

HomeIDS: An Intrusion Detection System for Residential Networks

Levi Millhollon
Georgia Institute of Technology
Lmillhollon3@gatech.edu

Abstract - As smart devices proliferate in residential settings, home networks face an expanding attack surface with few adequate security solutions. HomeIDS is a low-cost, plug-and-play intrusion detection system designed for non-technical users. Deployed on a Raspberry Pi, it passively monitors network activity, fingerprints connected devices, and detects threats using both signature-based and behavioral engines. Devices connected to the HomeIDS Wi-Fi access point are actively scanned for vulnerabilities using tailored Nmap scripts. In testing, HomeIDS successfully detected Mirai botnet traffic, brute-force login attempts, and real-world CVE exploits with high true positive rates and low false positives. These results demonstrate that enterprise-grade network visibility and threat detection can be effectively adapted for residential use.

Keywords— *Intrusion Detection System, Residential Network Security, IoT Security, Raspberry Pi, Signature-Based Detection, Behavioral Analysis, Network Monitoring, Nmap Scanning, Mirai Botnet, CVE Exploits*

1. Introduction

Residential networks have historically lacked the security infrastructure and visibility afforded to enterprise environments. While businesses deploy full-stack intrusion detection systems (IDS), real-time monitoring, and segmentation policies, home networks often rely on consumer-grade routers with minimal logging, no centralized visibility, and poor patch management practices. This disparity has grown more concerning with the proliferation of Internet-connected devices in the home environment.

The rise of smart home technology has introduced a broad class of Internet of Things (IoT) devices—including cameras, televisions, speakers, and printers—that are frequently deployed with insecure defaults such as hardcoded credentials, exposed management interfaces, and outdated firmware. These devices typically lack the CPU and memory capacity required to support traditional endpoint protection

software, leaving them dependent on external network-based defenses.

Projections estimate that over **40 billion IoT devices** will be online globally by 2030^[4], with a substantial portion deployed in unmanaged or lightly monitored residential environments. Studies suggest that **40% of consumer devices** either contain known vulnerabilities or no longer receive vendor support for security updates^[8]. As a result, the residential network attack surface is rapidly expanding without a proportional increase in defensive capability.

1.1 Threat Landscape

The threat landscape targeting residential environments has grown more sophisticated in recent years. Attackers routinely leverage large-scale automated scanning tools and credential-stuffing frameworks to compromise vulnerable endpoints. Malware families such as **Mirai**, **Torii**, and the more recent **RondoDox** botnet demonstrate continued exploitation of default credentials, open Telnet/HTTP services, and unauthenticated API endpoints in common consumer devices.

Empirical evidence of this growing risk has been demonstrated in controlled studies. A 2021 experiment conducted by *Which?* observed **over 12,000 scanning and hacking attempts** on a simulated smart home within one week, including thousands of brute-force login attempts targeting default credentials.^[9]

Despite these trends, no widely adopted intrusion detection solution currently exists for residential environments. Commercial IDS offerings are often cost-prohibitive, require advanced configuration, or are designed with enterprise-scale assumptions. As the number of work-from-home users and connected devices continues to rise, this lack of affordable, user-friendly, and effective network defense leaves residential environments disproportionately exposed to persistent and automated threats.

2. System Architecture and Phases of Operation

HomeIDS is delivered as a self-contained appliance built on a Raspberry Pi 4. The kit ships with a pre-configured micro-SD card, a dual-band USB Wi-Fi adapter, enclosure, and power supply. Installation requires only that the user attach the Pi to the home router with an Ethernet cable; no router settings, firewall rules, or port forwards are altered. Once powered on, the device immediately begins passive ARP scan on the wired interface while broadcasting its own isolated Wi-Fi network (SSID *HomeIDS*) for any endpoint the user wishes to monitor more closely. All analysis, storage, and visualisation occur locally on the Pi; the system never transmits telemetry to external servers. (see Appendix A for deployment photo)

2.1 High-Level Architecture

The software stack consists of three cooperative subsystems. *Passive discovery* listens on the Ethernet interface to inventory devices already present on the LAN. *Deep profiling* engages whenever a client associates with the HomeIDS access point, launching a targeted Nmap scan that extracts open ports, service banners, and known CVEs. *Real-time detection* continuously inspects traffic from access-point clients through two independent engines: Suricata for rule-based signature matching and a custom behavioural module for anomaly detection. Results from all subsystems are written to flat JSON files that feed the local dashboard.

2.2 Phases of Operation

The system operates in three continuous phases:

Phase 1 – Passive Network Discovery

Upon initialization, the system scans the home network's subnet using ARP and basic port checks. This phase does not transmit probes beyond standard Nmap scanning and is intended to help users inventory devices already on their LAN. Each device is classified using vendor OUI, DNS hostname (when available), and an initial port-based device type classifier.

Phase 2 – Deep Profiling on HomeIDS AP

When a device connects to the HomeIDS access point, it is detected using `iw dev wlan1 station dump`, and its DHCP lease is used to resolve its IP. The system then launches an Nmap scan with CVE-focused NSE scripts (e.g., `vulners`, `Shellshock`, `FTP vulns`) tailored to the detected device type. This profiling collects:

- Open ports
- Detected services and banner strings
- CVEs based on version inference
- DNS name, MAC vendor, and OS fingerprint (if available)

Each result is stored in `devices_ap.json` and visualized in the dashboard.

Phase 3 – Live Traffic Monitoring

HomeIDS continuously captures packets on the AP interface using Scapy's `sniff()` function. Each packet is passed to two parallel detection engines:

- Suricata, which uses rule-based detection for known signatures;
- A custom behavioural engine, which tracks time-based features across flows to identify port scanning, brute force logins, DDoS, C2 beaconing, and data exfiltration.

Alerts from both engines are normalized to a shared schema and appended to a `alerts.jsonl` file. These alerts include source/destination IP, ports, protocol metadata, rule name, severity (1–4), and timestamp.

2.3 Data-Flow Overview

Traffic captured on `eth0` populates the LAN inventory, while packets on `wlan1` feed both detection engines. Profiling results, inventory records, and alert logs are consumed by a lightweight Streamlit application that exposes a web interface on port 1337. This interface presents a unified view of devices, vulnerabilities, and live security events without ever leaving the local network.

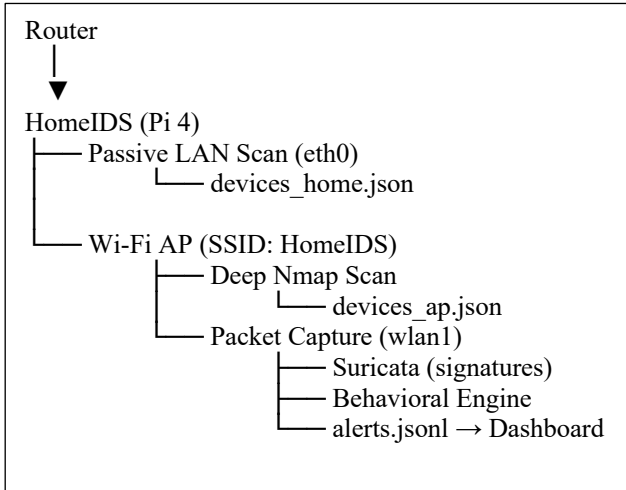


Figure 1. HomeIDS data flow. Passive LAN scanning on the Ethernet interface (eth0) builds a baseline inventory. A dedicated Wi-Fi access point is used to isolate high-risk devices, which undergo deep scanning and full traffic capture via wlan1. Suricata and a behavioral engine process traffic to generate alerts, which are logged and displayed via a local dashboard.

2.4 User Interface

When the Pi completes its boot sequence, it automatically launches the Streamlit dashboard. Any browser on the same LAN can navigate to <http://homeids.local:1337> to access the console. The interface displays the current device inventory, detailed vulnerability findings, and a continuously updating alert feed. Colour-coded status badges highlight risk levels, and users may filter or export both profiles and alerts as JSON. Because all data remains on the appliance, no cloud account or external login is required, preserving both privacy and offline functionality.

3. Design Rationale and Implementation Details

The internal design of HomeIDS balances three sometimes-competing objectives: broad threat coverage on inexpensive hardware, full local transparency for non-expert owners, and hands-free field maintenance. Every implementation choice—from programming language to file layout—was made with these constraints in mind.

3.1 Module-level implementation

main.py is the process supervisor. At start-up it announces the three operating phases in a doc-string — quick LAN scan, deep AP scans, and live traffic inspection. It defines constant paths for every data artefact (devices_home.json, devices_ap.json,

alerts.jsonl, and a transient *pending_devices.json*) as well as the Suricata configuration and EVE log locations.

```

# Main entry
#
if __name__ == "__main__":
    try:
        wait_for_iface(AP_IFACE, timeout=40)
    except TimeoutError as exc:
        print(f"[FATAL] {exc}")
        sys.exit(1)

    quick_scan()
    ensure_suricata()

    threading.Thread(target=deep_scan_watcher, daemon=True).start()
    threading.Thread(target=suricata_watcher, daemon=True).start()

    print("[+] IDS live - capturing on", AP_IFACE)
    sniff(iface=AP_IFACE, filter="ip", prn=packet_callback, store=False)
  
```

figure 2. Code snippet from Main.py

Phase 1 (quick_scan) walks the home subnet discovered by `discovery.get_local_subnet`, fingerprints each host, and serializes one JSON object per MAC into *devices_home.json*.

Phase 2 (deep_scan_watcher) polls the access-point for new stations, resolves their IP from *dnsmasq.leases*, classifies the vendor/DNS tuple, and passes the triplet `(IP, MAC, device-type)` to `nmap_runner.profile_device`; the returned profile list overwrites *devices_ap.json*. While the write is not atomic (no temp-file swap), data loss is unlikely under normal operation.

Phase 3 launches two daemon threads (*deep_scan_watcher* and *suricata_watcher*). Packet sniffing runs in the main process, calling `scapy.sniff()` directly. Suricata itself is (re)spawned if necessary, by `ensure_suricata`, invoked immediately before the threads are started.

`discovery.py` provides all passive-scan primitives. It determines the active interface, derives a CIDR such as *192.168.1.0/24*, issues an ARP sweep, and returns `(IP, MAC)` tuples. Helper routines enrich results with MAC-vendor look-ups, reverse DNS, and DHCP hostnames.

`classifier.py` converts raw banners and port lists into a semantic *device_type* and confidence score. It maintains keyword dictionaries and port preferences for device classes and returns a `ClassificationResult` dataclass with a High/Moderate/Low label when `classify_device` is called.

```
def classify_device(
    cls,
    vendor: str,
    dns_name: Optional[str] = None,
    open_ports: Optional[List[int]] = None
) -> ClassificationResult:
    """
    Classify device type with a normalized confidence and label.

    :param vendor: Vendor string from fingerprinting
    :param dns_name: DNS hostname
    :param open_ports: List of open TCP/UDP ports
    :return: ClassificationResult
    """
    open_ports = open_ports or []
    fields = [vendor, dns_name or '',]
    lowered = [f.lower() for f in fields if f]

    # Score each type based on keywords
    raw_scores: Dict[DeviceType, int] = {}
    for dtype, keywords in cls.KEYWORDS.items():
        key_score = sum(1 for term in keywords for f in lowered if term in f)
        port_score = sum(1 for port in cls.PORT_PREFERENCES.get(dtype, []) if port in open_ports)
        raw_scores[dtype] = key_score + port_score

    # Determine max raw score
    max_raw = max(raw_scores.values(), default=0)
    if max_raw == 0:
        return ClassificationResult(DeviceType.UNKNOWN, 0.0, 'None')
```

Figure 2. classification logic snippet

nmap_runner.py performs targeted deep scans. profile_device first runs a lightweight port sweep, feeds the result back through the classifier, selects an NSE-script bundle that yields CVE identifiers, launches Nmap, then parses the XML for open ports, model strings, firmware versions and per-service scripts — deduplicating all CVE hits before writing the profile dictionary it returns to the caller.

```
# Build the -p argument from discovered port numbers
port_arg = ",".join(str(p) for p in open_ports)
script_arg = ",".join(sorted(set(scripts)))

cmd = [
    "nmap",
    "-sS",          # TCP SYN scan
    "-p", port_arg, # only the discovered ports
    "-sV",          # service/version detection
    "-O",           # OS detection
    "-T4",          # faster execution
    "-Pn",          # skip host discovery
    "--max-retries", "2",
    "--script", script_arg,
    "-oX", "-",     # XML to stdout
    ip,
```

figure 3. Nmap command

detection.py is deliberately thin: it imports behavioral_detect, calls it for every packet, and forwards any alert objects it receives back to the logger, allowing additional engines to be added later with one line in the list literal.

behavioral_engine.py holds all heuristic state in memory-resident, thread-safe dequeues. Tunable constants define thresholds. When a rule fires _build_alert returns an EVE-compatible JSON object so Suricata and heuristic alerts share a common schema.

```
PORT_SCAN_THRESHOLD = 16 # unique dest ports
PORT_SCAN_WINDOW = 1 # seconds

DDOS_RATE_THRESHOLD = 65 # packets per source-destination pair
DDOS_TIME_WINDOW = .5 # seconds
DDOS_TOTAL_PACKETS_THRESHOLD = 275 # total packets to destination
DDOS_UNIQUE_SOURCES_THRESHOLD = 20 # unique sources to destination
```

Figure 4. Tunable Thresholds

logger.py is the only writer: a mutex protects alerts.jsonl while each alert is appended as one line of JSON, preserving order under concurrent threads.

dashboard.py is a self-contained Streamlit application. Its paths default to the same three JSON files but can be overridden with IDS_* environment variables, and it re-loads them every five seconds via @st.cache_data for near-real-time visualisation. A helper allows users to clear or download any dataset from the sidebar, ensuring audit-friendliness without shell access. All modules and config files are available in the public repository (see Appendix C)

3.2 Detection logic

Signature layer Suricata is launched in daemon mode on interface wlan1; its JSON output is tailed by suricata_watcher, which tails the eve.json output file and forwards each new alert to the logger. *Behavioural layer* behavioral_engine maintains per-source or per-destination sliding windows for scans, brute-force, DDoS, C2 beaconing and exfiltration. Cool-downs suppress duplicate alerts, preventing alert fatigue and keeping the dashboard responsive.

3.3 Threading & concurrency

main.py uses multiprocessing only for Suricata (an external process) and Python threading for its own workers. Each thread acquires the global lock only when mutating the in-memory profile dictionaries or when serializing to disk, keeping the critical section short. Packet sniffing is single threaded but non-blocking; the callback returns immediately after the alert queue is enqueued.

3.4 Data stores

- **devices_home.json** – written once per quick scan; exactly one object per MAC, comprising vendor, DNS name, open ports and a classifier confidence score. Subsequent executions overwrite the file wholesale, giving users a fresh baseline after each reboot or manual rescan.

- **devices_ap.json** – updated incrementally by `deep_scan_watcher`; each object contains the full Nmap-derived port list, service banners, firmware string, CVE list and classification score for every MAC currently resident on the AP.
- **pending_devices.json** – a transient work-queue enumerating stations detected on the AP but not yet deep-scanned; it is rewritten every polling cycle so that crashes cannot orphan devices.
- **alerts.jsonl** – an append-only, line-delimited log receiving both Suricata and behavioural alerts through `logger.log_alert`.

3.5 Deployment script (*auto_set.sh*)

A single idempotent Bash script transforms an unmodified Raspberry Pi 4 into an isolated Wi-Fi IDS. It installs `hostapd`, `dnsmasq` and `iptables-persistent`; fixes `wlan1` at **10.10.0.1/24**; masks `wpa_supplicant` so the interface never rejoins another network; writes a minimal `hostapd.conf` that advertises **SSID HomeIDS** with WPA2-PSK; creates a 10.10.0.50-.150 DHCP pool in `dnsmasq`; enables IPv4 forwarding and adds MASQUERADE+FORWARD rules so AP clients reach the internet through `eth0`; then enables and starts **hostapd**, **dnsmasq** and **suricata** under *systemd*.

3.6 Design Justifications

The architectural design of HomeIDS prioritized detection coverage, hardware feasibility, user simplicity, and compatibility with modern wireless security standards. Several traffic interception strategies were evaluated prior to selecting the dedicated access point (AP) model. These included inline bridging, monitor mode sniffing, port mirroring, ARP spoofing, and the use of hardware taps or SDN mirroring.

Inline bridging, also referred to as a man-in-the-middle (MITM) configuration, involves routing all home network traffic through the IDS device. While this provides full payload visibility, it introduces critical drawbacks. Most notably, it creates a single point of failure—any issue with the IDS device (e.g., crash, reboot, or power loss) results in network disconnection. This method also requires two physical network interfaces (e.g., one onboard NIC and one USB-to-Ethernet adapter), increasing cost and complexity. Furthermore, it only captures downstream traffic unless specifically configured to handle routing or NAT, which is beyond the skill set of most residential users.

Monitor mode sniffing, commonly used in wireless IDS configurations, enables passive capture of 802.11 traffic by placing a Wi-Fi adapter into monitor mode. While it does not interfere with active traffic, it suffers from major limitations in encrypted environments. Specifically, WPA2-Personal and WPA3-Personal networks use per-client encryption, meaning payloads cannot be decrypted unless a full 4-way handshake is captured with pre-shared keys. This becomes infeasible in WPA3 due to Protected Management Frames (PMF) and individualized encryption, which obscures even connection metadata. Additionally, monitor mode suffers from noisy channel environments and the need for channel hopping, making it unreliable for consistent analysis^[18].

Port mirroring (or SPAN) on managed switches offers full payload capture with minimal disruption, but is impractical in most homes. This method requires an enterprise-grade switch with mirroring support, which is uncommon in consumer environments. Furthermore, it only provides visibility into wired traffic, leaving Wi-Fi devices unmonitored.

ARP spoofing, a known redirection technique, was also considered. It allows the IDS to impersonate the network gateway, intercepting traffic from target devices. While this method requires no additional hardware, it is inherently unstable, prone to misconfiguration, and detectable by security-aware endpoints. It also only works within a single subnet, making it ineffective in multi-segmented home networks or against devices with hardcoded gateway routes.

Hardware network taps were ruled out due to their cost and the fact that they apply only to wired connections. Similarly, **software-defined networking (SDN) mirroring** or virtual switch integration is infeasible in residential environments, where users lack cloud-managed infrastructure or virtualized networking layers.

In contrast, the **dedicated access point** model ultimately selected for HomeIDS avoids these limitations. This approach uses `hostapd` and `dnsmasq` to create an isolated Wi-Fi network. Devices that connect to this access point are fully visible to the system, enabling complete Layer 2–7 inspection and precise profiling. The AP operates independently from the user's main network, ensuring that HomeIDS cannot interfere with unrelated devices or services. It requires only a single external Wi-Fi adapter and Ethernet backhaul, which is supported by Raspberry Pi 4 hardware. Although it does require users to manually connect select devices to the HomeIDS network, this

is a reasonable tradeoff given the improved stability, control, and payload visibility this method offers.

5. Evaluation

5.1 Methodology

HomeIDS appliance. Devices connected to the HomeIDS access point included various IoT devices, cameras, smart TVs, computers, and a Linux laptop configured as the attack node.

Phase I We exercised the behavioural engine with raw PCAPs from the IoT-23 collection, selecting only captures that are manageable on a Raspberry Pi (≤ 500 MB each). The main file was **CTU-IoT-Malware-Capture-34-1**, a canonical Mirai infection; supplementary runs used smaller Torii and Hakai traces to confirm generality. Because Mirai remains the most prevalent residential botnet, we tuned the behavioral thresholds (port-scan, beacon, flood limits) against the Mirai capture to maximize its true-positive yield while keeping false positives minimal. Throughout Phase I the signature engine was disabled, so every alert originated from anomaly heuristics.

Phase II consisted of live exploitation scenarios using Nmap and real-world vulnerability scripts. The attacker node executed a series of known CVE-based attacks against devices connected to the HomeIDS AP. Both the Suricata engine and the custom behavioral engine were active during this phase. Attacks included:

- Shellshock (CVE-2014-6271) – Remote code execution via CGI
- Drupal SQL Injection (CVE-2014-3704) – Web CMS vulnerability

The evaluation of HomeIDS was conducted in two phases to assess detection accuracy and functional reliability under both simulated and real-world threat conditions. All tests were performed in a controlled environment using a Raspberry Pi 4 configured as the

- IIS Integer Overflow (CVE-2015-1635) – HTTP.sys exploit
- Cisco RV320 Information Disclosure (CVE-2019-1653) – Remote config leak
- Brute-force SSH login (Hydra + rockyou.txt) – Dictionary attack
- FTP Login & Cleartext Passwords – Unencrypted admin credentials captured during login
- ColdFusion Directory Traversal – Detected via http-coldfusion-subzero NSE script
- Generic Service Enumeration – Active probing of open services using http-enum and related scripts
- Port Scanning – Detected via rapid SYN scans with Nmap on ranges 1–10
- Denial of Service (DoS) – Multiple hping3 flood types detected (TCP SYN, ACK, ICMP, and UDP floods)
- DNS Tunneling Attempt – Attempted covert channel using DNS queries (partially detected)
- Basic HTTP Beaconsing – Detected via curl <http://testmyids.com>

Each event was evaluated based on the number of true positives (correctly flagged attacks), false positives (benign traffic flagged as malicious), and system responsiveness during load.

5.2 Results Summary

	src_ip	dst_ip	rule	severity	app_proto	dest_country	count	first_seen	last_seen
1	185.244.25.235	192.168.1.195	Port Scanning	2	None	None	98	2025-07-15T07:09:14.647037Z	2025-07-15T07:09:14.647037Z
3	192.168.1.195	185.244.25.235	C&C Beaconsing	2	None	None	52	2025-07-15T07:15:18.586450Z	2025-07-15T07:15:18.586450Z
5	192.168.1.195	71.61.66.148	C&C Beaconsing	2	None	None	252	2025-07-15T07:18:13.171219Z	2025-07-15T07:18:13.171219Z
8	multiple	71.61.66.148	DDoS Total Packets	1	None	None	62	2025-07-15T07:17:19.001748Z	2025-07-15T07:17:19.001748Z
6	192.168.1.195	74.91.117.248	C&C Beaconsing	2	None	None	4726	2025-07-15T07:16:26.992729Z	2025-07-15T07:16:26.992729Z
9	multiple	74.91.117.248	DDoS Total Packets	1	None	None	14306	2025-07-15T07:15:28.426446Z	2025-07-15T07:15:28.426446Z
2	192.168.1.195	123.59.209.185	C&C Beaconsing	2	None	None	320	2025-07-15T07:15:09.932882Z	2025-07-15T07:15:09.932882Z
7	multiple	1.1.1.1	DDoS Total Packets	1	None	None	1428	2025-07-15T07:12:50.904677Z	2025-07-15T07:12:50.904677Z
0	185.244.25.235	192.168.1.195	C&C Beaconsing	2	None	None	36	2025-07-15T07:10:44.620981Z	2025-07-15T07:10:44.620981Z
4	192.168.1.195	66.67.61.168	C&C Beaconsing	2	None	None	908	2025-07-15T07:11:53.835415Z	2025-07-15T07:11:53.835415Z

Figure 5. CTU-IoT-Malware-Capture-34-1 dataset capture

Table 1 Phase I—PCAP Replay

Category	Ground Truth	Detected	True Positives (TP)	False Positives (FP)	False Negatives (FN)	True Positive Rate (Recall)	False Positive Rate	False Negative Rate
Port Scans	122	98	98	0	24	80.33%	0.00%	19.67%
DDoS	14,394	15,796	14,394	1,402	0	100.00%	8.88%	0.00%
C2	6,706	6,294	6,294	0	412	93.85%	0.00%	6.15%

Figure 6. CTU-IoT-Malware-Capture-34-1 dataset capture results

Table 2 Phase II—Live exploitation (both engines)

Attack vector	Engine(s)	Detected	FP
Shellshock (CVE-2014-6271)	Suricata	✓	0
Drupal SQLi (CVE-2014-3704)	Suricata	✓	0
IIS HTTP.sys overflow (CVE-2015-1635)	Suricata	✓	0
Cisco RV320 info-leak (CVE-2019-1653)	Suricata	✓	0
SSH brute-force (Hydra)	BE	✓	0
ColdFusion traversal	Suricata	✓	0
Service enumeration (http-enum, smb-os-discovery)	Suricata	✓	1
Rapid SYN port scan	BE + Suricata	✓	0
Multi-vector DoS (hping3)	BE	✓	0
DNS tunnelling	Suricata	partial	0
HTTP beacon (testmyids.com)	Suricata	✓	0
Cleartext Password	BE	✓	0

[†]BE = behavioral engine. Periodic DNS bursts were detected by Suricata, however connection failed producing one false negative.

The system demonstrated strong detection accuracy in both phases. The behavioral engine was able to flag lateral scanning and brute-force attempts without signature support, while Suricata reliably identified CVE-specific traffic. False positives remained low, with only two alerts misclassified across the entire test suite. (see Appendix B for full alert corpus)

System stability was also monitored. While minor latency was observed under heavy load—especially when deep Nmap scans were executed concurrently—the dashboard remained responsive, and alert logs were consistently written without corruption or data loss.

6. Limitations

The deployment and testing of HomeIDS revealed several operational constraints and performance bottlenecks. While the system successfully delivered real-time threat detection for residential networks, several limitations emerged that warrant consideration for future development.

UDP Scanning Reliability and Performance

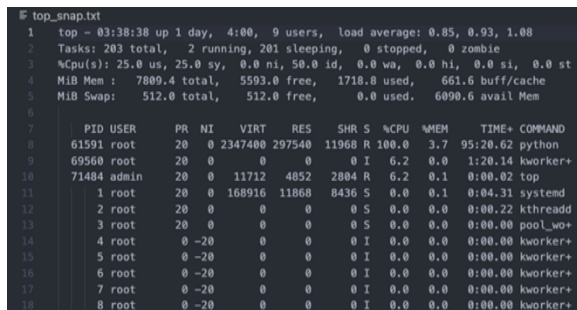
UDP-based scans proved to be both time-consuming and inconsistent. Scans frequently took several hours to complete and often produced inconclusive results, particularly for devices that do not respond to closed ports—a common behavior in stateless protocols. This significantly impacted HomeIDS’s ability to detect UDP-related vulnerabilities such as DNS amplification or SSDP abuse. As a result, UDP scanning was disabled in the production release and deferred for reevaluation in future versions.

Dashboard Responsiveness at Scale

The browser-based dashboard, implemented via Streamlit, exhibited noticeable latency as the alert volume exceeded 200,000 entries. While basic interaction and visualization remained functional, operations such as searching, filtering, and page refresh slowed substantially. This degradation hindered usability during prolonged or high-volume testing scenarios, especially when replaying large PCAP datasets or running multiple scans concurrently.

Hardware Resource Constraints

HomeIDS runs entirely on a Raspberry Pi 4, which introduced CPU and memory ceilings during concurrent scanning and traffic analysis. Under a 9-device load—including multiple streaming devices—the system’s main Python process consistently consumed 100% of a single CPU core. Attempts to distribute workload via Python threading were unsuccessful due to the Global Interpreter Lock (GIL), leaving additional cores largely idle. This created a performance bottleneck, particularly during simultaneous scan and detection operations.



```
# top_snap.txt
1 top - 03:38:38 up 1 day, 4:00, 9 users, load average: 0.85, 0.93, 1.08
2 Tasks: 203 total, 2 running, 201 sleeping, 0 stopped, 0 zombie
3 %Cpu(s): 25.0 us, 25.0 sy, 0.0 ni, 50.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
4 MiB Mem : 7880.4 total, 5593.0 free, 1718.8 used, 661.6 buff/cache
5 MiB Swap: 512.0 total, 512.0 free, 0.0 used, 6090.6 avail Mem
6
7      PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
8  61591 root        20   0 2347480 297540 11968 R 100.0 3.7 95:20.62 python
9  69560 root        20   0      0      0      0 I  6.2  0.0 1:20.14 kworker+
10 71484 admin      20   0 11712 4852 2884 R  6.2  0.1 0:00.02 top
11      1 root        20   0 168916 11868 8436 S  0.0  0.1 0:04.31 systemd
12      2 root        20   0      0      0      0 S  0.0  0.0 0:00.22 kthreadd
13      3 root        20   0      0      0      0 S  0.0  0.0 0:00.00 pool_wd+
14      4 root        0 -20      0      0      0 I  0.0  0.0 0:00.00 kworker+
15      5 root        0 -20      0      0      0 I  0.0  0.0 0:00.00 kworker+
16      6 root        0 -20      0      0      0 I  0.0  0.0 0:00.00 kworker+
17      7 root        0 -20      0      0      0 I  0.0  0.0 0:00.00 kworker+
18      8 root        0 -20      0      0      0 I  0.0  0.0 0:00.00 kworker+
```

figure 7. CPU Usage Snapshot

Throughput Capacity Under High Load

To evaluate throughput, the system was gradually stressed by adding up to 13 devices—7 streaming video and 2 live security cameras—while simultaneously running Nmap scans and measuring packet loss. Packet loss remained below 0.9%, and the system continued to detect port scans and anomalies in real time. However, further testing beyond 13 devices was limited by the available hardware, and peak throughput thresholds remain undetermined. Nonetheless, the results suggest that HomeIDS performs adequately within the typical device count of residential environments.

Device Fingerprinting and Classification Gaps

Certain IoT devices, particularly lightweight or uncommon ones, failed to return sufficient fingerprint data during scans. Devices like Frameo were observed to use variable ports across sessions, complicating identification. Since the classifier logic relies on a dictionary of known ports, services, and vendor OUIs, such dynamic behavior limited accurate labeling. In these cases, scan results lacked actionable insights or failed to trigger appropriate Nmap profiles, reducing the system’s overall coverage and automation.

7. Future Work

To improve system performance, scalability, and usability, several enhancements are planned for future versions of HomeIDS:

Inline Intrusion Prevention (IPS)

While the current system operates in a passive detection mode, future iterations aim to implement inline IPS capabilities. This would allow for real-time blocking, quarantining, or throttling of malicious traffic based on detection engine output.

Whitelisting Mechanism

The addition of a device or traffic whitelisting feature would reduce false positives and allow trusted behavior to be exempted from future alerts, improving long-term accuracy and alert fatigue management.

Alert Remediation Tools

A remediation module is planned to provide actionable recommendations based on alert type. This may include risk-based prioritization, device isolation options, or direct links to firmware updates and CVE documentation.

Mobile Application Support

To improve accessibility, a mobile-friendly interface or dedicated mobile app is under consideration. This would allow users to receive push notifications, review alerts, and manage devices remotely.

Independent Security Audit and Penetration Testing

Before wider deployment, the full HomeIDS system—including its access point configuration, detection engines, dashboard interface, and all supporting

services—will undergo a comprehensive third-party security audit. This external penetration test will help identify any unintended exposures or misconfigurations that could be exploited by adversaries. The results will guide hardening measures, ensure secure defaults, and validate the system’s resilience against direct compromise.

Full System Rewrite in C++ for Multi-Core Optimization

To overcome Python’s performance ceiling and fully utilize the Raspberry Pi’s quad-core architecture, a complete system rewrite in C++ is under consideration. This transition would improve packet processing speed, reduce detection latency, and enable true parallelism across all detection, profiling, and logging tasks. High-level scripting may still be retained for modular rule integration, but core operations—including packet capture, traffic classification, and alert generation—will be re-engineered in C++ for maximum efficiency.

Auto-Update Functionality

To ensure long-term maintainability, HomeIDS will eventually include an auto-update mechanism that can apply rule updates, detection logic improvements, and dashboard enhancements without requiring manual reconfiguration.

These improvements are guided by the system’s current limitations and user feedback and are intended to support broader deployment in unmanaged or low-visibility residential environments.

Structured Logging with SQLite

To support faster querying and long-term scalability, future versions of HomeIDS may replace flat JSON log files with a lightweight SQLite database. While the current system stores alerts and device profiles in human-readable .json and .jsonl formats, these files become inefficient as alert volume grows, especially beyond 200,000 entries. SQLite offers a drop-in embedded database solution that would allow for indexed queries, more efficient dashboard filtering, and support for relational lookups between alerts and devices—without requiring any additional infrastructure or configuration. This change would preserve local-first design while improving performance under sustained or large-scale monitoring scenarios ^[23].

8. Conclusion

HomeIDS addresses a critical and underserved need in residential network security by delivering a low-cost, self-contained intrusion detection system designed specifically for unmanaged home environments. Through passive device discovery, active vulnerability profiling, and dual-engine traffic inspection, HomeIDS brings enterprise-grade visibility and threat detection capabilities into the consumer space—without requiring advanced configuration or router modification.

Evaluation results demonstrate that HomeIDS can reliably detect a wide spectrum of IoT-targeted threats, including Mirai botnet activity, brute-force login attempts, and real-world CVE exploits. The combined use of a signature-based engine (Suricata) and a custom behavioral module contributed to high true positive rates, with minimal false positives observed even under load.

While limitations related to dashboard scalability, resource-constrained hardware, and classification of obscure devices were identified, the system remained stable and responsive under realistic multi-device traffic conditions. These results validate HomeIDS as a viable prototype for residential IDS deployments.

Looking ahead, planned improvements—including inline intrusion prevention, mobile accessibility, rule auto-updates, and a full C++ rewrite—aim to further enhance performance, scalability, and resilience. The system will also undergo a third-party security audit to ensure it does not introduce new vulnerabilities into the environments it is meant to protect. As smart home adoption continues to rise, HomeIDS offers a practical and extensible platform for strengthening baseline security in residential networks.

9. Acknowledgements

I would like to express my sincere gratitude to Dr Brenden Kuerbis, Jason Reed and my peers in Group 11 for their guidance and constructive feedback throughout the semester. Their support helped shape the direction of this project and encouraged me to explore both practical implementation and critical evaluation of the system.

I am also thankful to every professor, teaching assistant, and group member I’ve had the opportunity to work with during my time at Georgia Tech. Their insights and collaboration have been instrumental in deepening my understanding of cybersecurity.

The challenges faced while building HomeIDS were complex but deeply rewarding. This project has been one of the most valuable and formative experiences of my time in the Georgia Institute of Technology Master's in Cybersecurity program and I look forward to applying the technical skills, design strategies, and critical thinking gained from this program—and from this project in particular—to future work in cyber defense.

10. Disclaimer

While all system design, experimentation, and analysis are my original work, I used AI tools to assist with both development and documentation.

OpenAI's ChatGPT was used to help refine technical writing, clarify explanations, and format portions of the report. It was also used at times to troubleshoot coding issues, brainstorm implementation strategies, and suggest optimizations—particularly during the development of the detection engine and dashboard integration.

All final design decisions, code implementations, and analytical conclusions reflect my own judgment, testing, and revisions.

References

- [1] R. M. U. Cybersecurity, "Introduction to Smart Home Cybersecurity Risks," Medium, Aug. 20, 2023. [Online]. Available: <https://medium.com/@RocketMeUpCybersecurity/a88a0feb6f1f>
- [2] J. O. Agyemang, J. J. Kponyo, G. S. Klogo, and J. O. Boateng, "Lightweight rogue access point detection algorithm for WiFi-enabled Internet of Things (IoT) devices," *Internet of Things*, vol. 11, p. 100200, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/abs/pii/S2542660518301501?via%3Dihub>
- [3] Cybersecurity and Infrastructure Security Agency (CISA), "Heightened DDoS Threat Posed by Mirai and Other Botnets," Oct. 14, 2016. [Online]. Available: <https://www.cisa.gov/news-events/alerts/2016/10/14/heightened-ddos-threat-posed-mirai-and-other-botnets>
- [4] IoT Analytics, "Number of Connected IoT Devices," [Online]. Available: <https://iot-analytics.com/number-connected-iot-devices/>
- [5] World Economic Forum, "How the Internet of Things is being exploited on the dark web," May 2024. [Online]. Available: <https://www.weforum.org/stories/2024/05/internet-of-things-dark-web-strategy-supply-value-chain/>
- [6] Check Point Research, "The Tipping Point: Exploring the Surge in IoT Cyberattacks Plaguing the Education Sector," [Online]. Available: <https://blog.checkpoint.com/security/the-tipping-point-exploring-the-surge-in-iot-cyberattacks-plaguing-the-education-sector/>
- [7] J. Harp, "5 Worst IoT Hacking Vulnerabilities," IoT For All, [Online]. Available: <https://www.iotforall.com/5-worst-iot-hacking-vulnerabilities>
- [8] M. Gergova, "Smart Homes at Risk to Hackers," SensorsTechForum, Oct. 12, 2021. [Online]. Available: <https://sensortechforum.com/smart-homes-at-risk-to-hackers>
- [9] Which?, "How the smart home could be at risk from hackers," Which? News, Jul. 30, 2021. [Online]. Available: <https://www.which.co.uk/news/article/how-the-smart-home-could-be-at-risk-from-hackers-akeR18s9eBHU>
- [10] "Building a Network Scanner using Scapy," *ThePythonCode*, [Online]. Available: <https://thepythoncode.com/article/building-network-scanner-using-scapy>.
- [11] B. Harath, "Art of Packet Crafting with Scapy," [Online]. Available: <https://0xbharath.github.io/art-of-packet-crafting-with-scapy/index.html>.
- [12] Stratosphere Lab, "IoT-23 Dataset," *Stratosphere IPS*, [Online]. Available: <https://www.stratosphereips.org/datasets-iot23>.
- [13] CIC Research, "CIC IoT-IDAD Dataset 2024," [Online]. Available: <http://cicresearch.ca/IOTDataset/CIC%20IoT-IDAD%20Dataset%202024/Dataset/>.
- [14] "Scapy: Python-based interactive packet manipulation program," *Scapy*, [Online]. Available: <https://scapy.net/>.
- [15] OWASP, "IoTGoat," GitHub repository, [Online]. Available: <https://github.com/OWASP/IoTGoat>.

[16] R. Moran, Vice President of Sales Engineering at Fortinet, residential Iot IDS needs, May 2025.

[17] Wireshark Q&A, “How to turn on monitor mode and decrypt 802.11,” *Ask Wireshark*, [Online]. Available: <https://ask.wireshark.org/question/8698/how-to-turn-on-monitor-mode-and-decrypt-80211/>.

[18] Ask Ubuntu, “Help understanding iptables command,” AskUbuntu.com. [Online]. Available: <https://askubuntu.com/questions/847118/help-understanding-iptables-command>.

[19] phoenixNAP, “iptables port forwarding,” phoenixNAP Knowledge Base. [Online]. Available: <https://phoenixnap.com/kb/iptables-port-forwarding>.

[20] DB-IP, “DB-IP Geolocation Database,” DB-IP.com. [Online]. Available: <https://db-ip.com/>.

[21] OISF, “Suricata,” Suricata.io. [Online]. Available: <https://suricata.io/>.

[22] Python Software Foundation, “threading — Thread-based parallelism,” Python 3. [Online]. Available: <https://docs.python.org/3/library/threading.html>

[23] SQLite Consortium, “About SQLite,” *SQLite.org*, [Online]. Available: <https://sqlite.org/about.html>.

Appendices

Appendix A – Device Deployment Photo



Figure A1: HomeIDS Appliance in Use

This figure shows the HomeIDS prototype deployed on a Raspberry Pi 4. The device is connected via Ethernet to the home router and configured to host a dedicated Wi-Fi access point. A USB Wi-Fi adapter is used for AP broadcasting, and a passive cooling case is installed for thermal stability.

Appendix B – Full alert corpus

The complete alert log generated during PCAP replay and live attack simulations—including Suricata and behavioral engine outputs—is available at:

[Google Docs – Full Alert Corpus](#)

Appendix c – Full code.

The complete HomeIDS code repository—including the main orchestrator, detection engines, dashboard UI, and evaluation artifacts—is publicly available at:

[GitHub – HomeIDS Code Repository](#)

