Detecting and Mitigating Behavioral Inference from Smart Home IoT Traffic Metadata

Jordy M. Rodríguez De Gracia
Master in Computer Science
Advisor: Jeffrey Duffany, Ph.D.
Polytechnic University of Puerto Rico
Graduate Project EXPO, May 2026

Abstract *— The use of smart home devices in the past years, has grown on the day-to-day basis of a household. These IoT devices encrypt their cloud communications, but metadata like timing, frequency and protocol leak on the network with occupant routines. The purpose of this project is to demonstrate this privacy gap using a real residential network with five commercial IoT devices (iHome smart plugs, WiZ smart bulbs, and a TP-Link KL 125 plug) and then fortify the vulnerability, using a Raspberry Pi Zero 2 deployed as an in-network privacy shield referred to as "SecurePlug". Forty-eight hours of pre-deployment traffic revealed that iHome plugs broadcast UDP heartbeats at an almost constant 1.7 seconds interval, and that WiZ bulbs transmit mDNS announcement bursts containing device fingerprints in plaintext during every state change. A Python inference pipeline extracted 28 behavioral events from this traffic and reconstructed the occupant's daily routine (wake up, departure, arrival, and sleep) with 75% confidence. The real threat is metadata, not payload content. SecurePlug counters this with three interception layers: random jitter on iHome heartbeats (raising timing variance by 2,500x), jitter on WiZ UDP heartbeats, and full suppression of WiZ mDNS bursts. After 55 hours of shield operation, departure was undetectable on all four captured days, behavioral events dropped 64%, and inference confidence fell from 75% to 62%, insufficient to reconstruct a complete daily profile.*

Keywords *— Behavioral Inference, IoT, IoT Privacy, Network Metadata.*

INTRODUCTION

Nowadays, we might have used or installed some sort of smart home devices to better our way of living. Smart home usage has expanded at a faster pace, but on some occasions the security expectations set in these products rarely match the reality of residential network environments. Companies may have addressed payload privacy, most devices communicate via TLS or proprietary encrypted channels, but traffic metadata remains largely unprotected. Study on Information Exposure (done by article), demonstrated that even without payload inspection, device level network flows are sufficient to infer user activities with high accuracy [1]. This type of vulnerability can be troubling, because is invisible to the homeowner and requires no exploitation of the device itself, just an attacker with passive access to the Wi-Fi network sees enough.

The project intends to pursue that threat in a controlled but genuinely residential setting. Purchases five IoT devices, installed in a home network, and monitored for 48 hours using Wireshark before any intervention. The original hypothesis was that MQTT plaintext traffic on port *1883* would be the primary exposure, that was immediately refuted by the capture data. Instead, the exposure came from two directions: the constant regularity of IHome's UDP heartbeat cadence, and the content mDNS announcements that WiZ bulbs multicast during every on/off transition. The project then pivoted to address these findings, which showed if behavioral inference is possible even when all payloads are encrypted, then encryption alone is not a sufficient defense.

The network privacy shield referred to as "SecurePlug" contribution is a software running on a Raspberry Pi Zero 2, that intercepts and modifies the metadata traffic in real time without decrypting or altering the payload. The approach is validated by a second capture taken of 55 hours, while the shield

was active, analyzed using the identical inference pipeline. The remainder of the article will cover the experimental setup and capture methodology, the threat model and attack mechanism, the shield architecture and implementation, the behavioral inference pipeline, the comparative results, and conclusions.

Background and Related Work

When researching network traffic analysis of IoT devices, there is an established research history on the topic. Sivanathan's study showed that network behavior at scale across multiple device categories and flow level features are sufficient for accurate device identification and activity classification [2]. Their following doctoral work extended this to continuous behavioral monitoring and demonstrated that temporal regularity is the primary distinguishing factor [3].

On the topic of privacy threat, Apthorpe, Reisman, and Feamster in their study narrowed the lens to smart home specifically and showed that a passive observer on a home Wi-Fi network can infer device states and user actions with simple traffic rate analysis [4]. In Dong, Chen study, looked at how local service discovery protocols like mDNS expose device information to anyone on the same network, a finding that applies directly onto the WiZ bulb behavior capture in this project [5]. The defense side, Bezawada proposed that traffic shaping can meaningfully degrade inference accuracy [6]. That is the core principle for SecurePlug put into practice with off the shelf hardware. Beyond traffic patterns, the scope of what a local observer can extract is broader than most users assume. A survey done by Acar, Huang, Li, Narayanan, and Feamster mapped out those information classes across smart home environments [7], and a following study confirmed that metadata exposure holds up even when encryption is done right [1]. With this in mind, SecurePlug implements a low-cost response to the exposure found.

Set Up and Capture Methodology

Network Environment and Devices

The testbed used for this project was a real residential home network running on a 192.168.0.x subnet, managed by an Arris router. Five IoT devices were deployed for the experiment: two iHome Flow smart plugs, two Philips WiZ LED bulbs, and one TP-Link KL125 smart plug. All devices operated under normal household conditions, used as intended, with no artificial traffic manipulation. A Windows laptop running Wireshark 4.2.4 and Python 3.13.5 handled all captures and analysis.

Table 1
IoT Devices Deployed in the Testbed and their Observed Traffic Characteristics

Device	Model	Protocol	Pattern	Risk
iHome Plug ×2	iHome Flow	UDP Broadcast	1.7 s heartbeat → 255.255.255.255:6667	HIGH
WiZ Bulb ×2	Philips WiZ LED 60W	mDNS / UDP	mDNS burst on state change + HB :38900	HIGH
TP-Link KL125	KL125	TLS 1.3	Encrypted cloud traffic	LOW

Threat Model

The assumed adversary in this scenario is a passive observer with access to the home Wi-Fi network, like a neighbor, an ISP monitoring unencrypted local broadcast traffic, or a compromised device already on the same segment. This attacker does not need to break encryption, exploit device firmware, or have any physical access. Ren et al. established that this type of threat profile is realistic and sufficient to carry out behavioral inference [1]. The target information is straightforward: the occupant's daily routine, when they wake up, leave the house, return and go to sleep. As noted by Acar et al., that kind of information can support burglary planning, stalking, or insurance fraud [7].

One key finding from the pre-capture phase is worth noting: no MQTT plaintext traffic was found during the entire capture. Every device communicated either over TLS or via UDP without a broker, consistent with the broader industry shift toward transport encryption. However, as the following sections show, metadata alone carries enough information to reconstruct behavioral patterns without decrypting a single byte.

Wireshark Capture Configuration

Two separate captures were conducted using identical Wireshark settings. The before shield capture ran April 14-16, 2026, and the after-shield capture ran April 23-26, 2026. Wireshark was set to save a new file every 12 hours, producing five segments per capture. A display filter was applied to isolate only the five IoT devices IPs:

Before: host 192.168.0.2 or host 192.168.0.3 or host 192.168.0.14 or host 192.168.0.15 or host 192.168.0.11

One thing to account for between captures: device IPs changed due to DHCP lease renewal. This was confirmed by cross referencing MAC addresses in the router client list and after shield filter was updated accordingly. MAC addresses serving as the consistent device identifiers across both captures.

TRAFFIC ANALYSIS

iHome Smart Plug: UDP Heartbeat Pattern

The most noticeable finding from the pre shield capture was how mechanical the iHome traffic looked. Both plugs were broadcasting UDP packets to *255.255.255.255* on port *6667* at almost exactly 1.7 seconds per plug, taking turns in a predictable alternating sequence. Over the full 48 hour capture that came out to 69,468 iHome packets, 99.4% of all traffic captured. The inter packet interval variance measured across a 100 packet sliding window was approximately 0.001, which is essentially a clock, not a network device. To process the raw Wireshark export, a Python script was created called *parse_capture.py* to load the five CSV files, assign absolute timestamps to each packet, and classify them by device and event type. The output file *iot_events.csv* is used for all the subsequent analysis.

```python
def load_and_combine_csvs(file_paths: list, start_times: list) -> pd.DataFrame:

    all_data = []

    for i, file_path in enumerate(file_paths):
        print(f"Loading {file_path}...")
        df = pd.read_csv(file_path)

        # Add base time for this file
        df['base_time'] = start_times[i]
        df['file_num'] = i + 1

        # Convert relative time to absolute datetime
        df['abs_time'] = df.apply(
            lambda row: row['base_time'] + timedelta(seconds=float(row['Time'])),
            axis=1
        )

        all_data.append(df)
        print(f"  - Loaded {len(df)} packets")
```

Figure 1

Load each of the five Wireshark CSV files and converts the relative Time column into an absolute datetime by adding each file's known start time, producing a unified timeline of 69,860 events covering April 14–16, 2026.

That regularity is what makes it a vulnerability. When the occupant leaves the house, device interactions stop, no app commands or no state changes. But the iHome plugs keep broadcasting on schedule. The inference engine picks that up as an extended gap with no activity. On April 15, a continuous 88-minute gap was detected between 06:31 and 08:00, which produced a clear departure signal. No encryption changes that, because the silence itself is the data.

WiZ Smart Bulbs: mDNS Burst Fingerprinting

The WiZ Smart Bulbs exposed information through two separate mechanisms. The first is a UDP heartbeat sent to the subnet broadcast address (*192.168.0.255*) on port *38900* at regular intervals. The second is more concerning being every time a bulb turns on or off, it fires an mDNS burst on port *5353* to multicast group *224.0.0.251*. These bursts carry the devices MAC address, IP address, model identifier, and Matter protocol metadata, all in plain text. Any device passively listening on the local network receives a timestamped log of every bulb state change without doing anything at all.

```python
def classify_event_type(row: pd.Series) -> str:
    """Classify the type of event based on protocol and destination."""
    protocol = row['Protocol']
    destination = row['Destination']

    if protocol == 'MDNS':
        return 'discovery_announcement'
    elif protocol == 'UDP' and destination == '255.255.255.255':
        return 'broadcast_heartbeat'
    elif protocol == 'UDP' and destination.endswith('.255'):
        return 'subnet_broadcast'
    elif protocol == 'IGMPv2':
        return 'multicast_membership'
    else:
        return 'other'
```

Figure 2

Labels Each Packet by What It Represents. iHome UDP broadcast to 255.255.255.255 becomes broadcast_heartbeat. WiZ mDNS announcements become discovery_ announcements. WiZ subnet broadcasts to 192.168.0.255 become subnet_broadcast. These labels feed directly into the inference rules.

Taken together, these dual exposures feed directly into the behavioral inference rules described in section 5. The first mDNS burst cluster in the morning window (5-11am) signals wake-up, a burst cluster after an afternoon quiet period, indicates arrival, and the last cluster of the night signals sleep.

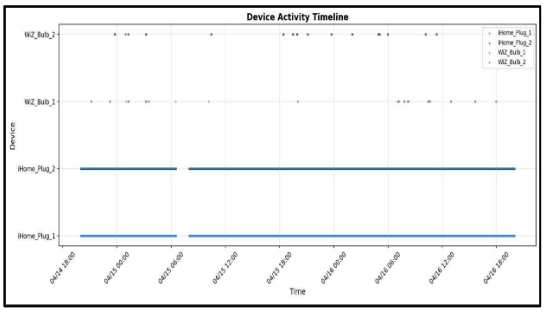

Figure 3

Device Activity Timeline (before shield), Each Dot Represents One Packet Event. The regular iHome cadence and discrete WiZ burst clusters are visible.

BEHAVIORAL INFERENCE

Parsing and Event Extraction

Once the Wireshark captures were exported as CSV files, parse_capture.py* was run to convert the raw packet data into a structured event dataset. The script mapped each source IP to its corresponding device using the MAC verified IP table, converted Wireshark's relative timestamps into real clock times, and labeled each packet with an event type based on its protocol and destination. The result was *iot_events.csv*, containing 69,860 timestamped and classified events covering the full 48 hour before shield capture.

It is worth noting that this same script was also used for the after-shield capture, with one adjustment, the device IP table was updated to reflect the new DHCP assignments after the second capture began:

```python
def classify_device(row: pd.Series) -> str:
    """Classify device based on IP address."""
    source = row['Source']

    device_map = {
        '192.168.0.2': 'iHome_Plug_1',
        '192.168.0.3': 'iHome_Plug_2',
        '192.168.0.11': 'KL125_Plug',
        '192.168.0.14': 'WiZ_Bulb_2',
        '192.168.0.15': 'WiZ_Bulb_1',
    }

    return device_map.get(source, 'Unknown')
```

Figure 4

Maps Each Source IP to its Device Name for the Week Before Shield Capture. For the after-shield run, IPs were updated.

The logic behind each rule follows naturally from what was observed in the traffic. Wake up and sleep are detected through WiZ bulb activity, because the lights are typically the first thing turned on in the morning and the last thing turned off at night. Departure is detected as a gap, when all four devices go quiet for more than 60 minutes during the day, it means no one is home. Arrival is the reverse, a burst of WiZ activity in the afternoon or evening after a period of silence. The methodology is grounded in the behavioral profiling framework developed by Sivanathan [3] and the burst detection approach described by Bezawada et al [6].

Inference Rules

A second script called *infer_behavior.py*, applied four rules to the parsed event data to detect behavioral events. The same rules and thresholds were used on both the before and after shield datasets to keep the comparison fair. The rule configuration is defined at the top of the script:

```
RULES = {
    'wake_up': {
        'devices': ['WiZ_Bulb_1', 'WiZ_Bulb_2'],  # Light activity indicates wake-up
        'event_types': ['discovery_announcement', 'subnet_broadcast'],
        'time_range': (5, 11),  # 5:00 AM - 11:00 AM
        'min_packets': 3,  # Minimum packets to trigger
        'description': 'First significant light activity in morning'
    },
    'sleep': {
        'devices': ['WiZ_Bulb_1', 'WiZ_Bulb_2'],
        'event_types': ['discovery_announcement', 'subnet_broadcast'],
        'time_range': (21, 3),  # 9:00 PM - 3:00 AM ( midnight)
        'min_packets': 3,
        'description': 'Last significant light activity at night'
    },
    'departure': {
        'devices': ['iHome_Plug_1', 'iHome_Plug_2', 'WiZ_Bulb_1', 'WiZ_Bulb_2'],
        'gap_threshold_minutes': 60,  # Gap longer than this, indicates departure
        'time_range': (6, 14),  # 6:00 AM - 2:00 PM
        'description': 'Extended gap in device activity during morning/midday'
    },
    'arrival': {
        'devices': ['WiZ_Bulb_1', 'WiZ_Bulb_2'],
        'event_types': ['discovery_announcement', 'subnet_broadcast'],
        'time_range': (15, 23),  # 3:00 PM - 11:00 PM
        'min_packets': 5,  # Higher threshold for arrival
        'description': 'Significant light activity after period of quiet'
    }
}
```

Figure 5
Rules block from infer_behavior.py

Before Shield Results

Running *infer_behavior.py* on the 69,860 events before shield dataset produced 28 detected behavioral events across three days, with the following profile:

Table 2
Before Shield Behavioral Profile Inferred from 69,860 IoT Traffic Event

Event	Average Time	Detection Rate	Confidence
Wake-up	08:11	1/2 days (partial cap)	HIGH
Departure	06:30 (88-min gap)	1/1 eligible days	HIGH
Arrival	19:15	2/2 days	HIGH
Sleep	02:24	2/2 days	HIGH

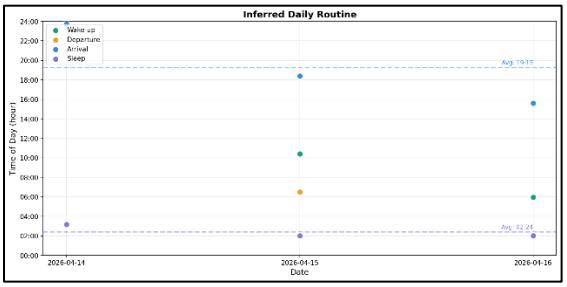

Figure 6
Inferred Daily Routine (before shield) Each Colored Dot Represents a Detected Event, Dashed Horizontal Lines Show Per-Event Averages. The departure signal is prominent April 15.

The departure event on April 15 is the most significant result from this phase. A continuous 88 gap with no device activity during daytime hours was enough for the algorithm to flag it as a departure with high confidence.

SECURE PLUG: THREE LAYER PRIVACY SHIELD

Hardware and Deployment

The SecurePlug shield runs on a Raspberry Pi Zero 2, connecting to the home network exclusively through Wi-Fi on wlan0*. The device was configured headless using Raspberry Pi OS Lite, accessed remotely via SSH at *secureplug.local* throughout the entire capture window.

The shield script was deployed as a permanent *systemd* service so it could run autonomously without an active SSH session. The *After=network.target* directive ensures the shield does not attempt to open network sockets before *wlan0* is fully active, and *Restart=always* keeps the service running across reboots without any manual intervention:

```
Systemd service file (/etc/systemd/system/secureplug-shield.service):
    [Unit]
    Description=SecurePlug IoT Privacy Shield
    After=network.target
    [Service]
    ExecStart=/usr/bin/python3 /home/pi/secureplug_shield.py
    Restart=always
    User=root
    [Install]
    WantedBy=multi-user.target
```

Figure 7
Script Header Documents All Three Protection Layers and Their Results. The embedded system service configuration shows configuration which prevents the shield from opening sockets before wlan0) is active and which kept the service running without a single restart.

The complete shield is implemented in *secureplug_shield.py*, running three independent protection layers in parallel threads, each targeting a different traffic pattern identified in Section 4.

Layer 1: iHome Heartbeat Jitter

Layer 1 targets the mechanical iHome heartbeat pattern identified in Section 4 as the primary departure signal. The shield binds a receive socket to

0.0.0.0:6667 with *SO_REUSEADDR* and *SO_BROADCAST*, intercepts every UDP broadcast from the iHome plugs, a re-emits each packet after a random delay between 0.5 and 3.0 seconds. The original payload is forwarded byte for byte, only the timing is changed. This is the traffic shaping countermeasure described by Bezawada et al. [6], applied at the socket layer without modifying device firmware or cloud infrastructure.

Each intercepted packet is handled in its own daemon thread, so delays run independently and do not block the receive loop:

```
def handle_ihome_packet(data, addr, fwd_sock):
    """
    eceives one iHome UDP broadcast, applies random jitter, re-emits.
    """
    delay = random.uniform(JITTER_MIN, JITTER_MAX)
    time.sleep(delay)

    try:
        fwd_sock.sendto(data, ('255.255.255.255', IHOME_PORT))
        with stats_lock:
            stats['ihome_intercepted'] += 1
    except Exception as e:
        with stats_lock:
            stats['ihome_errors'] += 1
        logger.warning(f'iHome forward error: {e}')
```

Figure 8

Applies a Random Delay Between JITTER_MIN (0.5s) and JITTER_MAX (3.0s) Before Re-Emitting Each iHome Packet to 255.255.255.255:6667. The original payload is preserved byte-for-byte. Each call runs in its own.

The shield also tracks a sliding window of the last 100 iHome inter-packet timestamps and logs every 60 seconds, providing proof that the jitter is working:

Over the 55 hours after shield captur, Layer 1 intercepted and reshuffled 74,881 iHome packets. Every one of those packets had its timing altered before being re-emitted.

Table 3

iHome Timing Variance Before and After Shield

Measurement	Before Shield	After Shield
iHome timing variance (socket layer)	≈ 0.001	≈ 2.5134 (+2,500×)
iHome timing variance (wire, Wireshark delta)	2.054	2.460 (+20%)
iHome means interval (wire)	2.534 s	2.695 s (+6%)
Packets intercepted over 55 hours	N/A (no shield)	74,881

Layer 2: WiZ Heartbeat Jitter

Layer 2 applies the same jitter logic to WiZ heartbeat on port *38900*, which broadcasts to the subnet address *192.168.0.255*. A separate receive socket filters for WiZ source IPs and re-emits each packet with the same [0.5, 3.0] seconds, same thread per packet design, same payload preserving approach. Over the 55 hours, 146 WiZ heartbeats were intercepted and reshuffled.

Layer 3: WiZ mDNS Burst Suppressor

Layer 3 works differently from the first two, instead of reshuffling packets, it drops them entirely. The Pi joins the mDNS multicast group *224.0.0.251* using the *IP_ADD_MEMBERSHIP* socket option, making it the primary recipient of all mDNS traffic on the local network:

```
def layer3_wiz_mdns_suppressor():
    """
    Layer 3: Joins mDNS multicast group 224.0.0.251 on port 5353.
    """
    logger.info('Layer 3 — WiZ mDNS Suppressor starting (multicast :5353)')

    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind(('', MDNS_PORT))

    # Join mDNS multicast group on all interfaces
    mreq = struct.pack('4sL',
                       socket.inet_aton(MDNS_MULTICAST),
                       socket.INADDR_ANY)
    sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)

    logger.info('Layer 3 — mDNS Suppressor active. Joined 224.0.0.251:5353')

    while True:
        try:
            data, addr = sock.recvfrom(65535)
            if addr[0] in WIZ_IPS:
                # SUPPRESS — do NOT forward
                with stats_lock:
                    stats['wiz_mdns_suppressed'] += 1
            # Non-WiZ mDNS traffic is silently ignored (not suppressed)
        except Exception as e:
            with stats_lock:
                stats['mdns_errors'] += 1
            logger.error(f'Layer 3 recv error: {e}')
            time.sleep(0.1)
```

Figure 9

Joins Multicast Group, Making the Raspberry Pi the Primary Recipient of All mDNS Traffic on the Network. Packets arriving from WiZ source IPs are received and counted but never forwarded.

Any mDNS packet arriving from a WiZ source IP is received and dropped. The device MAC address, IP address, model identifier, and Matter protocol metadata never reach any other device on the network. Over 55 hours, 337 mDNS bursts from WiZ bulb 1 were suppressed. WiZ bulb 2 experienced intermittent Wi-Fi disconnections during the capture, during those reconnection events the bulb transmitted mDNS announcements before

the shields socket could intercept them, a known limitation discussed in section 7*.

Result: Before and After Shield

After Shield Inference Results

The same pipeline used in section 5, parse_capture.py feeding into infer_behavior.py was run on the after-shield dataset of 76,396 events. The same thresholds and time window rules were kept unchanged so the comparison would be fair. The per day results are shown in table 4.

Table 4
After Shield Behavioral Inference Results Per Day

Date	Wake-up	Departure	Arrival
April 23	Not detected	Not detected	23:52
April 24	05:23	Not detected	15:16
April 25	07:02	Not detected	15:11
April 26	05:07	Not detected	Not detected

Complete Before vs After Comparison

Table 5 summarizes all measured metrics across both captures:

Table 5
Complete Comparison, Two Primary Success Metrics are Departure and Inference Confidence Degradation

Metric	Before Shield	After Shield	Result
Total packets captured	69,860	76,396	Expected (longer window + re-emit)
iHome timing variance	≈ 0.001	≈ 2.5134	+2,500× increase
Departure detected	YES — 88 min gap	NOT DETECTED (0/4 days)	Eliminated
Behavioral events detected	28	10	−64% reduction
Inference confidence	75% HIGH	62% DEGRADED	No complete profile
WiZ mDNS bursts suppressed	N/A	337	Layer 3 effective
Complete behavioral profile	YES	NO	Goal achieved
Shield uptime / errors	N/A	55 hrs / 0 errors	Stable

A few things worth pointing out from these numbers is the total packet count went up after the shield. That is expected, because the shield re-emits every intercepted iHome packet after a delay, therefore some packets appear twice in the Wireshark capture. The Behavioral events dropped from 28 to 10, a 64% reduction, and confidence fell from 75% to 62%. At 62%, the inference engine can detect some isolated events but cannot construct a complete occupant profile, which is the threshold that matters for the privacy threat.

After Shield Visualizations

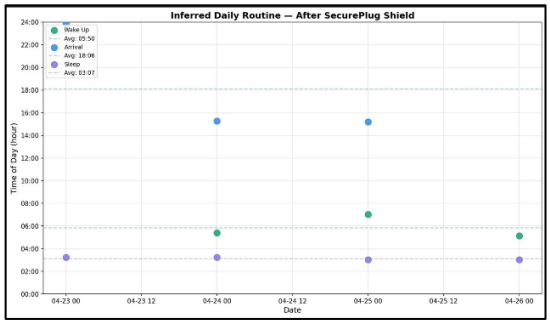

Figure 10
Inferred Daily Routine (after shield). Departure is absent from all four days. The wake-up and sleep estimates show more variation than the before shield profile, reducing their usefulness as behavioral predictors.

Conclusions

This project sets out to show two things: that behavioral inference from IoT traffic metadata is achievable in a real home network, and that it can be stopped with hardware anyone can buy. The before shield capture produced a complete occupant profile (wake up, departure, arrival, and sleep), from Wi-Fi traffic, with 75% confidence, without reading a single encrypted payload. The after shield capture, using the same devices and the same inference pipeline, could not reproduce that profile. Departure was undetectable on all four days, behavioral events dropped 64%, and confidence fell to 62%. The pivot from the original MQTT hypothesis also produced a finding worth noting. When no MQTT plaintext was found in the captures, the project adapted to target the metadata patterns that were present, iHome UDP heartbeat regularity and WiZ mDNS burst content.

Limitations

WiZ bulb 2 experienced intermittent Wi-Fi disconnections during the after shield capture. When the bulb reconnected, it transmitted mDNS announcements before the shield multicast socket had a chance to intercept them, with accounts for the 343 mDNS packets that appear in Wireshark despite the suppressor being active. A more complete implementation would need to monitor device

connectivity and handle reconnections events explicitly.

The inference rules were tuned to this specific household's traffic patterns. The departure threshold and the mDNS burst threshold reflect what observed in these captures, applying them to a different home would likely require recalibration. Finally, the shield was tested against a passive observer only. An active adversary who injects traffic, queries devices directly, or uses side channels outside the local Wi-Fi network would require countermeasures beyond what SecurePlug currently provides.

FUTURE WORK

The most immediate addition would be support for additional protocol classes. An adaptive jitter, where the delay range adjusts based on detected inference attempts, would make the shield more resistant to adversaries who account for fixed jitter ranges. Work a user-friendly browser configuration interface like a home router admin panel, allowing non-technical users to deploy SecurPlug without SSH access.

REFERENCES

[1] J. Ren et al., "Information exposure from consumer IoT devices: A multidimensional, network-informed measurement approach," in Proc. ACM Internet Measurement Conf. (IMC), Amsterdam, Netherlands, 2019, pp. 267–279.

[2] A. Sivanathan et al., "Classifying IoT devices in smart environments using network traffic characteristics," *IEEE Trans. Mobile Comput*., vol. 18, no. 8, pp. 1745–1759, Aug. 2019.

[3] A. Sivanathan, "Machine learning for IoT network traffic analysis," Ph.D. dissertation, School of Electrical Engineering and Telecommunications, Univ. of New South Wales, Sydney, Australia, 2020.

[4] N. Apthorpe, D. Reisman, and N. Feamster, "Closing the blinds: Four strategies for protecting smart home privacy from network observers," in *Proc. Privacy Enhancing Technologies Symposium (PETS)*, Minneapolis, MN, 2017.

[5] X. Dong et al., "Your IoTs are (not) mine: On the remote binding between IoT devices and users," in *Proc. ACM Asia Conf. Computer and Communications Security (ASIA CCS)*, Taipei, Taiwan, 2020, pp. 717–729.

[6] B. Bezawada et al., "IoTSense: Behavioral fingerprinting of IoT devices," *in Proc. ACM Workshop on Attacks and Solutions in Hardware Security* (ASHES), Toronto, Canada, 2018, pp. 1–8.

[7] U. Acar et al., "Peek-a-boo: I see your smart home activities, even encrypted!" in *Proc. ACM Internet Measurement Conf.* (IMC), Boston, MA, 2018, pp. 537–543.