International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 11 | Nov - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

TOGEPI: A Python-Based Host Intrusion Detection and Automated Firewall Response System with Linux and Windows Implementations

Anshool Kumar, Hemant Pise, Om Adhau and V. V. Panchbhai
Department of Electronics and Telecommunication Engineering, Priyadarshini College of Engineering,
Nagpur, India

Abstract—Operating systems record failed login attempts in their authentication logs, yet the native logging services do nothing to stop those attempts at the network boundary. This paper presents TOGEPI (Threat-Oriented Guard Engine for Protective Infrastructure), a lightweight Python prototype that joins four tasks into one pipeline: watching security logs, detecting brute-force behavior, extracting and validating the source IPv4 address, and blocking that address through the host firewall. Two implementations are described. The Linux reference version reads authentication logs and issues iptables rules, while the Windows console version offers a menu-driven interface, a synthetic event simulator and Windows Advanced Firewall commands. Detection combines keyword matching with a state-aware sliding window set to five failed attempts within 30 seconds. According to the project thesis, all ten functional test cases passed, and five controlled trials gave a mean detection-to-mitigation time of 0.537 s. Idle monitoring used 0.05%–0.15% CPU and 14.2 MB of RAM in the test environment. We also trace a Windows BLOCK_FAILED status to a code-level cause: the state-deduplication check returned a generic false value that the caller read as a firewall failure. The results are laboratory-scale evidence only, and we treat successful rule creation and independently verified packet dropping as two separate outcomes.

Keywords—host-based intrusion detection, brute-force mitigation, automated incident response, log analytics, iptables, Windows Firewall.

I. INTRODUCTION

Hosts that sit on public networks or enterprise LANs are probed for weak credentials around the clock by automated tools [1]. The main targets are Secure Shell (SSH) services on UNIX/Linux servers and the remote-management interfaces of Windows, where scripted attackers and botnets keep trying password guesses [1,2]. In a large-scale measurement study, Singh et al. [1] found that SSH brute-force campaigns form a constant background load on production servers and generate millions of login probes.

Every authentication attempt is written to a system log: /var/log/auth.log on Debian-based Linux distributions, or the centralized event log on Windows [1,3]. An entry carries a timestamp, the process identifier of the daemon, the username that was tried and the originating IPv4 address, and Unix-like systems commonly format such messages following the syslog convention [4]. These logs are nevertheless passive records. They describe events in fine detail, but nothing in them feeds back into network-layer filtering rules [1,5]. In the usual administrative routine, an operator has to read the log, recognize a brute-force pattern, pull out the attacker's address and type in the firewall commands by hand [2]. Where no Security Operations Center (SOC) staff are present around the clock, as in small offices, teaching laboratories and unattended edge devices, an automated attacker can try thousands of credentials before anyone notices [1].

NIST Special Publications 800-92 (Guide to Computer Security Log Management) and 800-94 (Guide to Intrusion Detection and Prevention Systems) both argue that organizations should move away from passive log archiving and toward real-time analysis of event streams, with automated responses that have low latency [5,6].

TOGEPI addresses this gap for a single host [7]. It was built as an undergraduate honors engineering project and automates the whole chain: it follows the security logs, recognizes suspicious authentication patterns, counts repeated failures against a time threshold, extracts and validates the source IP address, asks the operating-system firewall to block it, and writes a structured audit record.

This paper makes three contributions. (1) It describes a modular host-security architecture that runs on both Linux and Windows. (2) It reports the functional verification results together with a code-level analysis of the observed BLOCK_FAILED status. (3) It presents controlled latency and resource-use measurements and states plainly what the current evidence does not show.

II. RELATED WORK

A. Brute-Force Attacks and Authentication Monitoring

Password guessing is cataloged as MITRE ATT&CK technique T1110 (Brute Force) [2]. Singh et al. [1] studied four years of SSH authentication attempts collected from more than 500 servers and used the data to compare detection methods. They proposed Dictionary-Based Blocking (DBB), which exploits what is known about the usernames attackers target during the guessing phase. By their account, DBB blocked 99.5% of attacks in their evaluation, whereas a standard rate-based approach blocked 66.1%. Their analysis also showed that attacks which space out consecutive attempts slip past rate-based limiting, and that tracking failed authentication events remained a dependable signal for host-based brute-force detection [1]. Hamza and Al-Janabi [8] surveyed machine-learning methods for detecting brute-force attacks on SSH and FTP. They noted that most of the approaches rest on the same observation: automated attacks pack a high density of failures into a short period.

B. Security-Log Analysis and Anomaly Detection

Bridges et al. [9] surveyed host-based intrusion detection systems (HIDS) and observed that host logs expose decrypted application activity that network packet sniffers cannot see. That advantage places heavy demands on parsers, which must cope with large volumes of semi-structured text. He et al. [10] addressed this with Drain, an online log-parsing algorithm that uses a fixed-depth tree to extract message templates in linear time. Bhanage et al. [11] reviewed IT event processing with emphasis on pre-processing pipelines, machine-learning anomaly detectors and automated responses to operational failures. Their review underlines how much downstream analytics depend on sanitized data and clean token extraction.

C. Rule-Based Versus Learning-Based Detection

Recent work has applied machine learning and deep neural networks to log parsing and anomaly detection. Van Ede et al. [12] proposed DEEPCASE, a semi-supervised framework for contextual event analysis that arranges security events into sequences to help analysts. No et al. [13] introduced a training-free method for log anomaly detection built on pre-trained language models, and Landauer et al. [14] surveyed deep-learning approaches (RNNs, LSTMs and Transformers) for log data. Learning-based detectors tend to produce more false positives, are not easy to adapt as system behavior drifts, and need considerable memory and compute time for inference. Rule-based matching, on the other hand, is fast, computationally transparent and gives detection logic that a human can read [11].

D. Automated Intrusion Response and Practical Tooling

An Intrusion Response System (IRS) decides which action to take once an intrusion alert fires, and then carries it out. Rezapour et al. [15] produced a systematic mapping study of IRSs that contrasts passive alerting with packet-filtering responses and points to false-positive self-lockout as a key operational risk. Hughes et al. [16] used model-free reinforcement learning to choose response actions that have minimal side effects on legitimate system use.

Fail2Ban is an open-source tool, used in production, that performs the monitoring and rule-injection duties of an IRS. It tails log files, applies regular expressions to extract relevant events, counts attempts per source, and writes iptables rules to block offending hosts [17]. It covers many services, but its configuration involves a layered hierarchy of files, external packaging dependencies and Linux-specific daemon components.

E. Positioning of TOGEPI

Table I gives a qualitative comparison of the reviewed approaches with TOGEPI. TOGEPI does not claim to outdo mature frameworks such as Fail2Ban in the range of services covered, and it does not introduce a new detection algorithm. Its contribution is a modular Python prototype that is easy to access and has no external dependencies, and that brings together log streaming, state-aware thresholding of attempts, firewall interaction on two platforms and a built-in synthetic attack simulator for controlled laboratory evaluation.

Table I. Qualitative comparison of related approaches.

Approach	Telemetry / method	Response	Positioning
Singh et al. [1]	SSH logs; dictionary-based patterns	Automated in study	Empirical brute-force telemetry
Drain [10]	Online log parsing	Parser only	Structured log-template extraction
DEEPCASE [12]	Contextual security events	Analyst support	Contextual triage
Fail2Ban [17]	Host/service logs and filters	Firewall jails	Mature Linux-oriented tooling

Ap-proach	Telemetry / method	Response	Positioning
TOGEPI	Keyword matching + sliding win-dow	iptables / netsh inte-gration	Modular proto-type with simu-lator

In summary, the reviewed studies show that log parsing, contextual analysis and established tools like Fail2Ban each cover part of the host-level detect-and-respond prob-lem. Learning-based methods can give richer context, whereas transparent rule-based methods are easier to in-spect and to deploy in lightweight settings. TOGEPI there-fore positions itself as a compact educational prototype that combines log monitoring, per-source thresholding, IPv4 validation, firewall command integration on two plat-forms and controlled event simulation. It is not meant to replace mature IDS/IPS platforms.

F. Platform Implementations

Table II summarizes how the Linux and Windows imple-mentations differ.

Table II. Documented implementation differences.

Dimen-sion	Linux reference implementation	Windows console implementation
Operating system	Ubuntu Server 22.04 LTS (re-ported)	Windows 11 Pro-fessional (reported)
Input	/var/log/auth.log or a specified text file	Monitored log file and simulator
Firewall	iptables / Netfilter DROP rule	Windows Ad-vanced Firewall via netsh
Interface	Command-line ar-guments	Interactive num-bered console menu
Packaging	Python scripts	Standalone execut-able built with PyInstaller

III. METHODOLOGY

Fig. 1 shows the overall architecture of TOGEPI, and Fig. 2 shows how the software modules interact and how data moves through them at run time.

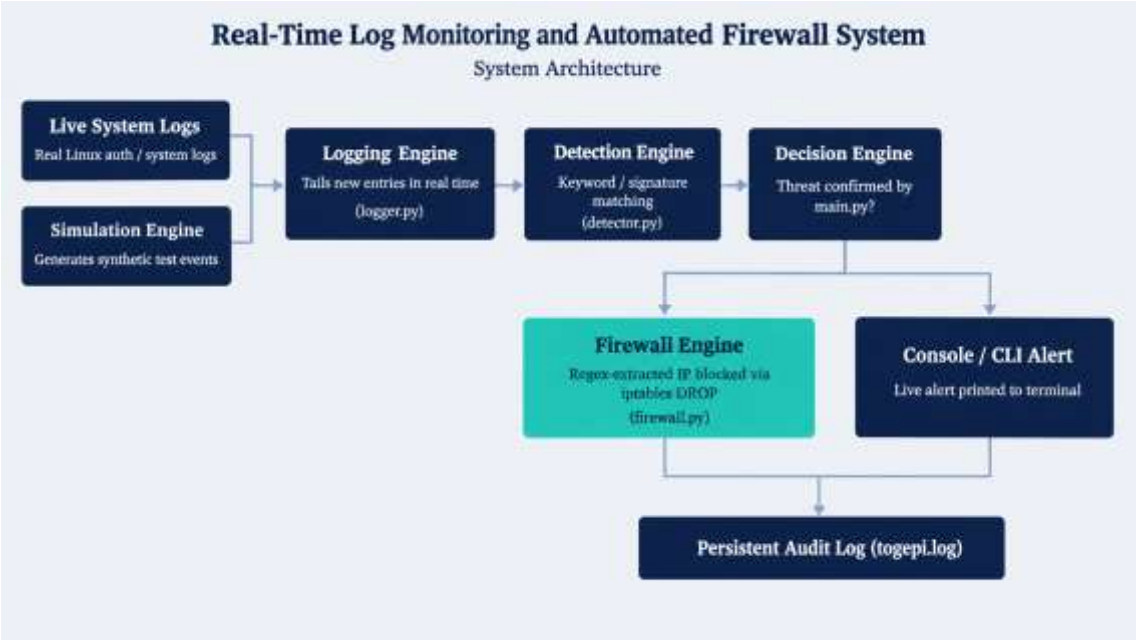


Fig. 1. Real-time log monitoring and automated firewall response system.



Fig. 2. TOGEPI software-module interaction and runtime data flow.

A. Input Ingestion and File Handling

The Logging Engine follows a log file continuously without placing a heavy load on the CPU. In the Linux reference implementation, logger.py works as a generator that yields each new line. When no new data is available it sleeps for a polling interval (0.5 s by default), and it reopens the file if truncation or log rotation invalidates the file descriptor. The Windows console build instead uses an offset-seeking routine in logger.py. It resumes from the last saved offset and records the new position after reading, that is, offset_new = f.tell() after f.seek(offset_last). Because only the appended text is read, the program never re-reads multi-megabyte files on each polling cycle.

B. Detection Logic and Attempt Tracking

Detection is based on screening each line against several keyword sets. Incoming text is converted to lowercase and compared with three predefined dictionaries:

- Authentication-failure keywords: failed password, login failed, authentication failed, multiple login failures, invalid password.
- Traffic/DoS keywords (Windows v2.0): high request rate, excessive requests, traffic spike, possible dos.
- Suspicious keywords: unauthorized, attack, denied, malware, exploit.

The Windows build, TOGEPI-Windows, adds state tracking with a temporal sliding window. A security event is written as $E_i = (t_i, IP)$, where t_i is its timestamp. A dictionary named failed_attempts maps each source address to the timestamps of its matching failures. A brute-force threat is declared when the number of timestamps for one address inside the window reaches the threshold:

$$|\{ t_j \in F[IP] : t_i - t_j \leq W \}| \geq N \qquad (1)$$

Here W = 30 s is the window length and N = 5 is the threshold.

C. IPv4 Extraction, Semantic Validation, and Sanitization

Firewall commands need well-formed IP strings, so raw log text goes through deterministic extraction followed by semantic validation. Extraction uses a regular expression that picks up the first dot-decimal sequence in the log entry, that is, four groups of one to three digits separated by dots. Validation, implemented in is_valid_ip, splits the extracted string into octets, checks that exactly four numeric segments are present, and requires each value o_k to satisfy

$$0 \leq o_k \leq 255, \quad k = 1, 2, 3, 4 \qquad (2)$$

A value such as 999.10.10.10 is therefore rejected before any firewall command is built. Trusted management addresses, by default only the loopback address 127.0.0.1, are whitelisted and skipped, so that local maintenance sessions can never trigger an automatic block.

D. Cross-Platform Firewall Execution

The Firewall Engine converts a blocking decision into an operating-system command. It calls subprocess.run() without a shell (shell=False), which removes the risk of shell-metacharacter injection. The two commands are:

- Linux Netfilter: sudo iptables -A INPUT -s <IP> -j DROP [18].
- Windows Advanced Firewall: netsh advfirewall firewall add rule name=TOGEPI_BLOCK_<IP> dir=in action=block remoteip=<IP>.

To avoid repeated firewall calls for the same address, TOGEPI keeps an in-memory set called blocked_ips. When an incoming IP is already in this set, the command is skipped and the message [FIREWALL] IP already blocked is logged.

E. Security Event Simulator

For safe verification without probing real networks, simulator.py generates synthetic security events in four profiles:

- Normal Activity: benign events such as successful user logins and health checks.
- Brute-Force Simulation: six consecutive failed-login lines from test IP 192.168.1.150, one second apart.
- DoS-like Simulation: a burst of rapid request-spike events from test IP 192.168.1.200.
- Mixed Security Test: normal and suspicious lines interleaved, to check how the detector copes with varied input.

Both test addresses come from the private 192.168.0.0/16 range; the blocks reserved for documentation [19] would be an alternative when logs are to be shared publicly. The complete decision flow is shown in Fig. 3.

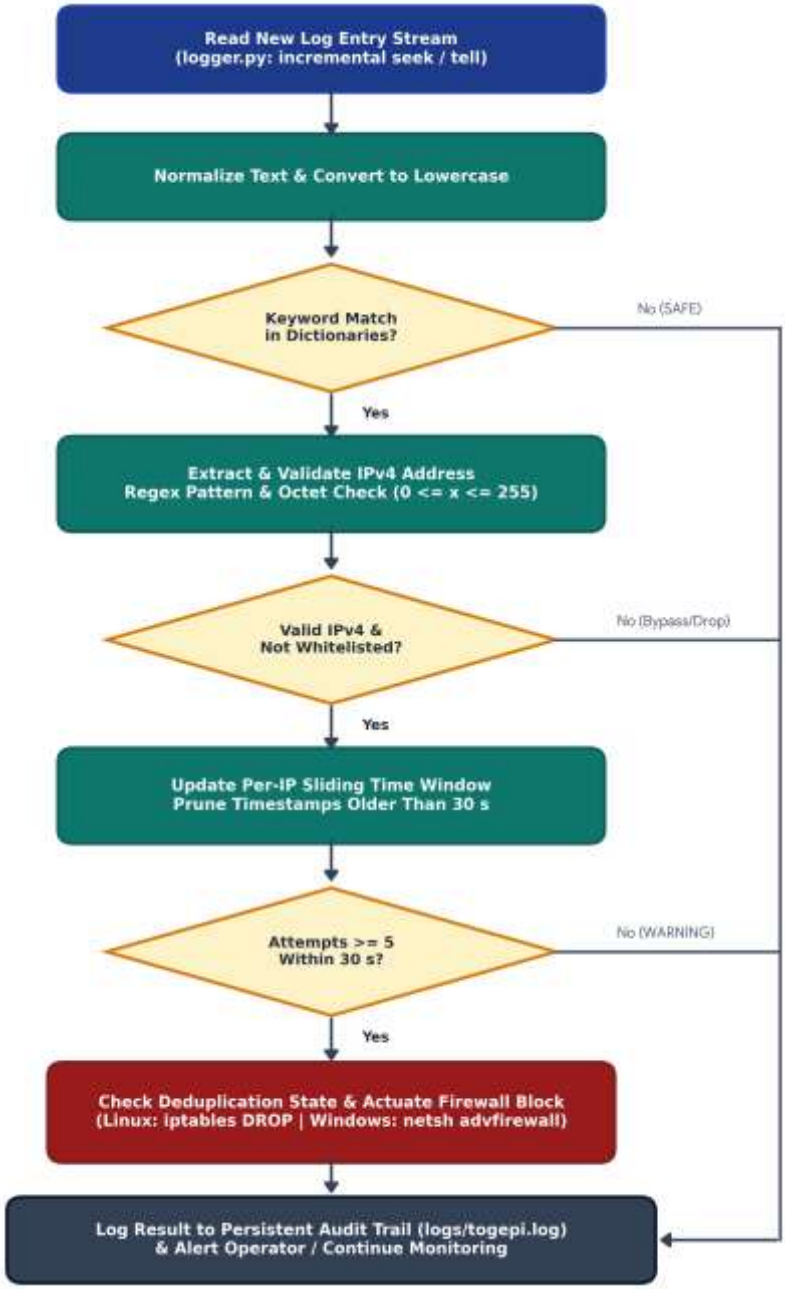


Fig. 3. Threat-detection and automated-response flowchart.

IV. RESULTS AND DISCUSSION

A. Functional Verification

The project thesis [7] reports ten functional test cases. They cover sub-threshold events, threshold triggering, duplicate suppression, independent tracking of separate sources, benign events, whitelisting, malformed IPv4 rejection, log lines without an IP token, keyword recognition and log rotation. All ten are reported as PASS (Table III).

Table III. Functional test cases and reported outcomes.

Test ID	Scenario	Reported outcome
TC-01	Four failures from one source; below threshold	PASS
TC-02	Fifth failure within 30 s	PASS
TC-03	Repeated failures after block; duplicate suppression	PASS
TC-04	Independent counters for two source IPs	PASS
TC-05	Benign log events	PASS
TC-06	Whitelisted loopback address	PASS
TC-07	Malformed IPv4 address (e.g., 999.1.1.1)	PASS
TC-08	Log line without an IP token	PASS
TC-09	"Invalid user" keyword match	PASS
TC-10	Log rotation / file recreation	PASS

B. Latency and Resource Use

Latency was measured from the commit of the fifth authentication-failure log line to the completion of the firewall subprocess. Table IV lists the five values reported in the thesis; their mean is 0.5372 s, rounded to 0.537 s.

Table IV. Detection-to-mitigation latency in five controlled trials.

Trial	Latency (s)	Reported status
Trial 1	0.522	PASS / sub-second
Trial 2	0.611	PASS / sub-second
Trial 3	0.478	PASS / sub-second
Trial 4	0.545	PASS / sub-second
Trial 5	0.530	PASS / sub-second
Mean	0.537	Sub-second

Table V gives the host resource profile reported in the thesis for four workloads.

Table V. Host resource profile reported in the thesis.

Workload	CPU utilization	RAM (script)	RAM (EXE)
Idle monitoring	0.05%–0.15%	14.2 MB	18.5 MB
Routine benign ingestion	0.20%–0.40%	14.5 MB	18.8 MB
Simulated brute-force burst	1.20%–1.80%	15.1 MB	19.4 MB
High-volume log flood (1000 lines/s)	2.40%–3.50%	16.8 MB	21.2 MB

These trials point to sub-second detection-to-mitigation completion and modest resource use in the stated test environment. They should nonetheless be read as prototype-level results. Finishing the subprocess and finding the rule in the firewall do not, on their own, prove that packets are dropped end to end, and the observed BLOCK_FAILED status needs the code-level explanation given next.

C. Root Cause Analysis of the Observed BLOCK_FAILED Outcome

In the elevated PowerShell trial, the audit log recorded Action=BLOCK_FAILED. Inspection of firewall.py and main.py shows how this status can arise: the duplicate-IP check returns False, and the caller treats every False return as a firewall execution failure.

Return code versus deduplication value. In firewall.py, block_ip(ip) returns a boolean. If the address is already in blocked_ips, the function logs [FIREWALL] IP already blocked: <IP> and returns False so that the operating-system command is not run again.

Alert dispatching in main.py. The calling code handles the result as follows:

```
blocked = firewall.block_ip(source_ip)
if blocked:
    ...
else:
    logger.write_alert(f"Threat={threat_type} | Source IP={source_ip} | Action=BLOCK_FAILED")
```

Because block_ip() returned False when it recognized a duplicate, main.py read the value as a failed command and wrote Action=BLOCK_FAILED to the audit stream, although the address was already in the firewall table.

Attempt-counter accumulation. The console banner shows Attempts: 4316. In the v2.0 console build, analyze_log() appends a timestamp to failed_attempts[IP] for every failure line and does not reset the counter after a block. Under continuous high-rate simulated probing the counter therefore rises without stopping, which shows that the data

structure kept working under high-volume event sequences.

D. Rule Creation Versus Verified Packet Dropping

When an IP address shows up in the SHOW BLOCKED IPs menu of TOGEPI, a rule whose name begins with TOGEPI_BLOCK_ has been read back from the operating-system firewall table. That confirms the rule exists, but not that traffic is being filtered. Whether packets are actually dropped depends on rule precedence, default profile policies and network-adapter bindings. Establishing real filtering would need independent network probes showing that active TCP connections are cut and later SYN packets are dropped.

The console record shows a brute-force alert for 192.168.1.150, then an "IP already blocked" message, then an Action=BLOCK_FAILED audit entry. That sequence supports the duplicate-state explanation; it does not prove that packets from that address were dropped.

Four things need to be kept apart: a detection alert, an attempted firewall command, successful rule creation, and independently verified packet dropping. The reported latency measures subprocess completion only, so it cannot by itself show that traffic was dropped end to end. The thesis therefore offers a controlled prototype evaluation, not evidence from internet-scale or production deployment.

V. CONCLUSION AND FUTURE SCOPE

A. Conclusion

TOGEPI is a lightweight host-security tool that links live log monitoring directly to firewall mitigation on both Linux and Windows. It combines log tailing, multi-pattern keyword scanning, a sliding-window frequency counter and IPv4 syntax checking into a detection-and-mitigation chain that runs on ordinary inputs. Controlled experiments tested whether the system behaves as intended, and the passing test cases indicate that most of the algorithms were implemented correctly. The mean time from detection to mitigation was 0.537 s, and idle monitoring used between 0.05% and 0.15% of the CPU. The analysis of the BLOCK_FAILED case clarifies how the host firewall path behaves, but confirming that mitigation really works would need further testing in realistic environments, together with packet-drop statistics.

B. Limitations

- Signature dependency. Detection is limited to the implemented keyword patterns. Attacks that avoid standard syslog formats, or that target services other than authentication, pass the detector unnoticed.
- No dynamic unblocking. Blocked addresses stay in the firewall table until someone removes them by hand through the console, so a legitimate user who mistypes a password may be locked out permanently.
- Distributed-attack blind spot. The threshold (N = 5) is applied per source IP. A botnet that spreads its probes across hundreds of addresses, each making only one or two attempts, can stay below it.
- IPv4 only. The regular expression and socket validation handle only 32-bit dot-decimal notation, so IPv6 addresses are ignored.

C. Future Scope

Planned work extends the prototype in several directions:

- Dynamic lease expiration (leaky-bucket model). Rules would be removed automatically after a configurable period (for example 3,600 s), with the penalty period doubling for repeat offenders.
- Machine-learning anomaly scoring. A lightweight statistical or unsupervised classifier, such as an Isolation Forest or an autoencoder, could act as a second-tier detector for slow, distributed brute-force campaigns.
- IPv6 and modern firewall tooling. Regex parsing would be extended to colon-hexadecimal notation, and the Linux backend would migrate to nftables syntax.
- Decentralized telemetry sharing. A lightweight gossip protocol could let edge nodes broadcast verified attacker IPs to neighboring hosts, immunizing local clusters in advance.
- Clearer status reporting and verification. Returning a distinct status for already-blocked addresses would remove the misleading BLOCK_FAILED entries analyzed in Section IV-C, and independent packet-level probes would close the gap discussed in Section IV-D.

REFERENCES

[1] S. K. Singh, S. Gautam, C. Cartier, S. Patil, and R. Ricci, "Where the Wild Things Are: Brute-Force SSH Attacks in the Wild and How to Stop Them," in Proc. 21st USENIX Symp. Networked Systems Design and Implementation (NSDI 24), Santa Clara, CA, USA, Apr. 2024, pp. 1731–1750. [Online]. Available: https://www.usenix.org/conference/nsdi24/presentation/singh-sachin

[2] MITRE ATT&CK, "Brute Force (T1110)," Enterprise Matrix. [Online]. Available: https://attack.mitre.org/techniques/T1110/. Accessed: Oct. 21, 2026.

[3] OpenBSD Project, "sshd(8) — OpenSSH SSH daemon," OpenBSD manual pages. [Online]. Available: https://man.openbsd.org/sshd.8. Accessed: Oct. 21, 2026.

[4] R. Gerhards, "The Syslog Protocol," RFC 5424, Internet Engineering Task Force, Mar. 2009. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5424/

[5] K. Kent and M. Souppaya, "Guide to Computer Security Log Management," NIST Special Publication 800-92, National Institute of Standards and Technology, Gaithersburg, MD, USA, Sept. 2006, doi: 10.6028/NIST.SP.800-92.

[6] K. Scarfone and P. Mell, "Guide to Intrusion Detection and Prevention Systems (IDPS)," NIST Special Publication 800-94, National Institute of Standards and Technology, Gaithersburg, MD, USA, Feb. 2007, doi: 10.6028/NIST.SP.800-94.

[7] A. Kumar, H. Pise, and O. Adhau, "Threat-Oriented Guard Engine for Protective Infrastructure (TOGEPI)," B.Tech (Honors) Thesis, Dept. Electron. & Telecommun. Eng., Priyadarshini College of Engineering, Nagpur, India, Oct. 2026.

[8] A. A. Hamza and R. J. S. Al-Janabi, "Detecting brute force attacks on SSH and FTP protocol using machine learning: A survey," J. Al-Qadisiyah Comput. Sci. Math., vol. 16, no. 1, pp. Comp. 21–31, 2024, doi: 10.29304/jqcsm.2024.16.11432.

[9] R. A. Bridges, T. R. Glass-Vanderlan, M. D. Iannacone, M. S. Vincent, and Q. Chen, "A survey of intrusion detection systems leveraging host data," ACM Comput. Surv., vol. 52, no. 6, Art. no. 128, 2019, doi: 10.1145/3344382.

[10]P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in Proc. 2017 IEEE 24th Int. Conf. Web Services (ICWS), Honolulu, HI, USA, 2017, pp. 33–40, doi: 10.1109/ICWS.2017.13.

[11]D. A. Bhanage, A. V. Pawar, and K. Kotecha, "IT infrastructure anomaly detection and failure handling: A systematic literature review focusing on datasets, log preprocessing, machine and deep learning approaches and automated tool," IEEE Access, vol. 9, pp. 156392–156421, 2021, doi: 10.1109/ACCESS.2021.3128283.

[12]T. van Ede et al., "DEEPCASE: Semi-supervised contextual analysis of security events," in Proc. 2022 IEEE Symp. Security and Privacy (SP), San Francisco, CA, USA, 2022, pp. 522–539, doi: 10.1109/SP46214.2022.9833671.

[13]G. No, Y. Lee, H. Kang, and P. Kang, "Training-free retrieval-based log anomaly detection with pre-trained language model considering token-level information," Eng. Appl. Artif. Intell., vol. 133, Art. no. 108613, 2024, doi: 10.1016/j.engappai.2024.108613.

[14]M. Landauer, S. Onder, F. Skopik, and M. Wurzenberger, "Deep learning for anomaly detection in log data: A survey," Mach. Learn. Appl., vol. 12, Art. no. 100470, 2023, doi: 10.1016/j.mlwa.2023.100470.

[15]A. Rezapour, M. GhasemiGol, and D. Takabi, "A Systematic Mapping Study on Intrusion Response Systems," IEEE Access, vol. 12, pp. 46524–46550, 2024, doi: 10.1109/ACCESS.2024.3381998.

[16]K. Hughes, K. McLaughlin, and S. Sezer, "A model-free approach to intrusion response systems," J. Inf. Secur. Appl., vol. 66, Art. no. 103150, 2022, doi: 10.1016/j.jisa.2022.103150.

[17]Fail2Ban Project, "Fail2Ban: A log-parsing application that protects computer servers from brute-force attacks," Official Documentation, 2024. [Online]. Available: https://www.fail2ban.org/

[18]Netfilter Project, "iptables(8): administration tool for IPv4 packet filtering and NAT," Linux manual page, 2023. [Online]. Available: https://netfilter.org/

[19]J. Arkko, M. Cotton, and L. Vegoda, "IPv4 Address Blocks Reserved for Documentation," RFC 5737, Internet Engineering Task Force, Jan. 2010. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5737/