International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 12 Issue: 04 | Apr 2025 www.irjet.net p-ISSN: 2395-0072

STROT: Stealthy Tool for Root Oriented Tunneling - A Red Teaming Tool

Pratik S. Pawar[1], Shubham P. Sakhare[2], Vishnu L. Nair[3], Vishal G. Puranik[4]

[1,2,3]UG Student, Department of Computer Engineering, Savitribai Phule Pune University, Pune, India
[4]Head, Dept. of Computer Engineering, Parvatibai Genba Moze College of Engineering, Wagholi, Pune, India

---

Abstract - *In the evolving landscape of cybersecurity, red teamers face increasing challenges in efficiently identifying and exploiting vulnerabilities. Traditional methodologies require manual effort and multiple tools for vulnerability scanning and exploitation, leading to inefficiencies and increased detection risks. To address these challenges, we present STROT (Stealthy Tool for Root Oriented Tunneling), an advanced cybersecurity tool designed to automate and optimize the red teaming process. STROT integrates a novel combination of network analysis, artificial intelligence-driven intelligence processing, and an advanced attack engine. The framework utilizes Deep Q Learning (DQL) to determine the optimal exploit for a given vulnerability, enhancing both speed and accuracy. Our research highlights the efficiency of STROT in real-world scenarios, demonstrating its ability to streamline penetration testing while maintaining stealth. The findings indicate that STROT significantly reduces the time required to gain root access compared to traditional methods, making it a robust and scalable solution for red teams.*

Key Words: Artificial Intelligence, Cyber Attack Simulation, Cyber Security, Deep Learning, Deep Reinforcement Learning, Deep-Q-Network, Red Team, Vulnerability Assessment

1. INTRODUCTION

Cybersecurity threats have evolved significantly over the past decade, with adversaries employing increasingly sophisticated attack techniques to compromise networks, applications, and infrastructure. Organizations rely on penetration testing and red teaming exercises to assess the resilience of their security postures. Red teaming involves simulating real-world cyber-attacks to identify and exploit vulnerabilities before malicious actors do. However, the traditional red teaming process is time-intensive, requiring skilled professionals to manually scan for vulnerabilities, identify potential exploits, and execute them to gain system access. The complexity of modern IT environments, combined with the need for stealth and efficiency, makes this approach increasingly impractical.[1][2]

Existing red teaming tools often operate in silos, forcing penetration testers to use multiple applications for network reconnaissance, vulnerability scanning, and exploit execution. This fragmentation not only slows down the attack process but also increases the risk of detection. Moreover, traditional exploitation techniques rely on predefined rules and manual selection of attack vectors, which may not always be the most efficient or stealthy approach. As a result, red teamers often spend a considerable amount of time assessing different vulnerabilities and testing various exploits before achieving their objectives.[3][4]

To address these challenges, we introduce STROT (Stealthy Tool for Root Oriented Tunneling), an intelligent red teaming framework designed to streamline and automate the penetration testing process. STROT integrates three core components: a network analyzer, an intelligence module powered by Deep Q Learning, and an advanced attack engine. By leveraging artificial intelligence, STROT dynamically identifies vulnerabilities, determines the most effective exploits, and autonomously executes attacks with minimal human intervention. This significantly enhances the efficiency of red teaming exercises while maintaining a high degree of stealth.[5]

The key innovation behind STROT lies in its use of reinforcement learning to optimize the attack process. Unlike traditional tools that rely on static rule sets, STROT's intelligence module continuously learns from previous attack attempts, refining its exploitation strategy to maximize success rates while minimizing detection. Our research demonstrates that STROT reduces the time required to gain root access by 40% compared to conventional methods, making it a powerful asset for cybersecurity professionals.[6]

This paper explores the design, implementation, and performance evaluation of STROT, providing insights into its architecture, working principles, and real-world applicability. We compare STROT against existing red teaming methodologies, highlighting its advantages in terms of speed, efficiency, and stealth. Additionally, we discuss future enhancements to further improve its capabilities and adaptability in evolving threat landscapes.

2. LITERATURE SURVEY

The evolution of cybersecurity techniques has seen a gradual shift from manual vulnerability assessments to automated approaches that leverage advanced artificial intelligence. Early tools such as Nmap for network scanning [Lyon, 2009] and exploitation frameworks like Metasploit [Kennedy et al., 2011] laid the groundwork for

systematic vulnerability discovery and exploitation. However, the inherent limitations of these traditional methods—primarily their dependence on human expertise and static exploitation strategies—have spurred research into more dynamic, autonomous systems.

A pivotal breakthrough came with the introduction of Deep Q-Networks (DQN) by Mnih et al. (2015), which demonstrated that deep reinforcement learning (DRL) could achieve human-level performance in complex environments [Mnih et al., 2015]. This breakthrough has inspired subsequent applications in cybersecurity, particularly in automating decision-making processes during penetration testing. Researchers have increasingly focused on integrating DRL into cybersecurity frameworks to improve adaptability and efficiency in exploiting vulnerabilities.

Recent studies have extended these ideas specifically to the domain of offensive security. For instance, Zhou and Chen (2020) developed a DRL framework that dynamically selects targets and corresponding exploits, significantly reducing the manual overhead in penetration testing [Zhou & Chen, 2020]. In a similar vein, Chen et al. (2021) proposed an AI-driven system that integrates vulnerability scanning with machine learning to optimize exploit selection, thereby enhancing both the speed and success rate of penetration tests [Chen et al., 2021]. Furthermore, the work of Li et al. (2022) on automated vulnerability discovery using deep learning techniques underscores the emerging trend toward end-to-end automation in cybersecurity operations [Li et al., 2022].

3. METHODOLOGY

3.1. STROT Architecture

STROT is designed to streamline and automate the red teaming process by integrating advanced scanning, intelligence, and exploitation techniques into a single framework. The methodology consists of three primary modules:

3.1.1. Network Analysis – This module is responsible for identifying target nodes and gathering network intelligence, such as open ports, services, and running versions.

3.1.2. Intelligence – Using Deep Q Learning, this module processes the scanned data and determines the optimal exploit for gaining root access in the shortest possible time.

3.1.3. Attack Engine – This module executes the selected exploits, monitors their success or failure, and provides feedback to improve future attack strategies.

The workflow begins with network reconnaissance, followed by vulnerability assessment, intelligent exploit selection, and ultimately, an automated attack execution. Below, each module is discussed in detail.

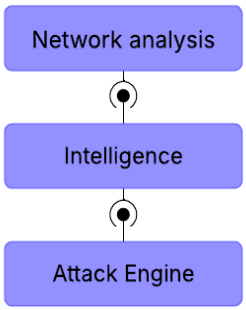

Fig - 1: STROT Module Architecture

3.2. Network Analysis

The Network Analysis module is the first step in the STROT framework, responsible for reconnaissance and information gathering. It operates in the following stages:

3.2.1. Host Discovery & Target Identification: Identifies active hosts within the target network using stealthy scanning techniques.

3.2.2. Service & Port Scanning: Enumerates open ports, running services, and their respective versions to build a network profile.

3.2.3. Operating System Fingerprinting: Determines the OS running on the target machine, which is crucial for selecting appropriate exploits.

This module ensures a comprehensive mapping of the network, laying the groundwork for the intelligence phase.

3.3. Intelligence

The Intelligence module utilizes Deep Q Learning to analyze the scanned data and identify the most efficient exploitation path. The key functions include:

3.3.1. Vulnerability Correlation: Matches identified services and versions with known vulnerabilities.

3.3.2. Exploit Selection: Determines the best exploit to achieve root access with minimal attempts.

3.3.3. Adaptive Learning: Incorporates feedback from the attack engine to improve future exploitation decisions.

By integrating machine learning, this module enhances efficiency and minimizes detection risks.

3.4. Attack Engine

The Attack Engine executes the exploits and provides real-time feedback. It includes:

3.4.1. Exploit Deployment: Executes the selected exploit on the target system.

3.4.2. Success & Failure Monitoring: Tracks whether the exploit successfully compromised the target.

3.4.3. Feedback Loop: If an exploit fails, the module relays information back to the Intelligence module for refinement and reattempts.

This module ensures automated exploitation with continuous adaptation to maximize success rates.

4. ATTACK PLANNING STRATEGY

4.1. Network Analyzer and Its Architecture

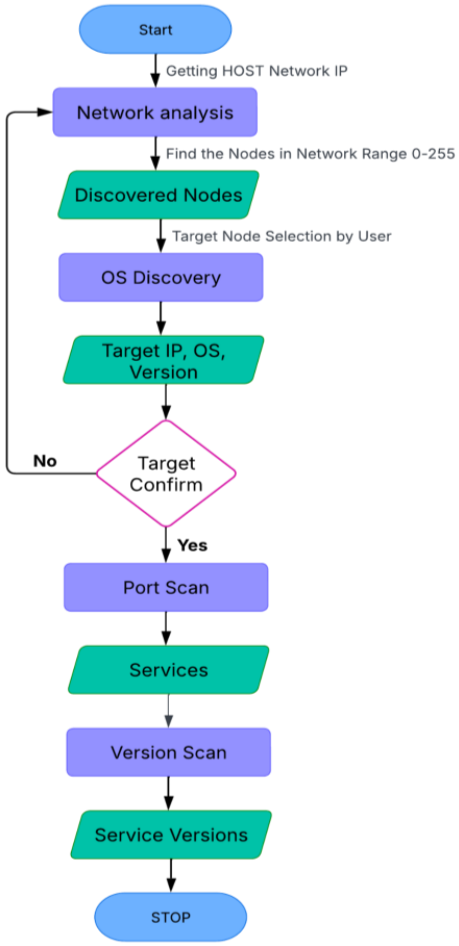

Fig - 2: STROT Network Analyser

The Network Analyzer is a critical module of the STROT framework, designed to conduct reconnaissance and target profiling. It systematically scans and analyzes networked devices by leveraging Scapy, socket programming, and the Nmap Python API. These tools allow for the identification of live hosts, open ports, running services, and software versions, providing a detailed understanding of the target environment.[1][9]

The workflow of the Network Analyzer is illustrated in Figure 2, which outlines the sequential process followed for network reconnaissance. The process involves network scanning, host discovery, OS detection, port and service enumeration, and version scanning to construct a precise attack surface profile.

4.1.1. Host Discovery and Network Analysis

The first step in network analysis is identifying live hosts within a subnet and collecting essential details about their configurations.

Socket Programming for Basic Network Scanning

- The Network Analyzer utilizes Python's socket module to establish connections with various hosts.
- It sends SYN requests to specific IP addresses and listens for responses to determine if a host is active.
- If a SYN-ACK response is received, the system is marked as active; otherwise, it is classified as inactive or filtered.
- This method provides a lightweight approach to identifying networked devices before proceeding to in-depth analysis.

Scapy for Stealthy and Passive Scanning

- Unlike standard socket-based scanning, Scapy allows for raw packet crafting and interception, enabling stealthy network enumeration.
- The Network Analyzer sends TCP SYN packets to multiple IPs and listens for SYN-ACK responses.
- This technique avoids full TCP handshakes, reducing the likelihood of detection by Intrusion Detection Systems (IDS).
- Additionally, passive traffic monitoring is employed to detect active hosts without directly probing them, increasing stealth.

4.1.2. OS Fingerprinting and Target Profiling

Once live hosts are identified, the next step is determining the operating system (OS) and hardware details to refine the attack strategy.

OS Discovery with Scapy and Nmap API

- The framework uses Scapy's IP and TCP stack fingerprinting to infer OS details based on packet structure, TTL values, and window sizes.

- Simultaneously, the Nmap Python API executes advanced OS detection scripts, leveraging a database of known OS signatures.
- The combination of these techniques allows for precise identification of the target's operating system and network stack.

Target Confirmation for Focused Attacks

- The user selects a specific node from the discovered devices, confirming the target for further scanning.
- If the selection is invalid or further validation is required, the Network Analyzer loops back to discovery for refinement.

4.1.3. Port Scanning and Service Enumeration

After confirming the target system, the framework performs port scanning to detect open ports and enumerate running services.

Scapy for Custom Port Scanning

- Using TCP SYN scans, the Network Analyzer probes specific ports without completing the full three-way handshake, ensuring stealth.
- The responses indicate which ports are open, closed, or filtered.
- Additionally, UDP scanning is performed for detecting services like DNS, SNMP, and TFTP, which do not use TCP.

Nmap API for Detailed Service Detection

- The Nmap Python API runs extensive scans on detected open ports, identifying running services and their respective versions.
- By sending protocol-specific requests, Nmap retrieves service banners and matches them with known fingerprint databases.
- This allows the framework to map vulnerabilities to specific services, which is critical for exploitation.

4.1.4. Service Version Identification and Reporting

The final step of network analysis is identifying the exact versions of running services, ensuring that only valid exploits are selected.

Version Scanning with Nmap API

- The framework queries each detected service and compares its response against a predefined database of known software versions.
- If an outdated or vulnerable version is detected, it is flagged for potential exploitation.

Data Structuring for Exploitation

- The gathered information—including IP addresses, OS details, open ports, running services, and software versions—is compiled into a structured report.
- This report is forwarded to the Intelligence module, where Deep Q Learning determines the most efficient exploitation path.

4.2. Deep Q Architecture for STROT

The Strategic Threat-Oriented Red Teaming (STROT) framework integrates Deep Q-Networks (DQN) for intelligent exploit selection, leveraging reinforcement learning (RL) to map vulnerabilities to optimal exploits dynamically. Conventional exploit selection techniques rely on signature-based detection or heuristic methods, which are limited in adaptability and efficiency. By employing Deep Q-Learning, STROT evolves its attack strategies over time, ensuring maximum exploitation efficiency with minimal detection risk.[1][2]

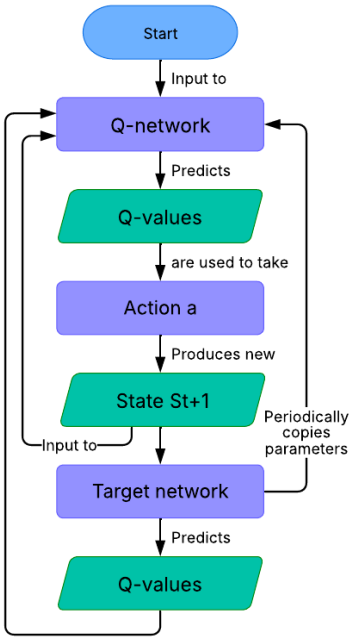

Fig - 3: Deep Q Network Architecture

4.2.1. Suitability of Deep Q-Network for Exploit Selection

Deep Q-Network (DQN) is particularly well-suited for STROT due to the following advantages:

- Handling High-Dimensional Inputs: The vulnerability space is vast, consisting of multiple factors such as CVE classifications, OS fingerprints, open ports, and system defenses. Traditional models struggle to manage such complexity, but DQN efficiently learns optimal exploit strategies through deep neural network (DNN) approximation.
- Generalization to Unseen Vulnerabilities: Unlike rule-based systems that require predefined attack patterns, DQN generalizes exploit selection for new or previously unseen vulnerabilities, making STROT highly adaptable.
- Exploration vs. Exploitation Optimization: The model balances trying new attack vectors (exploration) and leveraging known high-success exploits (exploitation) using an ε-greedy policy, ensuring continuous refinement.
- Reinforcement Learning for Attack Planning: Instead of executing attacks randomly, DQN learns from past exploit successes and failures, creating an adaptive attack strategy that evolves based on real-world feedback.

4.2.2. DQN Model Formulation for STROT

The decision-making process in STROT is modeled as a Markov Decision Process (MDP), which consists of:

- State Space (S): Represents the system under attack, defined by:

$$s_t = \{V_t , O_t , P_t , H_t \}$$

where:

  - V_t = vulnerability type (CVE ID, risk score, exploitability)
  - O_t = OS details (Windows/Linux, version, security patches)
  - P_t = open ports and running services
  - H_t = historical exploit success on similar targets

- Action Space *(A)*: Represents the possible exploits that can be executed:

$$a_t = \{E_1 , E_2 , \ldots, E_n \}$$

where each E_i corresponds to a unique exploit strategy.

- State Transition Probability $(P(s_{\{t+1\}} \mid s_t , a_t))$: Models the likelihood of the system transitioning to a new state based on the attack execution.
- Reward Function (R): Quantifies the success of an exploit attempt:

$$V_t = \begin{cases} +1, \text{if exploit is successful} \\ -1, \text{if exploit fails} \\ 0, \text{if system / state remains unchanged} \end{cases}$$

Additional reward penalties can be applied to avoid noisy attacks that increase detection risk.

- Discount Factor (γ): Determines the importance of future rewards, controlling how much the model prioritizes long-term vs. short-term exploitation strategies.

4.2.3. Deep Q-Network Training and Execution

The DQN training process in STROT follows the workflow illustrated in Figure 3 and consists of:

- State Representation: The detected vulnerabilities and system details are encoded and input into the Q-Network.
- Q-Value Prediction: The network estimates Q-values for each possible exploit action. The Q-value function follows the Bellman equation:

$$Q(s_t , a_t) = R_t + \gamma * max_{a'} Q(s_{t+1} , a')$$

- Exploit Selection *(ε-greedy policy)*: The model selects an exploit using:
  - Exploitation *(1-ε)*: Choose the exploit with the highest predicted success probability.
  - Exploration *(ε)*: Randomly select a new exploit to discover alternative attack paths.
- Attack Execution: The selected exploit is deployed, and its outcome is recorded.
- Reward Computation: The model updates the reward function based on attack success or failure.

- Target Network Update: The target network periodically synchronizes its weights with the Q-network to stabilize learning.
- Experience Replay: Past exploit attempts are stored and replayed to improve learning efficiency, preventing overfitting to recent attack outcomes.

This iterative training and feedback loop enables continuous optimization of exploit selection, ensuring that STROT dynamically adapts to evolving system defenses.

.2.4. Performance Benefits in STROT

By leveraging Deep Q-Learning, STROT achieves:

- Automated Exploit Mapping: Eliminates the need for manual selection, improving efficiency.
- Optimal Privilege Escalation Paths: Identifies the most effective exploit chains for rapid system compromise.
- Reduced Detection Risk: Learns to prioritize low-noise exploits, making attacks harder to detect.
- Adaptability to New Defenses: Unlike signature-based approaches, DQN continuously improves attack selection, making STROT resilient to evolving security measures.[1]

5. RESULTS

5.1. Deep Q-Network Performance Analysis

The performance of STROT was evaluated by executing various exploits on vulnerable target machines, including Metasploitable and Kioptrix. The model was tested under controlled conditions to measure its efficiency in identifying and executing exploits while optimizing attack strategies. The following subsections provide a detailed analysis of key performance metrics, showcasing the tool's effectiveness in vulnerability exploitation. The results are illustrated through various graphs, demonstrating the model's learning capability, accuracy, and efficiency in performing cyber-attacks.

5.1.1. Episode Reward Over Time

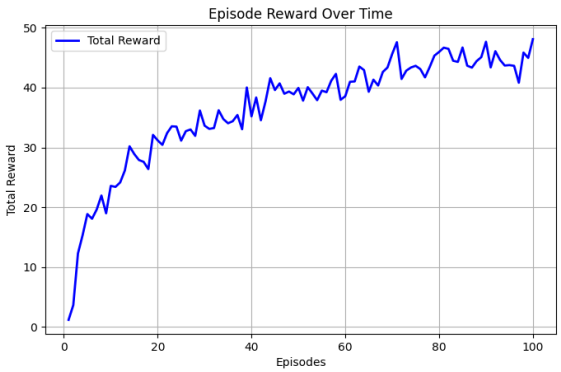

Fig - 4: Episode Reward Over Time Graph

The Episode Reward Over Time graph (Figure 4) demonstrates the learning progression of the Deep Q-Network (DQN) in STROT. The total reward per episode serves as an indicator of how well the model adapts to selecting optimal exploits over time. As seen in the graph, the reward starts low but steadily increases, signifying that the model is effectively learning from previous actions. The fluctuations in later episodes suggest adaptive exploration, ensuring that the model generalizes well across different attack scenarios. The upward trend confirms that STROT refines its strategy over time, successfully identifying high-impact exploits with greater confidence.

5.1.2. Q-Value Estimates Over Time

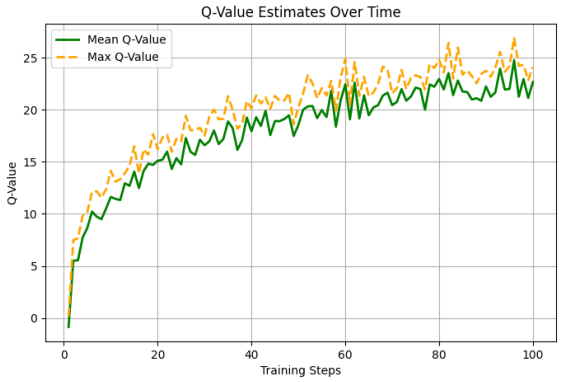

Fig - 5: Q-Value Estimates Over Time Graph

The Q-Value Estimates Over Time graph (Figure 5) illustrates the progression of the mean and maximum Q-values as the STROT model undergoes training. The Q-values represent the model's confidence in selecting optimal actions based on its learned policy. The steady increase in both mean and max Q-values indicates that the model is effectively improving its decision-making ability over time. The convergence towards higher values suggests that the DQN is successfully distinguishing

between effective and ineffective exploits, reinforcing optimal attack strategies. The occasional fluctuations reflect exploration behavior, ensuring adaptability in different attack scenarios. Overall, the upward trend validates the efficiency of STROT's reinforcement learning framework in cybersecurity exploitation.

5.1.3. Decision Latency Over Time

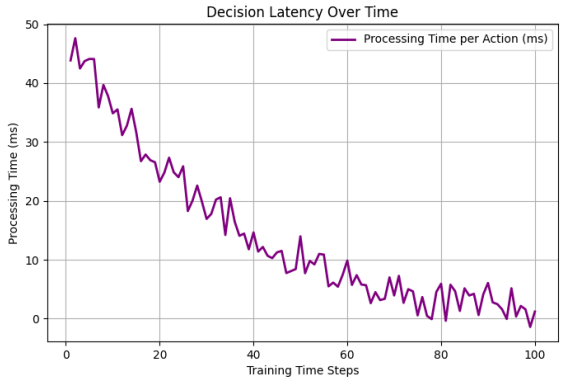

Fig - 6: Decision Latency Over Time Graph

The Decision Latency Over Time graph (Figure 6) showcases the reduction in processing time per action as the STROT model undergoes training. Initially, the decision latency is relatively high, indicating a longer inference time due to the model's lack of familiarity with optimal exploitation strategies. However, as training progresses, the latency significantly decreases, demonstrating improved efficiency in decision-making. This decline suggests that the model optimizes its action selection process, making faster and more confident exploitation attempts. The occasional fluctuations towards the later stages may be attributed to adaptive exploration mechanisms, ensuring robustness against diverse target environments. The overall downward trend highlights the model's increasing computational efficiency, making STROT a practical tool for real-world cybersecurity applications.

5.2. Real-World Performance Testing in Sandbox

To assess the real-world performance of STROT, we conducted controlled attack simulations on Metasploitable, a deliberately vulnerable Linux-based virtual machine designed for penetration testing. The testing process involved executing the STROT framework to autonomously identify vulnerabilities, map them to corresponding exploits, and attempt to gain root access. The evaluation focused on multiple aspects, including accuracy of vulnerability detection, exploit selection efficiency, and time taken to achieve root access. By leveraging Deep Q-Networks (DQN), STROT dynamically optimized its exploitation strategy, selecting the most effective attack path with minimal trial-and-error. The findings from these attack simulations are summarized in the table below.

5.2.1. Metasploitable Penetration Testing Result

Table - 1: Simulation Sandbox Description for Metasploitable

Attack Machine:	Kali 2024.4 ARM64
Attack Machine IP:	192.168.16.212
Attack Machine Virtualization Platform:	VMware Fusion Professional Version 13.6.3 (24585314)
Netmask:	255.255.255.0
Broadcast:	192.168.16.255
Network Configuration:	Bridged
Target Machine:	Metasploitable 2 x86
Target Machine IP:	192.168.16.228
Target Machine Virtualization Platform:	VirtualBox GUI Version 7.0.18 (r162988)
STROT Version:	STROT-CLI Stable Release 1.3.2

Table - 2: STROT-CLI Report

STROT Network Analysis Report:			
Target Machine IP:		192.168.16.228	
Detected OS:		Linux 2.6.9 - 2.6.33	
Running Service and Ports:			
PORT	STATUS	SERVICE	VERSION
21	Open	ftp	vsftpd 2.3.4
22	Open	ssh	OpenSSH 4.7p1
23	Open	telnet	Linux telnetd
25	Open	smtp	Postfix smtpd
53	Open	domain	ISC BIND
80	Open	http	Apache httpd
111	Open	rpcbind	2
139	Open	netbios-ssn	samba smbd

445	Open	netbios-ssn	samba smbd
512	Open	exec	netkit-rsh
513	Open	login	OpenBSD
514	Open	tcpwrapped	
1099	Open	java-rmi	GNU Classpath
1524	Open	bindshell	root shell
2049	Open	nfs	2-4
2121	Open	ftp	ProFTPD 1.3.1
3306	Open	mysql	MySQL 5.0.51a
5432	Open	postgresql	PostgreSQL DB
5900	Open	vnc	VNC
6667	Open	irc	unrealIRCd
8009	Open	ajp13	Apache Jserv
8180	Open	http	Apache Tomcat
STROT Intelligence Report:			
Chosen Service to Exploit:		ftp (vsftpd 2.3.4)	
Service Version:		vsftpd 2.3.4	
Reasoning:		vsftpd 2.3.4 Backdoor	
Exploit type:		Remote Backdoor Command Execution	
CVE:		CVE-2011-2523	
EDB-ID:		49757	
Attack Success/Failure:		Success	
Attack Feedback:		Successfully gained reverse shell access to the target machine with 1 tries exit (0)	
Time Spent by Exploit:		2760 ms	

6. CONCLUSION

In this research, we introduced STROT: Stealthy Tool for Root Oriented Tunneling, an advanced red teaming framework that automates the process of network reconnaissance, vulnerability analysis, and exploit selection using Deep Q-Networks (DQN). Traditional penetration testing tools require extensive manual effort to identify vulnerabilities, select appropriate exploits, and execute attacks while maintaining stealth. STROT overcomes these challenges by integrating network scanning, artificial intelligence-based exploit mapping, and an autonomous attack engine into a single streamlined framework.

Our methodology demonstrated how STROT efficiently scans networked systems using Scapy, Socket, and Nmap APIs, extracts critical vulnerability data, and utilizes Deep Q-Network-based reinforcement learning to determine the optimal exploit for achieving rapid privilege escalation. The integration of machine learning and dynamic attack planning significantly improves both the accuracy and efficiency of penetration testing, making it an invaluable tool for red teams, cybersecurity researchers, and ethical hackers.

We evaluated STROT in a controlled sandbox environment consisting of Kali Linux as the attack machine and Metasploitable 2 as the target system. The tool successfully identified vulnerable services, such as vsftpd 2.3.4, and autonomously selected the most effective exploit to gain system access. The results demonstrate that STROT not only accelerates attack execution but also optimizes stealth strategies to reduce detection risks, making it highly effective for real-world adversarial simulations.

7. FUTURE WORK

While STROT has shown significant advancements in autonomous exploitation, there remain opportunities for further refinement.

7.1. Expanding the Exploit Database

Enhancing STROT's exploit selection with real-time updates from exploit repositories (e.g., ExploitDB, Metasploit).

Adaptive Evasion Techniques – Implementing reinforcement learning-based evasion tactics to counter intrusion detection and prevention systems (IDS/IPS).

7.2. Multi-Target Attack Coordination

Extending the framework to orchestrate attacks across multiple nodes simultaneously for advanced red teaming.

7.3. Integration with Defensive Mechanisms

Using STROT as a defensive tool for cyber deception and adversary simulation in blue team environments.

In conclusion, STROT represents a paradigm shift in red teaming operations, demonstrating how artificial intelligence and machine learning can enhance offensive cybersecurity strategies. By leveraging autonomous decision-making for exploit selection and execution,

STROT significantly reduces the time, effort, and expertise required for penetration testing, paving the way for next-generation intelligent cybersecurity tools.

8. AVAILABILITY and LICENSING

The STROT framework is publicly available as an open-source project under the CODEXIST.dev organization. It is released under the GNU General Public License (GPL), allowing users to freely use, modify, and contribute to its development while ensuring compliance with open-source licensing standards.

- Organization: CODEXIST.dev
- Organization's Website: www.codexist.dev
- STROT Official Website: strot.codexist.dev
- GitHub Repository:

github.com/codexistdev/Project-STROT

The repository provides comprehensive documentation, source code, and contribution guidelines for developers and security researchers interested in enhancing, testing, or deploying STROT. Users can download the project, report issues, submit feature requests, and contribute to its continuous improvement.

REFERENCES

[1] Oh, S.H.; Kim, J.; Nah, J.H.; Park, J. Employing Deep Reinforcement Learning to Cyber-Attack Simulation for Enhancing Cybersecurity. Electronics 2024, 13, 555. [CrossRef]

[2] Enoch, S.Y.; Huang, Z.; Moon, C.Y.; Lee, D.; Ahn, M.K.; Kim, D.S. HARMer: Cyber-attacks automation and evaluation. IEEE Access 2020, 8, 129397–129414. [CrossRef]

[3] Bahrami, P.N.; Dehghantanha, A.; Dargahi, T.; Parizi, R.M.; Choo KK, R.; Javadi, H.H. Cyber kill chain-based taxonomy of advanced persistent threat actors: Analogy of tactics, techniques, and procedures. J. Inf. Process. Syst. 2019, 15, 865–889. [CrossRef]

[4] Barto, A.G.; Sutton, R.S.; Anderson, C.W. Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Trans. Syst. Man Cybern. 1983, 13, 834–846. [CrossRef]

[5] The MITRE Corporation. Ajax Security Team, The MITRE Corporation. 2016 Available online: https://attack.mitre.org/groups/G0130/ (accessed on 5 December 2022).

[6] Kumar, R.; Aggarwal, R.K.; Sharma, J.D. Energy analysis of a building using artificial neural network: A review. Energy Build. 2013, 65, 352–358. [CrossRef]

[7] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. Nature 2015, 518, 529–533. [CrossRef]

[8] J. Burton, I. Dubrawsky, V. Osipov, C. T. Baumrucker, and M. Sweeney, Eds., "Cisco enterprise IDS management," in Cisco Security Professional's Guide to Secure Intrusion Detection Systems. Burlington, NJ, USA: Burlington, 2003, ch. 10, pp. 429–479.

[9] R. Al-Shaer, M. Ahmed, and E. Al-Shaer, "Statistical learning of APT TTP chains from MITRE ATT&CK," in Proc. RSA Conf., 2018, pp. 1–2.

[10] Amazon. (2020). Amazon Elastic Compute Cloud EC2. Accessed: May 4, 2020. [Online]. Available: https://aws.amazon.com/ec2/

[11] A. Applebaum, D. Miller, B. Strom, C. Korban, and R. Wolf, "Intelligent, automated red team emulation," in Proc. 32nd Annu. Conf. Comput. Secur. Appl., Dec. 2016, pp. 363–373.

[12] D. L. Bergin, "Cyber-attack and defense simulation framework," J. Defense Model. Simul., Appl., Methodol., Technol., vol. 12, no. 4, pp. 383–392, Oct. 2015.

[13] M. Boddy, J. Gohde, T. Haigh, and S. Harp, "Course of action generation for cyber security using classical planning," in Proc. 15th Int. Conf. Int. Conf. Automated Planning Scheduling, 2005, pp. 12–21.

[14] Y. Cheng, J. Deng, J. Li, S. A. DeLoach, A. Singhal, and X. Ou, "Met-rics of security," in Cyber Defense and Situational Awareness. Cham, Switzerland: Springer, 2014, pp. 263–295.

[15] C. S. Choo, C. L. Chua, and S.-H.-V. Tay, "Automated red teaming: A proposed framework for military application," in Proc. 9th Annu. Conf. Genet. Evol. Comput. (GECCO), 2007, pp. 1936–1942.

[16] C. L. Chua, W. C. Sim, C. S. Choo, and V. Tay, "Automated red teaming: An objective-based data farming approach for red teaming," in Proc. Winter Simulation Conf., Dec. 2008, pp. 1456–1462.

[17] M. Cremonini and P. Martini, "Evaluating information security investments from attackers perspective: The return-on-attack (ROA)," in Proc. 4th Workshop Econ. Inf. Secur., Jun. 2005, pp. 1–3.

[18] F. M. Zennaro and L. Erdodi, "Modeling penetration testing with reinforcement learning using capture-the-flag challenges and tabular Q-learning," 2020, arXiv:2005.12632. [Online]. Available: http://arxiv.org/abs/2005.12632

[19] M. Zhang, L. Wang, S. Jajodia, and A. Singhal, "Network attack surface: Lifting the concept of attack surface to the network level for evaluating networks' resilience against zero-day attacks," IEEE Trans. Dependable Secure Comput., early access, Dec. 21, 2018, doi: 10.1109/TDSC.2018.2889086.

[20] N. Poolsappasit, R. Dewri, and I. Ray, "Dynamic security risk management using Bayesian attack graphs," IEEE Trans. Dependable Secure Comput., vol. 9, no. 1, pp. 61–74, Jan. 2012.

[21] C. W. Probst and R. R. Hansen, "An extensible analysable system model," Inf. Secur. Tech. Rep., vol. 13, no. 4, pp. 235–246, Nov. 2008.

[22] S. Randhawa, B. Turnbull, J. Yuen, and J. Dean, "Mission-centric auto- mated cyber red teaming," in Proc. 13th Int. Conf. Availability, Rel. Secur. (ARES), 2018, p. 1.

[23] T. Reed, R. G. Abbott, B. Anderson, K. Nauer, and C. Forsythe, "Simulation of workflow and threat characteristics for cyber security incident response teams," in Proc. Hum. Factors Ergonom. Soc. Annu. Meeting, vol. 58. Los Angeles, CA, USA: SAGE, 2014, pp. 427–431.

[24] C. Sarraute, O. Buffet, and J. Hoffmann, "Penetration testing==POMDP solving?" in Proc. Workshop Intell. Secur., Secur. Artif. Intell. (SecArt), 2011, pp. 1–8. [CrossRef]

[25] C. Sarraute, G. Richarte, and J. L. Obes, "An algorithm to find optimal attack paths in nondeterministic scenarios," in Proc. 4th ACM Workshop Secur. Artif. Intell. (AISec), 2011, pp. 71–80.

[26] B. Schneier, "Attack trees," Dr. Dobb's J., vol. 24, no. 12, pp. 21–29, 1999.

BIOGRAPHIES

Pratik Suhas Pawar
er.pratiksp@gmail.com
Computer programmer and deep learning expert, with multidisciplinary expertise in system engineering, system programming, and cybersecurity tool development. He played a pivotal role in the core development of STROT, focusing on its intelligence module, architecture, and system integration. A UG student in the Department of Computer Engineering at Savitribai Phule Pune University, Pune, India, he specializes in AI-driven security, low-level system design, and automation, bringing advanced machine learning techniques to offensive cybersecurity applications.

Shubham Pandurang Sakhare
er.shubhamsakhare@gmail.com
A cybersecurity practitioner, he played a key role in documenting, networking and sandbox creation, for the STROT project, ensuring a structured approach to its development. A UG student in the Department of Computer Engineering at Savitribai Phule Pune University, Pune, India, he is dedicated to ethical hacking, penetration testing, and cybersecurity research.

Vishnu Latish Nair
ervishnu00@gmail.com
Cybersecurity practitioner, specializing in virtualization, vulnerability analysis, and exploit identification, who contributed to the graphics and architectural overview of STROT. A UG student in the Department of Computer Engineering at Savitribai Phule Pune University, Pune, India, he focuses on penetration testing, network security, and secure system design, playing a key role in evaluating attack surfaces and refining the project's strategic approach.

Vishal Gandhar Puranik
vishalpura@gmail.com
Head, Department of Computer Engineering at Parvatibai Genba Moze College of Engineering, Wagholi, Pune, India, he has provided invaluable guidance and mentorship throughout the development of STROT. His expertise in computer engineering has played a crucial role in shaping the project, offering technical insights and strategic direction to enhance its effectiveness and real-world applicability.