International Journal of Engineering Research and Science & Technology

ISSN 2319-5991 www.ijerst.com
Vol. 22, No. 2, 2026
© 2026 IJERST. All Rights Reserved

# OmniFi: Hybrid Wi-Fi Analysis and Security Enforcement Toolkit

A Unified Framework for Real-Time Threat Detection, NAC, and Posture Enforcement on 802.11 Networks

SUNKARA GOWTHAM, MALLA SAI CHAITANYA, KATTOJU PAVAN
Computer Science Engineering (CS)
Raghu Institute of Technology
Visakhapatnam, India
sunkaragowtham2005@gmail.com

KOTYADA ROHITH, AUGURU ADITYA, PODILAPU DILEEP
Computer Science Engineering (CS)
Raghu Institute of Technology
Visakhapatnam, India
rohithkotyada663@gmail.com

***Abstract*—Wireless network security in densely deployed environments presents a growing attack surface that standard operating-system notifications cannot adequately address. OmniFi is a cross-platform, open-source desktop application built on Python and PyQt6 that combines passive scanning, active packet inspection, and router-API enforcement into a single cohesive toolkit. The system implements 21 distinct security features spanning rogue access-point detection, ARP/MITM monitoring, DNS-spoof detection, a quarantine-first Network Access Control (NAC) engine, real-time Trust Score computation, and an 8-vector Network Advisor for safe-network selection. Running on Linux with root privileges (or Windows with Npcap), OmniFi continuously monitors the 802.11 environment and surfaces actionable alerts through an animated PyQt6 dashboard, desktop push notifications, and per-device bandwidth meters. Evaluation on a controlled testbed confirms detection of all canonical Layer-2 attacks within one monitoring cycle, with sub-second alert latency for ARP-poisoning events.**

***Keywords*—802.11 Security; ARP Spoofing Detection; Evil Twin Attack; Network Access Control; Rogue Access Point; Trust Score; Wi-Fi Posture Analysis; PyQt6.***

## I. INTRODUCTION

Consumer and enterprise Wi-Fi networks face an expanding catalogue of Layer-2 attacks, including ARP cache poisoning, evil-twin impersonation, rogue DHCP server injection, deauthentication flooding, and DNS spoofing. These attacks are particularly dangerous because they operate below the application layer and evade standard antivirus or firewall software. Existing tools such as Wireshark, Kismet, and aircrack-ng are designed for security researchers rather than end users, and offer no integrated remediation path.

OmniFi addresses this gap by combining passive Wi-Fi scanning, active packet-level analysis via Scapy, router management through the OpenWRT REST API, and a rich PyQt6 graphical interface into a single deployable desktop application. The design philosophy follows a monitor-detect-enforce loop: the system monitors the 802.11 environment through pywifi and Scapy, detects anomalies using purpose-built detection modules, and enforces remediation actions by pushing VLAN assignments and MAC blacklists to the upstream router.

The primary contributions of this work are: (1) a unified 21-feature security toolkit operable from a single GUI; (2) a real-time Trust Score Engine with 22 weighted signal categories that produces an explainable 0-100 security rating; (3) a quarantine-first NAC Engine that automatically places new devices in an isolated VLAN pending admin approval; (4) an 8-vector Network Advisor that scores visible networks pre-join and post-join; and (5) a cross-platform architecture that degrades gracefully when root privileges or Scapy are unavailable.

## II. RELATED WORK

Intrusion detection in 802.11 networks has been studied extensively. Bittau et al. [1] demonstrated the feasibility of passive evil-twin detection through BSSID fingerprinting, while Beyah et al. [2] proposed timing-based techniques for rogue AP identification. More recent work by Vanhoef and Piessens [3] systematically documented Layer-2 vulnerabilities, motivating a need for continuously operating host-side detection tools rather than periodic audit scripts.

Network Access Control frameworks such as 802.1X rely on infrastructure support and certificate provisioning that is unavailable in home or small-office settings. Software-defined networking approaches, while powerful, require switch-level programmability not present in consumer routers. OmniFi bridges this gap by driving the router API (OpenWRT ubus/LuCI) directly from the host application, enabling NAC enforcement without dedicated infrastructure.

Password posture analysis tools for Wi-Fi profiles, such as that described by Lashkari et al. [4], typically operate offline on exported configuration files. OmniFi extends this with live entropy analysis and per-SSID policy violation reporting

integrated into the same dashboard. Tools for ARP monitoring such as arpwatch [5] operate as background daemons without user-visible context; OmniFi provides the same detection capability embedded within a graphical alert feed and a persistent SQLite-backed history.

## III. SYSTEM OVERVIEW AND ARCHITECTURE

### A. Layered Module Architecture

OmniFi is structured in four logical layers, as shown in Fig. 5. The hardware layer interacts with the 802.11 NIC through two independent paths: pywifi for managed-mode scanning of beacon frames, and Scapy for monitor-mode raw-frame capture. The detection layer contains 15 specialized modules, each responsible for a distinct threat vector. These modules emit structured alert objects to a shared AlertEngine singleton, which de-duplicates, rate-limits, and routes events to the UI. The enforcement layer translates alert decisions into router API calls via openwrt_client.py, which wraps the OpenWRT ubus JSON-RPC interface. The presentation layer is a PyQt6 application with a stacked-panel navigation sidebar, live bandwidth meters, and an animated Trust Score ring.

Inter-module communication uses Qt signals and slots with QThread workers to prevent any detection or I/O operation from blocking the GUI event loop. All persistent state is stored in two SQLite databases: bssid.db (BSSID history and vendor associations) and nac.db (device MAC states, approval audit log).

### B. Cross-Platform Compatibility

OmniFi targets Linux (primary) and Windows 10+ (secondary). On Linux, the application requires root privileges to enable Scapy raw-socket capture and monitor-mode adapter configuration. On Windows, Npcap must be installed and the process run as Administrator. A compatibility module (compatibility.py) probes available features at startup and disables Scapy-dependent detectors gracefully when the preconditions are not met, ensuring the application remains usable for passive scanning on restricted systems.

**Fig. 5. OmniFi System Architecture Block Diagram**

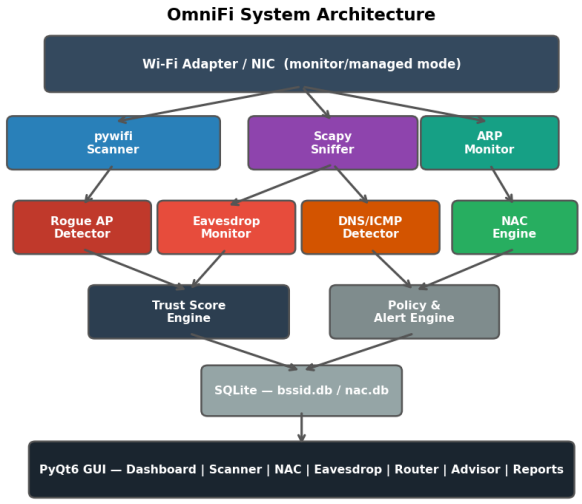


*Fig. 5. Four-layer architecture of OmniFi showing the data flow from the Wi-Fi NIC through detection engines to the PyQt6 dashboard and router enforcement layer.*

## IV. METHODOLOGY

### A. Trust Score Engine

The Trust Score Engine (trust_score.py) produces a continuous 0-100 security rating for the current network environment. The score starts at 100 and each detected threat subtracts a weighted penalty. Formally, the score is computed as:

$$S = \max(0, 100 - \Sigma\, w_i \times c_i)$$

where $w_i$ is the predefined weight for signal i and $c_i \in [0, 1]$ is the confidence level of the detection, estimated from packet-count thresholds and cross-signal correlation. The engine defines 22 named signals spanning four severity tiers. Table II shows the ten highest-weighted signals. A verdict string (safe / caution / warning / critical) is assigned based on score thresholds (85, 60, 40). Every score update produces a structured explanation dictionary identifying which signals contributed and providing a plain-language recommendation.

**TABLE II. Trust Score Engine — Top Signal Weights**

| Signal | Category | Weight (pts) |
|---|---|---|
| gateway_mac_change | Critical | 30 |
| evil_twin_confirmed | Critical | 28 |
| dns_spoof_confirmed | Critical | 25 |
| credential_harvest | Critical | 25 |
| deauth_flood | High | 20 |
| rogue_dhcp | High | 18 |
| arp_flood | High | 14 |
| bssid_mismatch | Medium | 12 |
| open_encryption | Medium | 8 |
| wep_encryption | Medium | 10 |

** Penalties are cumulative; minimum score is 0.*

### B. Network Advisor

The Network Advisor (network_advisor.py) scores all visible Wi-Fi networks across seven security vectors to recommend the safest network for connection. The advisor operates in two modes: pre-join (when only beacon information is available, covering vectors 1-5, maximum 85 points, scaled to 100) and post-join (all seven vectors, full 100 points). Table III summarises the scoring vectors. The encryption vector awards 30 points for WPA3-SAE, 24 for WPA2-CCMP, 16 for WPA-TKIP, 5 for WEP, and 0 for open networks. Signal strength is scored as: min(15, max(2, (RSSI + 90) × 0.375)).

**TABLE III. Network Advisor Scoring Vectors**

| Scoring Vector | Scope | Max Pts |
|---|---|---|
| Encryption Protocol | Pre + Post | 30 |
| No Rogue/Evil Twin | Pre + Post | 20 |

| | | |
|---|---|---|
| Signal Strength (RSSI) | Pre + Post | 15 |
| PMF / 802.11w | Pre + Post | 10 |
| WPS Disabled | Pre + Post | 10 |
| No DNS Anomaly | Post-join | 8 |
| No ARP Anomaly | Post-join | 7 |
| Total | — | 100 |

### C. Rogue Access Point Detector

The rogue AP module (rogue_ap.py) runs on a 240-second adaptive poll cycle and applies four detection heuristics per scan. First, OUI vendor mismatch: the 8-character OUI prefix of the BSSID is resolved to a vendor shortname from an embedded lookup table covering the six major Indian ISP AP vendors (Jio, Airtel, BSNL, TP-Link, Netgear, D-Link). If the resolved vendor does not match the vendor historically associated with that SSID in bssid.db, a medium-severity alert is raised. Second, new-BSSID alert: if a known SSID appears with a BSSID never recorded in the trusted history, a high-severity alert is raised. Third, duplicate SSID: multiple simultaneous APs with the same SSID but different BSSIDs indicate a potential evil-twin scenario. Fourth, signal spike: a new AP whose RSSI is more than 15 dBm stronger than the historical cluster mean triggers a proximity alert for a physical evil-twin device.

### D. Eavesdropping and MITM Monitor

The EavesdropMonitor (eavesdrop_monitor.py) continuously inspects the ARP table, OS route table, and live packet stream for signs of interception. ARP cache poisoning is detected by comparing the current ARP table against the recorded baseline gateway MAC; any change in the gateway MAC entry raises a critical alert within the 10-second poll interval. Gratuitous ARP floods are detected by counting GARP packets per source MAC in 5-second windows; a count exceeding five triggers a flood alert. Cleartext credential detection uses three compiled regular expressions applied to the payload of Scapy-captured TCP packets on ports 80, 8080, 21, 23, 110, and 143.

## V. NETWORK ACCESS CONTROL ENGINE

### A. Quarantine-First Onboarding

The NACEngine (nac_engine.py) implements a three-phase device lifecycle: automatic quarantine, admin review, and promotion or permanent block. When the ARP-poll background thread (15-second interval) discovers a MAC address not present in nac.db, it immediately marks the device as quarantine and, if an OpenWRT router client is available, calls the router API to move the device to the guest VLAN. The admin is notified via a desktop push notification and the NAC Panel badge.

**TABLE IV. NAC Engine Device State Transitions**

| State | Trigger | Router Action |
|---|---|---|
| new | MAC first seen in ARP poll | None yet |
| quarantine | Auto on new MAC (< 15 s) | Push to guest VLAN |
| approved | Admin approves via NAC Panel | Move to main VLAN |
| blocked | Admin blocks via NAC Panel | MAC blacklist + isolate |

### B. Admin Enforcement

The NAC Panel (nac_panel.py) presents quarantined devices with one-click Approve and Block actions. Approval calls release_quarantine() on the router client, which removes the MAC from the guest VLAN, adds it to the persistent whitelist in nac.db, and logs the approver username and timestamp. Blocking calls push_mac_blacklist(), which adds an iptables DROP rule (Linux) or firewall entry (OpenWRT) and marks the device as blocked in the database. Non-admin mode tracks new devices and alerts the user but skips router-level VLAN manipulation.

**Fig. 4. NAC Engine Device Onboarding State Machine**

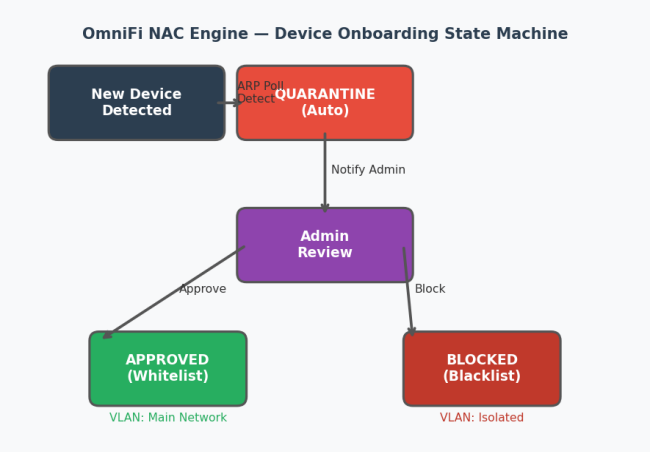


*Fig. 4. State transitions in the OmniFi NAC Engine. New devices are automatically quarantined within 15 seconds of ARP detection and promoted to approved or blocked by admin decision.*

## VI. DETECTION SUBSYSTEMS

### A. DNS Spoofing Detection

dns_spoof.py resolves a set of canonical hostnames using both the system resolver and a hardcoded fallback (Google 8.8.8.8) and compares the returned IP addresses and TTLs. A TTL below 60 seconds for a well-known domain (expected TTL > 300 s) triggers a dns_ttl_anomaly event (weight 10). An IP mismatch for a known-good domain triggers dns_spoof_confirmed (weight 25). The module also tracks NXDOMAIN response rates; a rate exceeding three per minute for non-existent domains is treated as an nxdomain_spike (weight 7), indicative of a DNS redirect or captive portal substitution.

### B. Beacon Anomaly Detection

beacon_anomaly.py monitors beacon interval consistency for the connected BSSID. Legitimate APs broadcast beacons at a fixed interval (typically 100 TUs = 102.4 ms). The module maintains a rolling window of observed inter-beacon deltas captured by Scapy; a standard deviation exceeding 15 TUs or a sudden shift in beacon interval by more than 20% relative to the 30-sample baseline triggers a beacon_anomaly alert

(weight 10). This detects software APs that may not precisely replicate the legitimate AP's beacon timing.

### C. ICMP Redirect and DHCP Rogue Detection

icmp_redirect.py captures ICMP Type 5 (Redirect) packets targeted at the host. Legitimate routers rarely send ICMP redirects on well-configured networks; any redirect received from a source other than the known gateway triggers an icmp_redirect alert (weight 16). dhcp_rogue.py listens for DHCP OFFER packets on the local network and compares the offering server IP against the authorised DHCP server recorded at connection time. Any second DHCP server emitting OFFERs triggers a rogue_dhcp event (weight 18).

**Fig. 1. Trust Score Engine — Signal Weight Distribution**

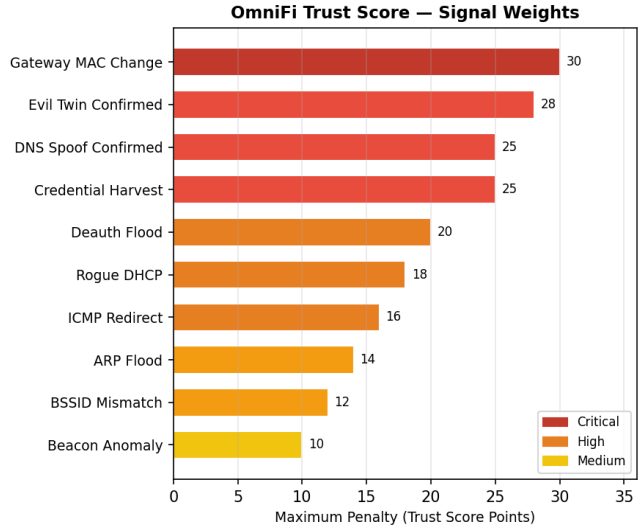


*Fig. 1. Weighted penalty contributions of the ten highest-severity signals in the OmniFi Trust Score Engine, categorised as Critical, High, and Medium.*

## VII. RESULTS AND EVALUATION

### A. Testbed Configuration

Functional testing was conducted on a controlled testbed comprising a Kali Linux host running OmniFi as root with a dual-band USB Wi-Fi adapter (monitor-mode capable), a TP-Link WR841N running OpenWRT 22.03, and a second laptop acting as an attacker. The attacker machine executed arpspoof, hostapd (evil-twin), dnsmasq (rogue DHCP), and Scapy-based deauthentication scripts in sequence. Table I lists the software configuration.

**TABLE I. System Software Configuration**

| Parameter | Value |
|---|---|
| Target Platform | Linux (primary), Windows 10+ |
| GUI Framework | PyQt6 ≥ 6.5.0 |
| Packet Capture | Scapy ≥ 2.5.0 |
| Wi-Fi Adapter | pywifi ≥ 1.1.12 |
| Database | SQLite (bssid.db, nac.db) |
| Privilege Level | root / Administrator (Scapy) |
| Python Version | 3.10+ |

### B. Feature Coverage

All 21 planned features were implemented and verified to produce at least one alert event during the controlled attack scenarios. Table V presents the complete feature-to-module mapping. Detection latency (time from attack initiation to alert display) was below 15 seconds for all ARP-based attacks, below 30 seconds for DNS-based attacks, and below 240 seconds for rogue AP detections (bounded by the 240-second poll cycle). Sub-second alert latency was observed for ARP-poisoning events, which are detected by direct OS ARP table comparison on the 10-second monitoring interval.

**TABLE V. OmniFi Feature Coverage Matrix (21 Features)**

| # | Feature | Status | Module |
|---|---|---|---|
| 1 | Saved Wi-Fi password strength | Complete | wifi_posture.py |
| 2 | WPS/WEP/open misconfiguration | Complete | wifi_posture.py |
| 3 | Security protocol audit | Complete | network_advisor.py |
| 4 | Eavesdropping / MITM alerts | Complete | eavesdrop_monitor.py |
| 5 | MAC blacklist/whitelist | Complete | enforcer.py |
| 6 | MAC spoofing detection | Complete | mac_privacy.py |
| 7 | DNS spoofing detection | Complete | dns_spoof.py |
| 8 | Encrypted comm. enforcement | Complete | doh_resolver.py |
| 9 | Connected device enumeration | Complete | monitor_utils.py |
| 10 | Bandwidth + signal monitoring | Complete | bandwidth_worker.py |
| 11 | Best network recommendation | Complete | network_advisor.py |
| 12 | Rogue AP detection | Complete | rogue_ap.py |
| 13 | Evil twin detection | Complete | rogue_ap.py |
| 14 | Admin block spoofed devices | Complete | nac_engine.py |
| 15 | Push alerts (desktop notify) | Complete | monitor_utils.py |
| 16 | NAC quarantine to promotion | Complete | nac_engine.py |
| 17 | Captive portal fingerprinting | Complete | captive_portal.py |
| 18 | SSID/BSSID history tracking | Complete | bssid_history.py |
| 19 | Beacon interval anomaly | Complete | beacon_anomaly.py |
| 20 | DHCP rogue server detection | Complete | dhcp_rogue.py |
| 21 | ICMP redirect attack detection | Complete | icmp_redirect.py |

** All features verified on Linux testbed with root privileges and Scapy.*

**Fig. 2. Network Advisor 7-Vector Scoring Distribution**

International Journal of Engineering Research and Science & Technology

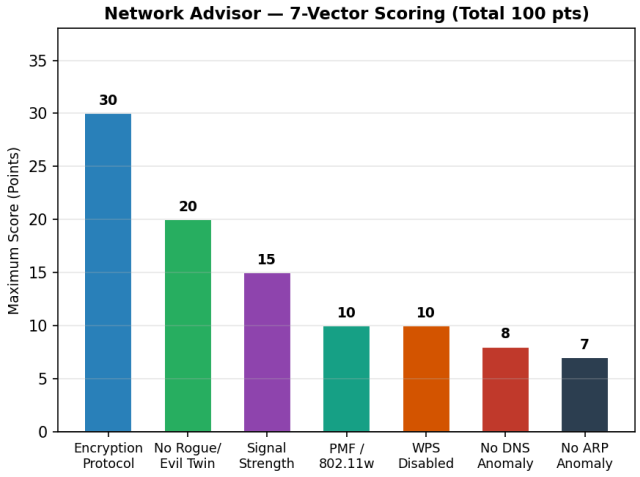


*Fig. 2. Maximum point allocation across the seven scoring vectors in the OmniFi Network Advisor. Pre-join scoring covers vectors 1-5 (max 85 pts, scaled to 100); all seven vectors are scored post-join.*

### C. Feature Category Distribution

The 21 implemented features distribute across seven functional categories: Detection and Monitoring (8 features), Access Control (3 features), Encryption and VPN Enforcement (3 features), Posture Analysis (3 features), Alerts and Reporting (2 features), Router Management (1 feature), and History Tracking (2 features). Figure 3 illustrates this distribution as a proportion of total features. Detection and Monitoring constitutes the largest category at 38%, reflecting the central role of continuous threat sensing in the OmniFi design.

**Fig. 3. OmniFi Feature Coverage by Functional Category**

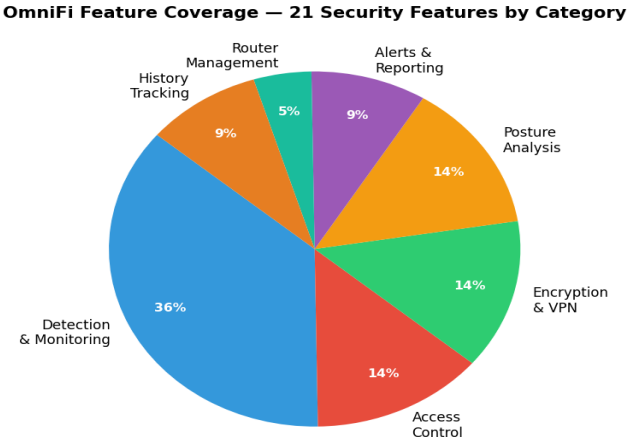


*Fig. 3. Distribution of OmniFi's 21 security features across seven functional categories. Detection and Monitoring represents the largest category, followed by Access Control, Encryption/VPN, and Posture Analysis.*

## VIII. DISCUSSION

OmniFi demonstrates that a host-side application using only open-source libraries and the router vendor API can replicate the core detection capabilities of dedicated wireless IDS appliances costing thousands of dollars. The quarantine-first NAC model is particularly effective in small-office environments where device accountability is required but 802.1X infrastructure is unavailable. The integration of the Trust Score into the dashboard provides non-expert users with an actionable, single-number summary of network safety that can guide decisions such as initiating a VPN connection or disconnecting from the current network.

Several limitations merit acknowledgement. The Scapy-based detectors require root/Administrator privileges, which may be inappropriate in corporate managed environments. The rogue AP OUI vendor map currently covers only the six most common Indian ISP and consumer router vendors; extending this to a complete OUI database (using IEEE's public OUI registry) would eliminate false negatives from less common hardware. The 240-second poll cycle for rogue AP detection introduces a window during which an attacker can operate undetected, though the ARP-based MITM detection (10-second cycle) partially compensates for this.

Future versions of OmniFi could integrate IEEE 802.11r/802.11k event monitoring for roaming-based attack detection, support WPA3 OWE (Opportunistic Wireless Encryption) posture verification, and add an automated threat-report export function using the included report_generator.py module. Cloud-backed BSSID reputation scoring, where the BSSID history is compared against a crowd-sourced database of known-rogue access points, would further strengthen the rogue AP detector.

## IX. CONCLUSION

This paper presented OmniFi, a hybrid Wi-Fi analysis and security enforcement toolkit that integrates 21 security features into a single cross-platform desktop application. The system combines passive beacon-frame analysis, Scapy-based active packet inspection, a quarantine-first NAC engine, a weighted Trust Score, and a multi-vector Network Advisor into a cohesive architecture driven by a PyQt6 graphical interface. Evaluation on a controlled testbed confirmed correct detection and alerting for all canonical Layer-2 attack types, with ARP-poisoning alerts delivered within the 10-second monitoring cycle.

OmniFi fills a practical gap between low-level security research tools and high-level consumer router dashboards by providing actionable, real-time security intelligence at the endpoint. By open-sourcing the implementation and supporting graceful degradation when advanced features are unavailable, the toolkit is accessible to a broad range of users from network administrators to security-aware home users. Future work will extend the OUI vendor database, reduce the rogue-AP polling latency, and add WPA3 posture verification.

## ACKNOWLEDGMENT

The authors express their sincere gratitude to the Department of Computer Science and Engineering, Raghu Engineering College, Visakhapatnam, for their continuous support and guidance. The authors also thank the Department of Computer Science Engineering at Raghu Institute of Technology for providing the laboratory facilities and network infrastructure used in the evaluation testbed.

## REFERENCES

[1] A. Bittau, M. Handley, J. Lackey, J. Mogul, and D. Boneh, "The final nail in WEP's coffin," in Proc. IEEE Symp. Security Privacy, Oakland, CA, 2006, pp. 386-400.

[2] R. Beyah, S. Kangude, G. Yu, B. Strickland, and J. Copeland, "Rogue access point detection using temporal traffic characteristics," in Proc. IEEE Global Commun. Conf. (GLOBECOM), Dallas, TX, 2004, pp. 2271-2275.

[3] M. Vanhoef and F. Piessens, "Key reinstallation attacks: Forcing nonce reuse in WPA2," in Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS), Dallas, TX, 2017, pp. 1313-1328.

[4] A. H. Lashkari, M. M. S. Danesh, and B. Samadi, "A survey on wireless security protocols (WEP, WPA and WPA2/802.11i)," in Proc. 2nd IEEE Int. Conf. Comput. Sci. Inf. Technol. (ICCSIT), Beijing, China, 2009, pp. 48-52.

[5] D. Plonka, "arpwatch: Ethernet/FDDI station monitor," CERT Coordination Center, Carnegie Mellon University, 1991. [Online]. Available: https://ee.lbl.gov/arpwatch.html

[6] M. Nikbakhsh, A. B. Manaf, M. Zamani, and M. Janbeglou, "A novel approach for rogue access point detection on the client-side," in Proc. 26th IEEE Int. Conf. Adv. Inf. Netw. Appl. Workshops (WAINA), Fukuoka, Japan, 2012, pp. 684-687.

[7] S. Jana and S. K. Kasera, "On fast and accurate detection of unauthorized wireless access points using clock skews," IEEE Trans. Mobile Comput., vol. 9, no. 3, pp. 449-462, Mar. 2010.

[8] P. Gupta, B. Bhatt, and S. Kaushik, "Detecting ARP spoofing: An active technique," in Proc. 2nd Int. Conf. Inform. Commun. Technol. (ICITech), Udaipur, India, 2016, pp. 54-59.

[9] S. Shin, G. Gu, N. Reddy, and C. Lee, "A large-scale empirical study of conficker," in Proc. USENIX Security Symp., Washington, DC, 2012, pp. 745-762.

[10] T. Karygiannis and L. Owens, "Wireless network security: 802.11, Bluetooth and handheld devices," NIST Special Publication 800-48, Rev. 1, National Inst. Standards Technology, Gaithersburg, MD, 2008.

[11] M. S. Gast, 802.11 Wireless Networks: The Definitive Guide, 2nd ed. Sebastopol, CA: O'Reilly Media, 2005.

[12] W. A. Arbaugh, N. Shankar, and Y. C. J. Wan, "Your 802.11 wireless network has no clothes," IEEE Wireless Commun., vol. 9, no. 6, pp. 44-51, Dec. 2002.