Branch-Commit-Validate: A Git-Inspired Workflow for Autonomous Red-Team Agents

Chinmay Shringi[1][0009−0008−0325−7668], Alon Hillel-Tuch[1][0000−0002−2307−3486], and Sariya Rizwan[2][0009−0008−5312−162X]

[1] Department of Computer Engineering and Computer Science, New York University, Brooklyn, United States of America {cs7810,alon.h}@nyu.edu
[2] Lubin School of Business, Pace University, Brooklyn, United States of America sl00399n@pace.edu

Abstract. Red-team penetration testing is critical for uncovering vulnerabilities and strengthening defenses. Automating these complex engagements requires traversing vast, dynamic attack spaces and adjusting to real-time target changes, an area where existing search-based planners and deep reinforcement-learning (RL) agents struggle due to combinatorial explosion and prolonged training requirements. This paper presents Branch, Commit and Validate, a Git-inspired workflow that orchestrates specialized agents through parallel hypothesis exploration (Branch), metadata-rich recording of CPU-hours and operator-interaction time (Commit), and automated reliability and performance validation before integration (Validate). We intend to test the framework in a multi host, multi service simulated enterprise setting, gauging gains in decision delay, exploration efficiency, and validated coverage.

Keywords: validation frameworks · autonomous red teaming · version control · Penetration testing · Continuous integration (CI) · Deep reinforcement-learning · Multi-agent systems

1 Introduction

Red-team penetration testing provides organizations with proactive insights into security weaknesses before adversaries can exploit them. According to the 2023 CyberRisk Inc. survey, more than 80% of firms perform regular red-team exercises, each requiring approximately 120 human-hours and often exceeding planned durations due to manual pivot planning and reconnaissance [1] [2]. To mitigate these resource demands, researchers have introduced autonomous software agents - programs capable of perceiving network states, formulating attack plans, and executing exploits with minimal human input [3]. However, current automation techniques face three main obstacles. First, the number of potential attack paths increases exponentially with network size, causing search algorithms to stall and RL agents to demand hundreds of episodes to converge [4], [5]. Second, static attack graphs precompute a limited set of paths and cannot adapt when target systems reconfigure mid-engagement [6]. Third, while multi-agent

frameworks distribute tasks such as reconnaissance, exploitation, and privilege escalation, they often operate without formal coordination, resulting in redundant actions and conflicting outcomes [7], [8].

To address these limitations, we employ concepts from Git, a version-control, shared-data system with support for concurrent development via branches, context-aware commits, and merge-gated by CI testing. Our Branch-Commit-Validate lifecycle adapts these principles to autonomous red teaming: specialized Reconnaissance, Vulnerability Analysis, and Exploitation Agents generate parallel attack hypotheses (Branch); successful findings are committed along with CPU-hours and operator-interaction minutes to a central repository (Commit); and each commit undergoes automated reliability, performance, and coverage validation before merging (Validate). We will demonstrate that this approach ensures traceability, scalability, and quality gating in red-team automation.

2 Branch-Commit-Validate Workflow

In this section, we detail each phase of our workflow proposal and ground our definitions in existing literature. From a sequencing perspective, a Reconnaissance Agent (Recon Agent) initiates campaign setup by scanning the target network to enumerate hosts, open ports, and services, and spawning distinct exploration threads called branches. Vulnerability Analysis Agents operate on these branches in parallel, formulating targeted hypotheses (for example, "Host 10.0.0.42 running SSH (Secure Shell) 7.4 may harbor CVE-2018-15473") and executing tests. This parallel frontier exploration (advancing many sibling hypotheses concurrently) resembles Monte Carlo Tree Search techniques, which improve discovery in large decision trees through concurrent path simulations [9].

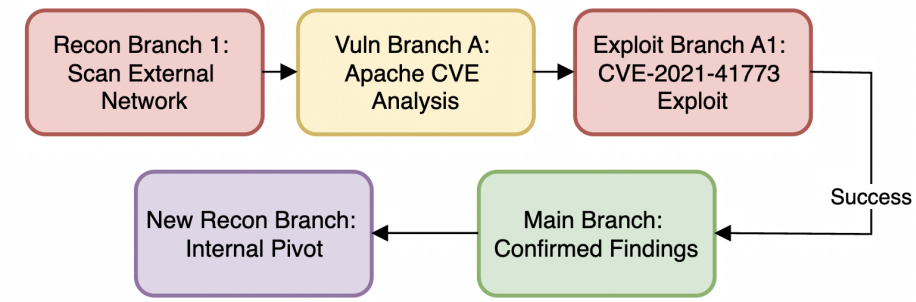

Fig. 1. Proposed Branch-Commit-Validate workflow architecture.

Fig. 1 illustrates the full Branch-Commit-Validate lifecycle: the Recon Agent (red) performs host discovery and port scanning, generating branches passed to

the Vulnerability Agent for Common Vulnerabilities and Exposures (CVE) analysis as well as attack graph construction. These branches are then extended to the Exploit Agent for exploit attempts and shell or data access. Validated commits from successful exploits populate the Main Branch (green). A customized Pivot Agent may be activated to start reconnaissance from the recently compromised host when a successful exploit has been verified and integrated into main, hence creating new recon branches further into the network. From this point, Pivot Agents (purple) launch new reconnaissance branches on compromised hosts.

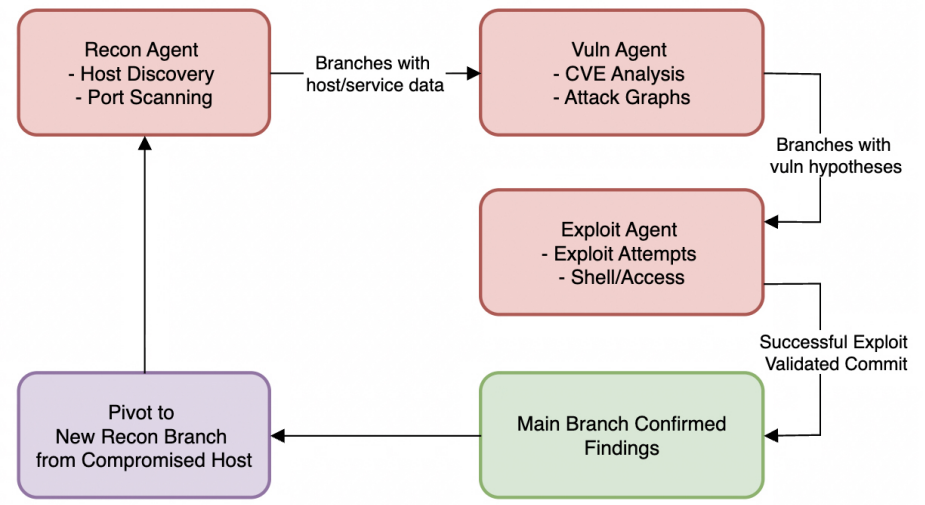

Fig. 2. Example of attack process during a penetration test.

Fig. 2 presents a concrete example: Recon Branch 1 scans the external network, leading Vuln Branch A to perform Apache CVE analysis and Exploit Branch A1 to successfully leverage CVE-2021-41773, its validated findings merge into the Main Branch and trigger a pivot, spawning a New Recon Branch for internal assessment. In contrast, Recon Branch 2 probes an internal server, Vuln Branch B tests SSH weak-credential hypotheses, and Exploit Branch B1's brute-force SSH attempt fails; this branch is pruned and excluded from the main campaign. To structure our Git-inspired process, we break the lifecycle into three sequential phases:

- **Branch:** Parallel hypothesis generation and exploration by specialized agents.
- **Commit:** Metadata-rich recording of confirmed findings, including CPU-hours and operator interaction.
- **Validate:** Automated checks for reliability, performance, and meaningful coverage gain before integration.

Collectively known as BCV, each phase is described below.

2.1 Branch: Parallel Hypothesis Exploration

The process begins with a Reconnaissance Agent performing active and passive scans to enumerate hosts, open ports, and services. For every discovered target (e.g., IP address and port combination), the Agent creates a new branch, spawning a Vulnerability Analysis Agent tasked with probing specific hypotheses such as "Host 10.0.0.42 running SSH 7.4 may be vulnerable to CVE-2018-15473." This parallel exploration strategy mirrors Monte Carlo Tree Search (MCTS) methodologies, which have demonstrated superior path discovery in large decision spaces by running concurrent simulations [9]. By dividing the search space among multiple branches, we alleviate the Bellman's "curse of dimensionality" that plagues monolithic and RL-based planners [4] [5].

2.2 Commit: Metadata-Enriched Recording

- **Action Trace:** An ordered log of commands and payloads deployed.
- **CPU-Hours:** Cumulative compute time consumed during branch execution.
- **Operator-Interaction Minutes:** Total human oversight time (e.g., prompt confirmations or manual adjustments).

This enhanced metadata collection builds upon best practices in collaborative security workflows, enabling reproducibility, auditability, and informed resource planning [10]. We provide a comprehensive view of automation efficiency and simplify comparisons among approaches by including human-effort measures in addition to computational expenses.

2.3 Validate: Automated Reliability And Performance Checks

Before a commit merges into the main campaign, a Validation Agent executes a three-pronged quality check. First, reliability validation replays the exploit sequence in an isolated environment to verify the consistency of shell access or payload effect. Second, performance validation measures response latency and enforces a configurable threshold (e.g., < 200 ms round-trip time) to ensure practical exploit execution. Third, coverage validation calculates the incremental reachability gain, requiring measurable incremental reachability gain (e.g., new subnets, hosts, or privilege domains) to ensure only strategically meaningful branches are integrated. Commits failing any criterion are automatically pruned or scheduled for re-examination, thereby maintaining campaign integrity and efficiency [11].

3 Related Work

Automation in penetration testing spans several paradigms. Attack-graph-based planners model network exploitation as a search problem, navigating graph nodes and edges representing hosts and vulnerabilities [12], [6]. Deep RL frameworks,

such as those by Hu et al., train agents through reward-driven exploration of scanning, exploitation, and lateral movement phases [4]. More recently, large language models (LLMs) power systems such as AutoAttacker, orchestrating summarization, planning, and action modules in iterative cycles to generate complex attack sequences [13]. While these methods advance the state of automation, they often lack integrated support for parallel branch management, resource-centric metrics, and gated integration. Our Branch-Commit-Validate workflow synthesizes these elements within a version-control-inspired model, offering enhanced scalability, auditability, and controlled human-in-the-loop collaboration.

4 Experimental Evaluation

We have developed a prototype implementation of the Branch-Commit-Validate framework and plan to evaluate it in a controlled simulated enterprise network environment. The testbed will consist of approximately 50 virtual hosts configured with a range of commonly exploited services (e.g., web servers, database engines, SSH). Our evaluation will benchmark BCV against a baseline monolithic penetration testing planner by comparing metrics such as the number of unique vulnerabilities confirmed, the reduction in redundant scanning and exploitation attempts, and the average time between commit creation and validation. We intend to run multiple independent trials and apply statistical tests (e.g., paired t-tests) to assess significance. These experiments will help quantify BCV's effectiveness in improving vulnerability coverage, reducing operational overhead, and accelerating the confirmation of findings.

5 Conclusion

We have presented BCV, a novel application of distributed version control and CI paradigms to autonomous red-team operations. By enabling parallel hypothesis exploration, capturing both computational and human-effort metrics, and enforcing automated quality gates, our framework delivers improved coverage, efficiency, and reliability. Future work will explore adaptive branch-prioritization heuristics based on commit metrics, integration of sandboxed CI pipelines for safe exploit replay, and scaling evaluations to larger, real-world network topologies. We also plan to examine the trade-offs between automation depth and operator oversight by analyzing detailed interaction logs.

Acknowledgments. This research was conducted as part of ongoing work in autonomous systems and cybersecurity at New York University and Pace University.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Ponemon Institute and Bishop Fox: The State of Offensive Security: 2023 Benchmarking Report. Ponemon Industry Report (2023)
2. Chu, G., Lisitsa, A.: Poster: Agent-based (BDI) modeling for automation of penetration testing. In: 2018 16th Annual Conference on Privacy, Security and Trust (PST), pp. 1–2. IEEE, Belfast (2018). https://doi.org/10.1109/PST.2018.8514211
3. Balaji, P.G., Srinivasan, D.: An Introduction to Multi-Agent Systems. In: Srinivasan, D., Jain, L.C. (eds.) Innovations in Multi-Agent Systems and Applications - 1. Studies in Computational Intelligence, vol 310. Springer, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14435-6_1
4. Hu, Z., Beuran, R., Tan, Y.: Automated Penetration Testing Using Deep Reinforcement Learning. In: 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), pp. 2–10. IEEE, Genoa (2020). https://doi.org/10.1109/EuroSPW51379.2020.00010
5. Abbass, H., Bender, A., Gaidow, S., Whitbread, P.: Computational Red Teaming: Past, Present and Future. IEEE Computational Intelligence Magazine **6**(1), 30–42 (2011). https://doi.org/10.1109/MCI.2010.939578
6. Konsta, A., Lafuente, A.L., Spiga, B., Dragoni, N.: Survey: Automatic generation of attack trees and attack graphs. Computers & Security **137**, 103602 (2024). https://doi.org/10.1016/j.cose.2023.103602
7. Ghanem, M.C., Chen, T.M., Nepomuceno, E.G.: Hierarchical reinforcement learning for efficient and effective automated penetration testing of large networks. J Intell Inf Syst **60**, 281–303 (2023). https://doi.org/10.1007/s10844-022-00738-0
8. Ghanem, M., Chen, T., Ferrag, M.A., Kettouche, M.: ESASCF: Expertise Extraction, Generalization and Reply Framework for Optimized Automation of Network Security Compliance. IEEE Access (2023). https://doi.org/10.1109/ACCESS.2023.3332834
9. Browne, C.B., et al.: A Survey of Monte Carlo Tree Search Methods. IEEE Transactions on Computational Intelligence and AI in Games **4**(1), 1–43 (2012). https://doi.org/10.1109/TCIAIG.2012.2186810
10. Burger, E.W., Goodman, M.D., Kampanakis, P., Zhu, K.A.: Taxonomy Model for Cyber Threat Intelligence Information Exchange Technologies. In: Proceedings of the 2014 ACM Workshop on Information Sharing & Collaborative Security (WISCS '14), pp. 51–60. ACM, New York (2014). https://doi.org/10.1145/2663876.2663883
11. Avgerinos, T., Cha, S.K., Rebert, A., Schwartz, E.J., Woo, M., Brumley, D.: Automatic exploit generation. Commun. ACM **57**(2), 74–84 (2014). https://doi.org/10.1145/2560217.2560219
12. Chen, Z.; Kang, F.; Xiong, X.; Shu, H. A Survey on Penetration Path Planning in Automated Penetration Testing. Appl. Sci. **14**, 8355 (2024). https://doi.org/10.3390/app14188355
13. Xu, J., Stokes, J.W., McDonald, G., Bai, X., Marshall, D., Wang, S., Swaminathan, A., Li, Z.: AutoAttacker: A Large Language Model Guided System to Implement Automatic Cyber-attacks. arXiv preprint arXiv:2403.01038 (2024)
14. Skandylas, C., Asplund, M.: Automated penetration testing: Formalization and realization. Computers & Security **155**, 104454 (2025). https://doi.org/10.1016/j.cose.2025.104454
15. Scarfone, K., Souppaya, M., Cody, A., Orebaugh, A.: Technical Guide to Information Security Testing and Assessment. NIST Special Publication 800-115, National Institute of Standards and Technology (2008). https://doi.org/10.6028/NIST.SP.800-115