them into the **hacl** clause along with their location, protocol, and port information. Additionally, applications running on specific ports are added to the **networkServiceInfo** clause. All identified vulnerabilities are then associated with the **vulExists** and **vulProperty** clauses. However, a drawback of this script is that it requires manual input of the target host IP/name to ensure successful input generation. By default, the script assumes that the attacker is located on the internet.
- Another script has been developed to handle large networks using a curated engine while scanning for all vulnerabilities. This script takes a parameter **k**, which determines the number of vulnerabilities with the highest CVSS scores to consider. It then regenerates the attack graph accordingly. In certain scenarios, the sheer volume of discovered vulnerabilities may lead to an excessively large attack graph, complicating security analysis and tracking. To address this issue, an option to specify the maximum number of vulnerabilities (**max k**) has been incorporated. This allows the attack graph to be minimized, focusing only on the most critical threats and simplifying the analysis process.

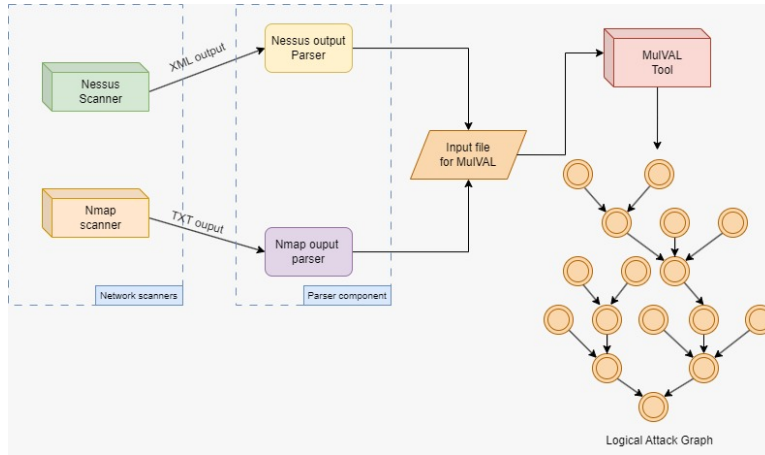


Figure 5.2: Tool Logic Diagram.

These logics are constructed based on the results obtained from Nmap scans, and subsequently, In later stage Nessus scan results are also parsed into input generation. This comprehensive approach ensures that both Nmap and Nessus scans contribute to the input generation process, enhancing the overall effectiveness of the automation.

5.4 User Interface (UI) and Technological Stack

To improve the user experience, all scripts needed to be combined related to input generation into a single, user-friendly tool. This tool should cater to regular users with a simplified interface.

Functionalities : The tool will consist of two main components -

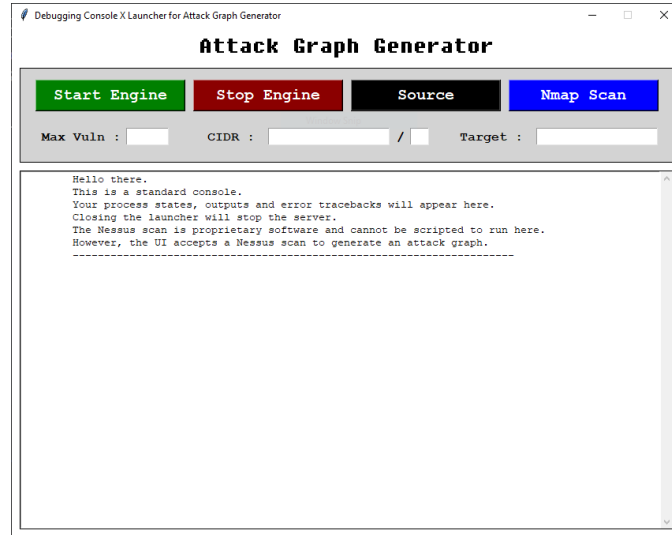


Figure 5.3: The Main Tkinter Window.

Start Engine/Stop Engine controls the Backend server, Nmap Scan scans the current network, Max Vuln specifies the number of vulnerabilities ordered by CVSS score, CIDR denotes Classless Inter-Domain Routing for Scanning, and Target specifies the host for generating the attack graph.

- **A Web Server :** This component provides the standard web interface (UI) for users.
 - Users can upload their scan results to the web server.
 - The web server processes the uploaded data and generates an attack graph.
 - Users can download the attack graph as a local file.
 - The web UI also displays a preview of the attack graph for quick visualization.



Figure 5.4: Web UI landing page.

- The figure below shows the section where the user needs to upload the scan results.

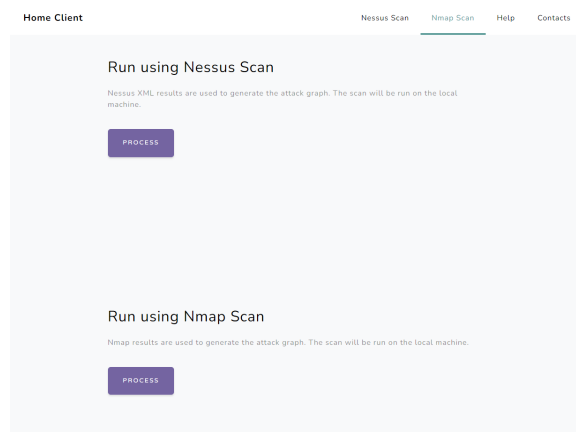


Figure 5.5: Nessus and Nmap Scan Results Upload Section.

- **A Scanner** : This component is responsible for network scanning. It utilizes Nmap for network discovery and vulnerability identification.

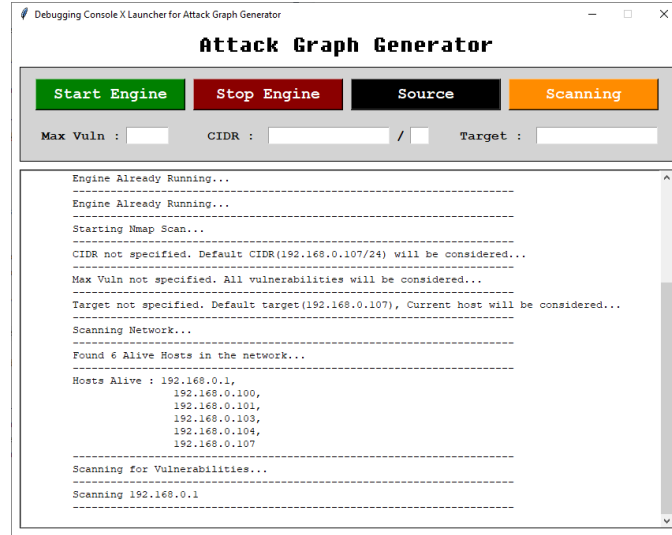


Figure 5.6: Example of Nmap Scanning Implementation.

Supporting Technology : To create a visually appealing user interface (UI), we opted for a combination of technologies:

- **Launcher/Debugging Console** : Python Tkinter was used to create a visual application that includes a launcher for the web UI and backend server. An in-built console was also incorporated to monitor the logs of ongoing functions.
- **Front-end** : HTML, CSS, and JavaScript - This industry-standard trio provides flexibility and visual richness for the web interface.
- **Back-end** : Flask is a lightweight web framework commonly used in Python for building web applications. It efficiently handles user interactions and server-side logic. It is extensively used to handle the scripting and running MulVAL in the background.

- After scanning, on uploading the engine automatically runs MulVAL in the background, displays the Attack Graph as a preview, and provides an option to download it.

Combining MulVAL Attack Graph Generation and Input Generation into a unified tool offers practicality and simplicity. Generating attack graphs becomes effortless, eliminating the need for scripting knowledge.


```

attackerLocated(internet).
attackGoal(execCode(server_192_168_1_1,_)).

hacl(internet, server_192_168_1_1, 'tcp', 53).
vulExists(server_192_168_1_1, 'CVE-2020-25681', 'domain').
vulProperty('CVE-2020-25681', remoteExploit, privEscalation).
vulExists(server_192_168_1_1, 'CVE-2020-25682', 'domain').
vulProperty('CVE-2020-25682', remoteExploit, privEscalation).
networkServiceInfo(server_192_168_1_1, 'domain', 'tcp', 53, 'dnsmasq').

hacl(internet, server_192_168_1_1, 'tcp', 80).
networkServiceInfo(server_192_168_1_1, 'http', 'tcp', 80, 'Boa').

hacl(internet, server_192_168_1_14, 'tcp', 135).
networkServiceInfo(server_192_168_1_14, 'msrpc', 'tcp', 135, 'Microsoft').

hacl(internet, server_192_168_1_14, 'tcp', 139).
networkServiceInfo(server_192_168_1_14, 'netbios-ssn', 'tcp', 139, '').

hacl(internet, server_192_168_1_14, 'tcp', 445).
networkServiceInfo(server_192_168_1_14, 'microsoft-ds?', 'tcp', 445, '').

```

Figure 5.9: Sample MulVAL input.

3. **Input Generation Script** : A script has been created to utilize the mapped information for generating inputs. This script verifies hosts, categorizes them into the hacl clause with location, protocol, and port details, and adds applications running on specific ports to the networkServiceInfo clause. Identified vulnerabilities are linked to the vulExists and vulProperty clauses. While this script automates much of the process, it still requires manual input of the target host's IP/name to ensure accurate input generation.
4. **Handling Large Networks and High Vulnerability Volume** : To manage large networks, another script has been developed that uses a curated engine to scan for all vulnerabilities. This script includes a parameter k, which specifies the number of vulnerabilities with the highest CVSS scores to consider for the attack graph. This helps in generating a manageable attack graph focused on the most critical threats, simplifying the security analysis and tracking process.

5.6 Summary

In this chapter, we have detailed our efforts to modernize and enhance the MulVAL framework, addressing its outdated dependencies and extending its functionality to meet contemporary security analysis needs. Through meticulous problem-solving and innovative scripting, we tackled challenges ranging from obsolete data formats to unsupported libraries, ensuring MulVAL's compatibility with current systems and security standards.

We identified and resolved key issues, such as the transition from the discontinued NVD Vulnerability XML Feed to the JSON format, and the adaptation of MySQL controllers and XML frameworks to accommodate newer versions. By leveraging Python scripts, we replaced legacy Java modules and streamlined the input generation process, ensuring seamless operation and enhanced efficiency.

Furthermore, we explored the integration of the Autopentest-DRL repository, which revolutionized the approach to input generation by utilizing Nmap for comprehensive network scanning and vulnerability assessment. This not only reduced dependencies but also facilitated the automation of input template generation, significantly improving the overall workflow.

To enhance user experience, we developed a user-friendly tool combining a web server and scanner, incorporating a Tkinter-based launcher and debugging console, and utilizing Flask for backend processing. The tool enables users to upload scan results, generate attack graphs, and visualize them through a web interface, bridging the gap between complex security analysis and accessible, practical application.

Overall, our efforts have revitalized MulVAL, ensuring its continued relevance and effectiveness in the ever-evolving landscape of cybersecurity. By addressing its limitations and integrating modern technologies, we have created a robust and versatile tool that empowers users to perform advanced security analyses with greater ease and accuracy.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In conclusion, this project has successfully updated the MulVAL framework to meet modern security needs and integrated it with advanced tools to improve its functionality. The updates included fixing old dependencies, moving away from outdated data formats, and making the input generation process easier. By using Python scripts and modern web technologies, we have kept MulVAL relevant and effective for security analysis.

Adding the Autopentest-DRL repository's topology generator for input templates and creating a user-friendly interface have greatly improved the tool's usability and efficiency. These improvements allow users to perform detailed network scans, create comprehensive attack graphs, and easily visualize vulnerabilities. The project's success shows that it's possible to update older systems and highlights the importance of constantly improving cybersecurity tools.

This work has created a strong base for future developments, offering many opportunities to further enhance MulVAL. Future work could focus on better automation of input generation, adding more security tools, and further improving the user interface to support a wider range of use cases and user skill levels. Overall, this project emphasizes the need for adaptable and reliable tools to maintain strong cybersecurity defenses.

6.2 Future Work

While discussing about the Current problem statement, It is evident that the Tool serves in Automating from generating the input template to generating the attack graph driving MulVAL independently.

But It is still found in a room of improvement. Although it establishes a base of building a common framework to incorporate more and more functionallities for MulVAL to take advantage of, It still lacks the support of multiple scans. Adhocs can be performed and the generation of the input template goes through a channel of files, But still user needs to be proficient in python and relative technologies for any other Scans or scalability if they want to use with the tool.

- **Current Limitations**

- The tool can process Nessus results and generate an attack graph, but the process is not fully automated. Since Nessus is proprietary software, Nessus scripts cannot be incorporated into the tool to be driven using Python.
- The backend input generation requires further development to incorporate OVAL definitions, as the tool currently does not support OVAL.
- The nodes in the generated graph are quite descriptive, which causes them to take up a lot of Horizontal space in a PDF. This becomes problematic when the graph contains many vertices, such as in scenarios where a host has numerous vulnerabilities. The resulting graph can become chaotic, with overlapping edges making analysis difficult and time-consuming.
- The parsers are designed to function correctly only if Nessus and Nmap results maintain a consistent format, including extensions and data structures. However, since Nessus and Nmap frequently update their output formats, it becomes necessary to regularly update the parsers to ensure compatibility with these changes.
- MulVAL requires a Debian-based distribution to run effectively. While the tool itself is OS-independent, MulVAL is not. For this project, we extensively used WSL (Windows Subsystem for Linux) to run MulVAL on a Windows system.

- **Future Prospects and Related Development**

- To increase the scalability of MulVAL, it needs to become OS-independent. Achieving this requires extensive customization of MulVAL. As dependencies may change, new alternatives will need to be identified and integrated. This will involve a significant amount of research and development.
- Graphviz is a workable solution, but it struggles with dynamic visual node structures, making the visual graph harder to analyze. A better alternative or a complete redevelopment of Graphviz is needed to enhance its capability. Since Graphviz is open-source, it can be customized to create graphs that are more visually appealing and understandable.
- A CI/CD pipeline needs to be maintained for all existing parsers, as well as for those that will be added in future versions of the tool.
- MulVAL relies on Datalog, a subset of Prolog. While currently functional, comprehending its intricacies demands substantial effort. Debugging without a solid grasp of its workings can be a daunting task, especially as Prolog's usage diminishes. Thankfully, alternatives exist, such as Elixir, XAML with dotNet, SilQ, or NoSQL with Python.

Bibliography

- [1] Autonomous Penetration Testing using Reinforcement Learning — arxiv.org. <https://arxiv.org/abs/1905.05965>. [Accessed 30-05-2024].
- [2] AutoPentest-DRL/user_guide.md at master · crond-jaist/AutoPentest-DRL — github.com. https://github.com/crond-jaist/AutoPentest-DRL/blob/master/user_guide.md. [Accessed 30-05-2024].
- [3] Deep Reinforcement Learning — deepmind.google. <https://deepmind.google/discover/blog/deep-reinforcement-learning/>. [Accessed 30-05-2024].
- [4] GitHub - crond-jaist/AutoPentest-DRL: AutoPentest-DRL: Automated Penetration Testing Using Deep Reinforcement Learning — github.com. <https://github.com/crond-jaist/AutoPentest-DRL>. [Accessed 30-05-2024].
- [5] GitHub - risksense/mulval: A logic-based enterprise network security analyzer — github.com. <https://github.com/risksense/mulval>. [Accessed 30-05-2024].
- [6] NVD - XML Vulnerability Feed Retirement — nvd.nist.gov. <https://nvd.nist.gov/general/news/XML-Vulnerability-Feed-Retirement>. [Accessed 30-05-2024].
- [7] The Power of Attack Graphs in Cloud — info.xmcyber.com. <https://info.xmcyber.com/the-power-of-attack-graphs-in-cloud>. [Accessed 29-05-2024].
- [8] Tools for Generating and Analyzing Attack Graphs — link.springer.com. https://link.springer.com/chapter/10.1007/978-3-540-30101-1_17. [Accessed 29-05-2024].
- [9] M. Albanese, S. Jajodia, and S. Noel. Time-efficient and cost-effective network hardening using attack graphs. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, pages 1–12. IEEE, 2012.
- [10] M. S. Barik, A. Sengupta, and C. Mazumdar. Attack graph generation and analysis techniques. *Defence science journal*, 66(6):559, 2016.
- [11] O. Grigorescu, A. Nica, M. Dascalu, and R. Rughinis. Cve2att&ck: Bert-based mapping of cves to mitre att&ck techniques. *Algorithms*, 15(9):314, 2022.
- [12] Z. HU. Automated penetration testing using deep reinforcement learning. 2021.

- [13] Z. Hu, R. Beuran, and Y. Tan. Automated penetration testing using deep reinforcement learning. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 2–10. IEEE, 2020.
- [14] R. Joshi and E. S. Pilli. *Fundamentals of Network Forensics*. Springer, 2016.
- [15] H. Kabir, M.-L. Tham, and Y. C. Chang. Twin delayed ddpq based dynamic power allocation for mobility in iort. *Journal of Communications Software and Systems*, 19(1):19–29, 2023.
- [16] C. Liu, A. Singhal, and D. Wijesekera. A logic-based network forensic model for evidence analysis. In *Advances in Digital Forensics XI: 11th IFIP WG 11.9 International Conference, Orlando, FL, USA, January 26-28, 2015, Revised Selected Papers 11*, pages 129–145. Springer, 2015.
- [17] I. Matthews, J. Mace, S. Soudjani, and A. van Moorsel. Cyclic bayesian attack graphs: a systematic computational approach. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 129–136. IEEE, 2020.
- [18] X. Ou, W. F. Boyer, and M. A. McQueen. A scalable approach to attack graph generation. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 336–345, 2006.
- [19] A. Purdy. The global cyber security challenge. *NA: Huawei*, 2016.
- [20] D. Tayouri, N. Baum, A. Shabtai, and R. Puzis. A survey of mulval extensions and their attack scenarios coverage. *IEEE Access*, 2023.