Final Security Test Specification and Tools Report

Mobile Platform	Hybrid Application ; IoT System ; Android App ; IoT System
Application domain type	Smart Wearables
Authentication	Yes
Authentication schemes	Biometric-based authentication ; Channel-based authentication ; ID-based authentication ; Channel-based authentication ; Biometric-based authentication ; Factors-based authentication
Has DB	Yes
Type of database	SQL (Relational Database)
Which DB	SQLite
Type of information handled	Personal Information ; Confidential Data ; Personal Information ; Confidential Data ; Critical Data
Storage Location	Both
User Registration	Yes
Type of Registration	The users will register themselves
Programming Languages	Dart ; Kotlin
Input Forms	Yes
Upload Files	No
The system has logs	Yes
The system has regular updates	Yes
The system has third-party	Yes
System Cloud Environments	Hybrid Cloud
Hardware Specification	Yes
HW Authentication	Basic Authentication (user/pass)
HW Wireless Tech	3G ; 4G/LTE ; 5G ; Bluetooth ; Wi-Fi ; Bluetooth ; GPS ; LoRa ; 3G ; 4G/LTE ; 5G ; Bluetooth ; Wi-Fi ; Wi-Fi ; Bluetooth ; GPS
Device or Data Center Physical Access	Yes

Security Testing for Cellular Jamming Attacks

1. Overview

Assess detection, mitigation, and resilience of cloud/mobile/IoT deployments to *cellular jamming* (broadband/narrowband, reactive, protocol-aware/"smart" jammers) by combining: RF instrumentation (SDR), protocol-aware testbeds (srsRAN / OpenBTS / OpenAirInterface), spectrum monitoring, detection algorithms (RSSI/EVM/ML), and higher-layer resilience tests (service failover, handover, fallback to other RATs). Key goals: detect jamming quickly, measure outage area/effect, validate mitigations (frequency hopping, alternative paths, multi-SIM failover, redundant connectivity), and create actionable remediation & detection rules.

---

2. Short Hardware & Software Prerequisites (lab)

- RF shielded environment (Faraday cage) or licensed test frequencies.
- SDRs for transmission/reception: **USRP B210 / N210 / X310 (Ettus)**, **HackRF One**, **LimeSDR**, **bladeRF**. (USRP recommended for research-grade TX control & power.)
- Software stacks: **srsRAN (srsLTE / srsRAN)**, **OpenBTS / OpenAirInterface** for creating test eNodeB/gNB and UEs in lab.
- RF analysis: GNURadio, SigDigger, rtl_power/rtl_sdr tools, Wireshark (for decoded S1 / RRC messages when available).
- Jamming test software (research implementations / proof-of-concept only): JamRF / GNUradio-based jammers (use only in shielded / authorized lab).
- Detection & analytics: tools or code to measure RSSI/RSRP/RSRQ, BER, Packet Loss, and **EVM** per resource block (recent works show EVMâ†'useful for detection).

---

3. High-level Testing Phases / Workflow

1. **Design & authorization** — obtain written approvals and define test band, time, and containment (Faraday cage). Document rules of engagement. (Mandatory.)
2. **Baseline collection** — measure normal KPI: RSRP/RSRQ/RSSI, throughput, packet loss, latency, and EVM; collect traces from target UEs, IoT nodes, and edge cloud services. Store seed corpus for later comparison.
3. **Instrumented test eNodeB / gNB setup** — deploy srsRAN/OpenAirInterface/OpenBTS to create controlled cell(s). Connect test UEs (real phones in lab or simulated UEs).
4. **Jammer types & staging (lab-only)** — run controlled jammers in increasing scope:

- *Narrowband constant tone* (single RB / single carrier).
- *Wideband noise* (entire channel band).
- *Reactive* (jam only when a packet is detected).
- *Smart/targeted* (jam specific control channels such as PSS/SSS/PBCH or SIB/paging windows). Use SDR (USRP/HackRF) with prebuilt GNUradio flows or research code (JamRF, custom GNURadio).

5. **Detection experiments** — validate detection algorithms: RSSI thresholds, packet-level metrics, throughput/BER shifts, and **EVM-based** per-RB detection (EVM shown effective for LTE/5G jamming detection). Compare simple threshold methods vs ML classifiers trained on spectrograms/IQ or KPI features.
6. **Service-level impact & resilience** — measure mobile app behavior, IoT telemetry dropouts, cloud-side alarms, SIEM events; validate failover/retry, multi-SIM handover, or satellite fallback where applicable.
7. **Mitigation validation**— test frequency hopping-like countermoves in lab (if applicable), increased redundancy, edge caching, and detection → blacklisting of affected cells.
8. **Reporting & safe clean-up** — provide reproducible test manifests and remove any active jamming devices, verify network recovery.

---

3. Practical Testing Setup (playbook - safe lab only; condensed)

1. **Get approvals** — written authorization + reserved test frequency or Faraday cage. (Do not proceed without this.)
2. **Deploy controlled cell** — run srsRAN eNodeB on isolated frequency, attach test UEs. Command examples (lab):

- `sudo apt-get install srslte` then configure `enb.conf` and run `sudo srsepc` + `sudo srseNB` (see srsRAN docs).

3. **Baseline KPIs** — collect RSRP/RSRQ, throughput, and EVM via SDR receiver + UE logs. Save traces.
4. **Start jammer (very controlled)** — using USRP + GNURadio jamming flow (ensure power limits & containment): start with narrowband tone on a single RB; monitor network behavior; then stop. (Example GNURadio flow = JamRF / custom flow.)
5. **Detection evaluation** — run simple detectors (RSSI drop threshold) and EVM-per-RB detector (compare to baseline). Evaluate detection latency and false positives.
6. **Service impact & mitigations** — measure app timeouts / message retransmit, IoT telemetry gaps, and test fallback (e.g., switch to alternate SIM or RAT) in controlled environment. Document parameters that trigger mitigation.
7. **Automation & reproducibility** — script the scenario (Ansible/Docker) including SDR flows, start/stop times, and capture all logs/PCAPs and SDR IQ recordings for offline analysis.

---

4. Security Testing Approaches & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST / Physical	Hardware jamming (controlled, lab-only)	USRP (Ettus Research)	USRP	Both (affects Android/iOS/IoT)
Black-box / Low-cost	DAST / Physical	Proof-of-concept jamming (narrow/wideband)	HackRF One	HackRF One	Both
Gray-box / Emulation	DAST / Protocol	Controlled LTE/5G test eNodeB / gNB	srsRAN (srsLTE / srsRAN)	srsRAN	Both (test UEs + IoT modems)
Gray-box	DAST / Protocol	Open-source mobile network testbeds	OpenAirInterface / OpenBTS	OpenAirInterface / OpenBTS	Both
Black-box / Research	DAST / RF	SDR/flow-based jamming implementations	JamRF (GNUradio flows)	hJamRF	Both
Black-box / Passive	DAST / Passive	Spectrum reconnaissance & passive detection	RTL-SDR + rtl_power / gr-gsm / SigDigger	RTL-SDR / SigDigger	Both
Gray-box / Detection	DAST / Signal metrics	EVM / KPI-based detection & analytics	Custom EVM monitoring (MATLAB/Python) + SDR input	EVM method	Both
Gray-box / Network	DAST / Traffic	Service-level impact testing (app / IoT telemetry)	Wireshark / PCAP analysis	Wireshark	Both
White-box / Simulation	SAST / Model	RF & protocol simulation (no TX)	NS-3 / MATLAB / GNUradio simulation	NS-3 / GNUradio	Both
Gray-box / ML	DAST / ML detection	Spectrogram / IQ ML detectors	TensorFlow / PyTorch (custom models)	TensorFlow / PyTorch	Both

---

5. References

1. Pirayesh, H., & Zeng, H. (2022). Jamming attacks and anti-jamming strategies in wireless networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 24(2), 767â€“809.
2. Shaik, A., Borgaonkar, R., Asokan, N., Niemi, V., & Seifert, J.-P. (2016). Practical attacks against privacy and availability in 4G/LTE mobile communication systems. *NDSS*.
3. Capota, C., Popescu, M., Badula, E. M., Halunga, S., Fratu, O., & Popescu, M. (2023). Intelligent Jammer on Mobile Network LTE Technology: A Study Case in Bucharest.*Applied Sciences*, 13(22), 12286.
4. Ã–rnek, C., & Kartal, M. (2023). Securing the Future: A resourceful jamming detection method utilizing the EVM metric for next-generation communication systems. *Electronics*, 12(24), 4948.
5. Ali, A. S., Baddeley, M., Bariah, L., Lopez, M. A., Lunardi, W. T., Giacalone, J. P., & Muhaidat, S. (2022). Jamrf: performance analysis, evaluation, and implementation of rf jamming over wi-fi. *IEEE Access*, 10, 133370-133384.
6. Lichtman, M., Jover, R. P., Labib, M., Rao, R., Marojevic, V., & Reed, J. H. (2016). LTE/LTE-A jamming, spoofing, and sniffing: threat assessment and mitigation. *IEEE Communications Magazine*, 54(4), 54-61.

Security Testing for Wi-Fi Jamming Attacks

1. Overview

A Wi-Fi Jamming Attack is a type of Denial-of-Service (DoS) attack where an attacker deliberately disrupts wireless communication by flooding the airwaves with interference. Security testing is essential to detect vulnerabilities, protect network availability, and ensure resilient wireless infrastructure.

Why Is Wi-Fi Jamming Security Testing Important?

1. Ensures Network Availability;
2. Protects Against Targeted Disruption;
3. Supports Regulatory Compliance;
4. Improves Incident Response;
5. Mitigates Broader Cyber Threats.

2. Security Testing Approaches & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST	Constant / random RF jamming (PHY-level)	USRP (UHD), HackRF	Ettus USRP, HackRF	Both
Black-box	DAST	Reactive / protocol-aware jamming (selective)	SDR + custom GNU Radio/SDR scripts	GNU Radio	Both
Black-box / Gray-box	DAST	802.11 management-frame attacks (deauth/disassoc)	aircrack-ng (aireplay-ng), mdk3/mdk4	Aircrack-ng, MDK4	Both (Linux); Android (rooted) for mobile testing
Black-box	DAST / Network	Frame injection / replay (DoS)	Scapy, tcpreplay, Wireshark	Scapy, Wireshark	Both
Gray-box	DAST / Runtime	Jamming detection & classification (TinyML / Edge-AI)	TensorFlow Lite, TinyML libs, Raspberry Pi + RTL-SDR	TensorFlow Lite	Android (both), IoT
White-box	SAST	Firmware / driver review for resiliency (retries, backoff)	CodeQL, SonarQube	CodeQL	Both
Gray-box	DAST / Hardware	Low-cost jam-POC (IoT boards) & RF shielding tests	ESP8266/NodeMCU, RF attenuators, RF spectrum analyzer	ESP8266	IoT (affects Android/iOS clients)
Black-box	DAST / Network	Monitoring / packet capture during jamming	Wireshark, Kismet, Zeek	Kismet	Both

Gray-box	DAST / Simulation	Channel-hopping / mitigation testing	Hostapd + channel scripts, wpa_supplicant	hostapd	Both

3. Short Testing Setup

Legal & isolation first — set up a physically isolated RF test area (Faraday cage / shielded room) or use frequency bands where you have explicit permission. Jamming is illegal outdoors or on shared networks. (see Legal note below).

Testbed hardware

- Host/sensor: Linux laptop with an external PCIe Wi-Fi card supporting monitor/injection (Atheros/Intel depending on driver).
- SDR: USRP B200/B210 or HackRF for PHY-level jamming and reactive jamming experiments.
- IoT/mobile targets: Android phone(s) (include rooted device for deeper injection/monitoring), representative IoT devices (ESP8266/ESP32), iOS device only for monitoring (iOS limits active injection without jailbreaking).

3. **Baseline measurements**

- Capture normal RSSI, packet loss, throughput, and client behavior using Wireshark/Kismet before any attack runs. Record spectrum with a cheap RTL-SDR or spectrum analyzer to get baseline noise floor.

4. **Attack types to run**

- **Management-frame DoS (deauth/disassoc):** use aireplay-ng or mdk4 to inject deauth frames and measure client disconnects and reconnection behavior. Works well from Linux; limited on iOS.
- **Constant / random RF jamming:** generate continuous wideband noise using SDR (USRP/HackRF) to measure effects on throughput and service availability.
- **Reactive jamming:** implement protocol-aware reactive jammer that only transmits when target frames are observed (lower power & stealthy). Use GNU Radio + SDR; measure detection and classification.
- **Selective channel/frame replay/injection:** use Scapy/tcpreplay for frame replay to provoke retransmission storms or confusion.

5. **Detection & mitigation tests**

- Deploy edge detection (TinyML or RSSI+packet loss heuristics) on Raspberry Pi / Android to classify jamming vs. congestion. Test channel-hopping / AP-level mitigation (auto-channel switch, client backoff).

6. **Data collection & triage**

- For each test record: RF spectrum trace, pcap, RSSI/time series, client logs, AP logs. Reproduce with controlled parameters (power, duty cycle, reactive thresholds). Use these artifacts for root-cause and to evaluate mitigations.

7. **SAST / firmware review**

- Where you control firmware (IoT devices, AP firmware), run static analysis to verify backoff/retry logic, management-frame protection (802.11w/MFP), and recovery code (graceful reconnection).

4. Quick Checklist

- Management-frame protection (802.11w) enabled and robust? Test deauth resilience.
- Can a reactive or selective jammer force repeated client reauths (battery drain for IoT)? Measure power impact.
- Does the AP/client implement channel-hopping or channel avoidance? Test migrating clients and channel re-assignment.
- Detection capability: can edge-devices distinguish congestion vs jamming (RSSI + packet loss + spectral features)?
- Are firmware/driver retries/backoff sane (to avoid infinite loops or amplification)? Review code.

5. References

1. Pirayesh, H., & Zeng, H. (2022). Jamming attacks and anti-jamming strategies in wireless networks: A comprehensive survey. *IEEE communications surveys & tutorials*, 24(2), 767-809.
2. Schepers, D., Ranganathan, A., & Vanhoef, M. (2022). On the robustness of Wi-Fi deauthentication countermeasures. In *Proceedings of the 15th ACM conference on security and privacy in wireless and mobile networks* (pp. 245-256).
3. Xu, W., Trappe, W., Zhang, Y., & Wood, T. (2005). The feasibility of launching and detecting jamming attacks in wireless networks. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing* (pp. 46-57).
4. Nguyen, D., Sahin, C., Shishkin, B., Kandasamy, N., & Dandekar, K. R. (2014, August). A real-time and protocol-aware reactive jamming framework built on software-defined radios. In *Proceedings of the 2014 ACM workshop on Software radio implementation forum* (pp. 15-22).
5. Hussain, A., Abughanam, N., Qadir, J., & Mohamed, A. (2022). Jamming detection in iot wireless networks: An edge-ai based approach. In *Proceedings of the 12th International Conference on the Internet of Things* (pp. 57-64).

Security Testing for Orbital Jamming Attacks

1. Overview

- In the scenario involving *Orbital Jamming attacks*, attackers intentionally create radio frequency (RF) interference or hostile emissions that disrupt satellite signals used for Global Navigation Satellite Systems (GNSS) and satellite communications (SATCOM). This interference can lead to outages or spoofing, resulting in timing errors that affect mobile devices, Internet of Things (IoT) gateways, and cloud-connected services.

- **Why it matters for cloud-mobile-IoT:** many IoT nodes, mobile apps and cloud systems depend on GNSS timing/position and satcom backhaul. Jamming or spoofing upstream or at LEO/MEO/HEO links can cause device location/timestamp corruption, loss of connectivity, or cascading service degradation.
- Testing are done in **controlled labs** with GNSS/SATCOM simulators, SDRs and shielded chambers (or via vendor test services) to emulate jammers/spoofers and measure resiliency. Commercial GNSS simulators and high-end SATCOM emulators are commonly used for realistic orbital/propagation scenarios.

---

2. Security Test Approaches & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
White-box	DAST	GNSS / SATCOM scenario simulation (orbit, propagation, interferer)	Rohde & Schwarz GNSS & Vector Signal Generators / Spirent GSS9000	Rohde & Schwarz GNSS / Spirent GSS9000	Both
Gray-box	DAST	Open-source GNSS signal generation for lab testing	GPS-SDR-SIM / GNSS-SDR	PS-SDR-SIM / GNSS-SDR	Both
Black-box	DAST	Controlled jamming / directed interference (lab, shielded)	USRP (Ettus) / HackRF One + GNU Radio	USRP (Ettus) / HackRF One / GNU Radio	Both
White-box	DAST	Anti-jamming / antenna nulling & CRPA testing	High-end vector signal generators + CRPA testbeds (R&S / Spirent)	Rohde-Schwarz / Spirent	Both
Gray-box	DAST	Spectrum monitoring & RFI detection	Spectrum analyzer (Keysight, Rigol), RTL-SDR, RF-Explorer	Keysight / Rigol / RTL-SDR	Both
White-box	SAST/DAST	Satellite terminal/modem conformance and link emulation	Skydel / GSG GNSS & SATCOM emulators	Skydel / Safran - Navigation & Timing	Both
White-box	DAST	On-orbit / spacecraft RF monitoring & telemetry analysis	OPS-SAT test experiments / satellite telemetry ingest	OPS-SAT / Satellite telemetry	Both
Gray-box	DAST	Receiver performance under jamming/spoofing	GPS/GNSS receiver test suites (R&S, Spirent) / GNSS-SDR	Rohde-Schwarz / GNSS-SDR	Both
Gray-box	DAST	End-to-end service disruption simulation (mobile/IoT â†' cloud)	Testbed orchestration: Docker, ns3, tc (traffic control), Zeek	Zeek / ns-3	Both
White-box	SAST	Regulatory compliance & RF shielding validation	Anechoic chamber / RF attenuators / licensed test range	Vendor labs (Keysight, ETS-Lindgren)	Both

---

3. Representative Test Scenarios

1. **GNSS jamming tolerance:** in chamber, increase broadband noise near L1/L5 and measure GNSS receiver time-to-first-fix, PNT degradation, and failover behavior for IoT devices and mobile phones. Record application impacts (e.g., time stamps, scheduled tasks).
2. **Directed narrowband jammer against SATCOM uplink:** simulate carrier nulling or narrowband interference on satellite uplink frequency (lab only) and measure SATCOM modem BER, link margin, and reconnection time. Use vendor emulators for precise link budgets.
3. **Spoofing + replay hybrid:** use GNSS simulator to create plausible false GNSS constellation (time/position ramp) to test mobile wallet / IoT geofencing and cloud time sync robustness. Validate RAIM/receiver anti-spoofing or application checks.
4. **CRPA nulling & anti-jam verification:** test multi-antenna anti-jam systems with injected interferer to validate beamforming/nulling performance and verify position/timing recovery.
5. **End-to-end cloud impact test:** while GNSS degraded, simulate how IoT fleet telemetry, mobile app features and cloud scheduling reliant on PNT behave—check logs, alert triggers, and failure modes. Use ns3 / Docker testbed to emulate large-scale effects.

4. References

1. Tedeschi, P., Sciancalepore, S., & Di Pietro, R. (2022). Satellite-based communications security: A survey of threats, solutions, and research challenges. *Computer Networks*, 216, 109246.
2. Olsson, G. K., Nilsson, S., Axell, E., Larsson, E. G., & Papadimitratos, P. (2023, April). Using mobile phones for participatory detection and localization of a gnss jammer. In *2023 IEEE/ION Position, Location and Navigation Symposium (PLANS)* (pp. 536-541). IEEE.
3. Otay, H., Humadi, K., & Kurt, G. K. (2023, November). Dark Side of HAPS Systems: Jamming Threats towards Satellites. In *2023 IEEE Future Networks World Forum (FNWF)* (pp. 1-6). IEEE.
4. Pirayesh, H., & Zeng, H. (2022). Jamming attacks and anti-jamming strategies in wireless networks: A comprehensive survey. *IEEE communications surveys & tutorials*, 24(2), 767-809.
5. Troglia Gamba, M., Polidori, B. D., Minetto, A., Dovis, F., Banfi, E., & Dominici, F. (2024). GNSS radio frequency interference monitoring from LEO satellites: An in-laboratory prototype. Sensors*, 24(2), 508.
6. Gilabert, R., Gutierrez, J., & Dill, E. (2024, September). GPS Multipath Emulation using Software Generated Signals. In *2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC)* (pp. 1-6). IEEE.

Security Testing for GPS Jamming Attacks

1. Overview

GPS jamming is interference that denies PNT (position, navigation, timing) and is detected using power/AGC/C/N0 monitoring, correlation/quality metrics, multi-antenna and multi-constellation methods, and ML models — testing requires RF hardware (SDR/USRP/HackRF), controlled test ranges (Faraday/isolated), and careful legal authorisation.

2. Security Test Approaches & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST (RF)	Broadband / narrowband RF jamming (PHY-level)	USRP B200/B210, HackRF	Ettus USRP, HackRF	Both (Android/iOS clients, IoT)
Black-box	DAST (RF)	Signal generator + GPS-SDR-SIM based waveform jamming	GPS-SDR-SIM + SDR transmit chain	GPS-SDR-SIM	Both
Gray-box	DAST / Detection	Reactive/triggered jamming & power sweep tests	GNU Radio flowgraphs, sdrplay/USRP scripts	GNU Radio	Both
Gray-box	DAST / Network	Measure device behaviour & time-sync degradation	Android GPSLogger, iOS location logs, RTK/PPP receivers	Android: GPSLogger	Android, iOS
Gray-box	DAST / ML detection	ML-based jamming detection (edge models)	TensorFlow Lite, scikit-learn (feature extraction)	TensorFlow Lite	Both (edge IoT, Android)
White-box	SAST / Config	Firmware & application review for fallback handling	Static code analysis (CodeQL), manual review	CodeQL	Both
Black-box	DAST / Forensics	Spectrum capture & logging	RTL-SDR, spectrum analyzer, SigDigger	RTL-SDR	Both
Gray-box	DAST / Emulation	Lab emulation & repeatable scenarios (drones, vehicles)	Spirent / Orolia (if available) or SDR-based simulators	Orolia	Both

3. Short Testing Setup — Practical Steps

1. **Legal & safety first**

- Obtain written authorization and local/regulatory clearance. Perform tests only in an RF-isolated enclosure (Faraday cage) or licensed test range. Jamming outside controlled environments is illegal and can disrupt critical services.

2. **Hardware & software inventory**
- SDR transmitter (USRP B200/B210 or HackRF), SDR RX (RTL-SDR or USRP), spectrum analyzer (if available).
- GNSS simulation software: **GPS-SDR-SIM** or vendor simulators (Spirent/Orolia) for controlled signal generation.
3. **Baseline & instrumentation**
- Deploy test targets: Android phones (multiple OS versions), iOS device(s), representative IoT GNSS modules (u-blox, MediaTek), and a reference high-quality GNSS receiver (RTK/PPP) to compare. Log: receiver NMEA, C/N0, AGC, number of satellites, fix type, PPS/timestamp stability, and application-level behaviour (e.g., navigation app route deviation).
4. **Controlled jamming experiments** (start low power / short durations)
- **Broadband noise jamming**: transmit wideband noise covering L1/L2 to observe loss of lock and C/N0 drop.
- **Narrowband sweep**: sweep transmitter power/frequency and duty cycle to map susceptibility and receiver thresholds.
- **Reactive jamming**: trigger interference only when device is tracking to simulate stealthy denial. Use GNU Radio to implement reactive flows.
5. **Measurements & detection signals**
- Monitor AGC, sudden jumps in received power, C/N0 reductions, satellite count changes, and GNSS receiver alarms. Collect SDR spectrum traces and NMEA logs for each run. These metrics are common for jamming detection.
6. **ML / feature-based detection prototypes**
- Extract features (power spectral density, AGC trends, C/N0 time series, Doppler anomalies) and train a lightweight classifier (scikit-learn / TensorFlow Lite) to detect jamming vs. benign interference. Recent work shows ML+multimodal approaches can be highly effective.
7. **Resilience & mitigation tests**
- Test multi-constellation (GPS+GLONASS+Galileo), multi-frequency receivers and anti-jamming hardware (CRPA, antenna nulling) and evaluate improvement. Validate fallback behaviours for apps (e.g., use inertial sensors, Wi-Fi/GNSS fusion).
8. **Reporting & safe teardown**
- For each test case record: test ID, equipment & power, duration, spectrum capture, NMEA logs, observed effects (loss of lock, time offset, position error), and suggested mitigations. Remove transmitter and verify environment returned to baseline.

---

4. Quick Checklist (priority tests)

- Can a low-power jammer cause loss of fix or position/time errors on consumer Android/iOS devices? (measure C/N0 and #SV).
- Threshold mapping: what TX power / duty cycle causes denial for each receiver? (do power sweep).
- Stealthy/reactive jamming: can short bursts timed to acquisition prevent reacquisition yet remain hard to detect? (use reactive SDR flows).
- Detection: does AGC/C/N0-based detection or ML classifier reliably flag jamming before service loss? (evaluate false positives).
- Mitigation effectiveness: quantify benefit from multi-constellation, multi-frequency, inertial/GNSS fusion or anti-jamming antennas.

---

5. References

1. Ghanbarzade, A., & Soleimani, H. (2025). GNSS/GPS spoofing and jamming identification using machine learning and deep learning. *arXiv preprint arXiv:2501.02352*.
2. RadoÅ¡, K., BrkiÄ‡, M., & BeguÅ¡iÄ‡, D. (2024). Recent advances on jamming and spoofing detection in GNSS. *Sensors*, 24(13), 4210.
3. Khan, S. Z., Mohsin, M., & Iqbal, W. (2021). On GPS spoofing of aerial platforms: a review of threats, challenges, methodologies, and future research directions. *PeerJ Computer Science*, 7, e507.
4. Liu, S., Cheng, X., Yang, H., Shu, Y., Weng, X., Guo, P., ... & Yang, Y. (2021, January). Stars can tell: A robust method to defend against gps spoofing using off-the-shelf chipset. In *Proceedings of The 30th USENIX Security Symposium (USENIX Security)*.
5. Abhishek, A. N., Balaji, A., Avinash, D., Harish, D., & Santhameena, S. (2025, March). Evaluating GNSS Spoofing and Jamming Attacks on UAV Navigation: Implementation and Impact. In *2025 IEEE 14th International Conference on Communication Systems and Network Technologies (CSNT)* (pp. 540-546). IEEE.
6. Ahmad, M., Farid, M. A., Ahmed, S., Saeed, K., Asharf, M., & Akhtar, U. (2019, January). Impact and detection of GPS spoofing and countermeasures against spoofing. In *2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)* (pp. 1-8). IEEE.

Security Testing for Bluesnarfing Attacks

1. Overview

The **Bluesnarfing attack** is a serious, legacy security vulnerability that exploits weaknesses in older Bluetooth implementations to **extract data** from a device without the owner permission or knowledge. Unlike Bluejacking (which only pushes unsolicited data), Bluesnarfing **pulls** sensitive data, making it a critical threat to confidentiality.

In the **Cloud-Mobile-IoT ecosystem**, Bluesnarfing targets older mobile phones, early smart devices, or IoT peripherals with unpatched Bluetooth stacks, aiming to steal address books, calendars, and authentication tokens before the data can reach the cloud.

---

2. Detailed Testing Setup and Procedures

The testing process focuses on confirming that the device refuses unauthorized access to its internal file structure and services.

Data Extraction Simulation (Black-box)

- **Procedure:** Use specialized Bluetooth tools like **Bluesnarfer** or **BlueDump** from a laptop in close proximity to the target device. These tools are designed to send specific, malformed Bluetooth requests (often targeting the **OBEX Push Profile** or **Service Discovery Protocol (SDP)**) that bypass the authentication layer.
- **Goal:** Attempt to **extract sensitive data** stored locally on the device, such as the phone address book (`telecom/pb.vcf`) or calendar entries (`telecom/cal.vcs`). A secure device must successfully **reject the connection** and prevent any unauthorized file system access, even when the device is discoverable.

Authentication Bypass Testing (Gray-box)

- **Procedure:** Utilize powerful packet manipulation frameworks like **Scapy** to craft and inject custom Bluetooth packets. This allows testers to target specific weaknesses, such as attempting to bypass the **pairing process** or spoofing the security mode (e.g., trying to downgrade a secure connection to an unsecure mode).
- **Goal:** Verify that the device Bluetooth stack strictly adheres to the protocol security specifications and **rejects any packets** that attempt to manipulate or bypass the required authentication and encryption keys.

Service Discovery and Information Leakage (Black-box)

- **Procedure:** Use standard **Bluetooth Scanners** (**hcitool**, **Kismet**) to scan the target device and list all available **Services Discovery Protocol (SDP)** records.
- **Goal:** Ensure the device is **not advertising sensitive services** that could provide a foothold for an attacker, such as an open Serial Port Profile (SPP) or File Transfer Protocol (FTP) profile without mandatory authentication/pairing. The device should only expose necessary, generic services, and not reveal internal application or operating system details.

Bluetooth Stack Configuration Review (White-box)

- **Procedure:** Conduct a **Manual Code Review** of the device **firmware** or operating system files that control the Bluetooth stack configuration.
  Goal: Ensure that the default security settings are maximized. Specifically, verify that:
  - **Pairing Mode:** Requires user confirmation for pairing (no Just Works pairing for critical services).
  - **Security Level:** Critical services (like OBEX) are configured to require **Authentication** and **Authorization**.
  - **Legacy Support:** If the device must support legacy Bluetooth, verify that all known Bluesnarfing vulnerabilities for that stack version have been patched or mitigated.

---

3. Security Testing Tools

Testing for Bluesnarfing resilience is crucial for backward compatibility and involves simulating the attack environment to verify that the device Bluetooth stack properly enforces authentication and authorization protocols.

Test Approach	Analysis Type	Approach Name	Testing Tool	Hyperlink for Tool	Platform
Black-box	DAST	Data Extraction Simulation	Bluesnarfer, BlueDump, hcitool (part of BlueZ)	Bluesnarfer (GitHub)	Both
Gray-box	DAST	Authentication Bypass Testing	Custom Scripts (Python/Scapy) targeting Bluetooth SDP records	Scapy	Both
White-box	SAST	Bluetooth Stack Configuration Review	Manual Code Review (examining Bluetooth stack configuration)	N/A (Manual process)	Android, iOS, IoT Firmware
Black-box	DAST	Information Leakage (SDP Services)	Bluetooth Scanners (e.g., hcitool, kismet)	hcitool (BlueZ)	Both

---

4. References

1. Blancaflor, E., Purificacion, P. M. G., Atienza, R. B., Yao, J. J. M., & Alvarez, D. A. C. (2023). Exploring the depths of Bluetooth attacks: A critical analysis of Bluetooth exploitation and awareness of users. In *2023 6th International Conference on Computing and Big Data (ICCBD)* (pp. 52-59). IEEE.
2. Rohith, R., Moharir, M., & Shobha, G. (2018). SCAPY-A powerful interactive packet manipulation program. In *2018 international conference on networking, embedded and wireless systems (ICNEWS)* (pp. 1-5). IEEE.
3. Blancaflor, E., Billo, H. K., Dignadice, J. M., Domondon, P., Linco, M. R., & Valero, C. (2024). Bluetooth Simulated Reconnaissance Attack Through the Use of HCITool: A Case Study. In International Conference on Cloud Computing and Computer Networks (pp. 133-143). Cham: Springer Nature Switzerland.

Security Testing for Bluejacking Attacks

The **Bluejacking attack** is a non-exploitative security vulnerability that uses the Bluetooth protocol to send unsolicited messages (text or images, often spam or advertisements) to nearby Bluetooth-enabled devices. It typically operates within a limited range (around 10â€“100 meters).

In the **Cloud-Mobile-IoT ecosystem**, Bluejacking is less of a data theft threat and more of an **annoyance** and a **Denial of Service (DoS) vulnerability** that can drain battery life or be used for social engineering (phishing). Security testing focuses on verifying the default settings and the ability of the device's operating system or application to properly handle incoming Bluetooth messages from unknown sources.

---

1. Detailed Testing Setup and Procedures

The testing process focuses on client configuration, battery resilience, and social engineering risk.

1.1. Message Sending Simulation (Black-box)

- **Procedure:** Use a Bluetooth-enabled laptop running **Bluez** (a Linux Bluetooth stack tool) or similar specialized mobile apps to scan for discoverable devices and send an unsolicited object/message (e.g., a vCard, note, or image) to a target mobile phone or IoT peripheral.
- **Goal:** Verify that devices with default settings are **not set to discoverable mode** permanently. If a device receives the message, ensure the operating system (OS) forces a **prompt for user approval** before any data transfer or notification occurs. This confirms the device is resistant to unauthorized push attacks.

1.2. Device Discoverability and Information Leakage (Black-box)

- **Procedure:** Use Bluetooth scanning tools like **hcitool** or **Kismet** to passively monitor the testing environment for devices advertising their presence via Bluetooth Low Energy (BLE) beacons or Classic Bluetooth.
  Goal:
  - Confirm that the device's advertised name (**Bluetooth Device Name**) does not contain personally identifiable information (PII).
  - Verify that any associated **IoT application** running on the mobile phone does not force the phone's Bluetooth to remain 'Always Discoverable' or 'Always On' when the application is in the background.

1.3. Denial of Service (DoS) and Battery Drain Testing (Gray-box)

- **Procedure:** Use **Custom Scripts** built with a tool like **Scapy** to rapidly flood the target device with continuous Bluetooth connection requests (pairing attempts or Service Discovery Protocol queries).
- **Goal:** Determine if the constant handling of unsolicited requests causes a significant **battery drain** or a noticeable **performance slowdown** (DoS) on the target mobile or IoT device. A resilient device should throttle connection attempts from the same source after a few failures.

1.4. Bluetooth Permission Review (White-box)

- **Procedure:** Conduct a **Manual Code Review** of the mobile application's manifest files and source code.
- **Goal:** Ensure that the application only requests the minimum necessary Bluetooth permissions (`BLUETOOTH`, `BLUETOOTH_ADMIN`, etc.) and does not attempt to enable discoverability or accept connections without the explicit, user-initiated intent. This prevents the application from inadvertently making the user vulnerable to Bluejacking.

2. Security Testing Approach & Tools

The testing setup involves simulating a Bluejacking attacker and monitoring how the client device (mobile phone or IoT peripheral) processes the unsolicited data and whether it exposes sensitive information through the Bluetooth stack.

Test Approach	Analysis Type	Approach Name	Testing Tool	Hyperlink for Tool	Platform
Black-box	DAST	Message Sending Simulation	Bluejacking Tools (e.g., Bluez, BlueDump), Mobile Apps (Android/iOS)	Bluez	Both

Black-box	DAST	Device Discoverability/Exposure	Bluetooth Scanners (e.g., hcitool, kismet/BTLE)	hcitool (BlueZ)	Both
Gray-box	DAST	Denial of Service (DoS) Testing	Custom Scripts (Python/Scapy) for flooding the device with connection attempts	Scapy	Both
White-box	SAST	Bluetooth Permission Review	Manual Code Review	N/A (Manual process)	Android, iOS

3. References

1. Blancaflor, E., Purificacion, P. M. G., Atienza, R. B., Yao, J. J. M., & Alvarez, D. A. C. (2023). Exploring the depths of Bluetooth attacks: A critical analysis of Bluetooth exploitation and awareness of users. In *2023 6th International Conference on Computing and Big Data (ICCBD)* (pp. 52-59). IEEE.
2. Rohith, R., Moharir, M., & Shobha, G. (2018). SCAPY-A powerful interactive packet manipulation program. In *2018 international conference on networking, embedded and wireless systems (ICNEWS)* (pp. 1-5). IEEE.
3. Blancaflor, E., Billo, H. K., Dignadice, J. M., Domondon, P., Linco, M. R., & Valero, C. (2024). Bluetooth Simulated Reconnaissance Attack Through the Use of HCITool: A Case Study. In International Conference on Cloud Computing and Computer Networks (pp. 133-143). Cham: Springer Nature Switzerland.

Security Testing for Wi-Fi SSID-tracking Attacks

1. Overview

A VM Migration Attack exploits vulnerabilities during the transfer of virtual machines between hosts, while Wi-Fi SSID Tracking Attacks manipulate network identifiers to deceive devices — both demand rigorous security testing to prevent data breaches and service disruption.

Why Security Testing Is Essential:

- Proactive Defense: Detects vulnerabilities before attackers do;
- Compliance Assurance: Meets regulatory standards for data protection;
- Trust and Reliability: Ensures users and clients can rely on secure infrastructure.

1. Security Test Approaches & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST	Passive sniffing (probe/beacon collection)	Kismet, Wireshark, tcpdump	Kismet, Wireshark	Both
Black-box	DAST / Data	Wardriving / geo-correlation (DB)	WiGLE (app & DB), custom scripts	WiGLE	Both
Black-box / Gray-box	DAST	Active elicitation / probe injection	Scapy, Bettercap, aireplay-ng	Scapy, Bettercap, Aircrack-ng	Both (Linux); Android (rooted) for active tests
Gray-box	DAST / ML	Probe-request fingerprinting / clustering	Python (scapy, pandas), scikit-learn, timestamps	scapy	Both
White-box	SAST	Firmware/OS review (randomization logic)	CodeQL, SonarQube, manual code review	CodeQL	Both
Gray-box	DAST	Simulated multi-collector wardrive	Raspberry Pi + multiple Wi-Fi adapters, Kismet	Kismet	Both

Black-box	DAST / Privacy	Probe content analysis (SSIDs reveal PII)	pcap â†’ CSV parsers, grep, regex	custom scripts (Python)	Both
Gray-box	DAST / Detection	Test OS privacy settings (MAC randomization)	Android adb, iOS config profiles, device logs	[adb]()	Android, iOS

---

2. Testing Setup — Step-by-step (practical)

Legal / ethics first: always get written permission for any active tests; for passive collection follow local privacy & data-protection rules and anonymise results. The literature treats probe requests as sensitive data because they often expose PII.

1. **Lab & hardware**
- Linux laptop or Raspberry Pi with 1—3 external Wi-Fi adapters that support monitor mode (Atheros/realtek with good driver support).
- Optional GPS for wardriving, and a small mobile (Android/iOS) testbed with devices of different OS versions.
- Install Kismet + Wireshark + scapy + python data libs.

2. **Baseline passive capture**
- Put adapters in monitor mode and capture for a representative period (e.g., 30 min) in the target environment. Save pcaps and export CSV of: timestamp, source MAC, SSID (if present), RSSI, channel, and any IEs. Kismet can log GPS + time for wardriving.

3. **Probe-content analysis**
- Parse pcaps (scapy or tshark) and scan SSID fields for PII patterns (e.g., long numeric strings, email patterns, names, home router defaults). The 2022 field study found SSIDs leaking passwords and personal data in a notable fraction of probe requests.

4. **MAC randomization testing**
- Measure randomization behaviour: record sequences of MACs and see when devices use global vs randomized MAC; test with MAC randomization toggled on/off in settings (where available). Prior studies show many devices still leak persistent identifiers or fail to randomize reliably.

5. **Fingerprinting & re-linking**
- Extract frame features (IE fields, supported rates, sequence timing, probe burst timing, vendor IEs). Cluster traces using time-based and feature-based clustering to attempt to link multiple randomized MACs to one device. Use scikit-learn clustering (DBSCAN / hierarchical). Deep-learning approaches can also achieve high accuracy on 802.11ac IEs.

6. **Active elicitation (ONLY with permission)**
- Use Scapy/Bettercap to send directed probe requests or elicit responses; measure whether devices respond with PNL contents or reveal hidden SSIDs. Active probing can increase identification but must be used only in lab or permitted environments.

7. **Wardriving / historical correlation**
- (Optional) run a controlled wardrive (or simulate multiple collectors) and store geotagged SSID sightings. Query local copies of WiGLE or your dataset to attempt location-based linking and POI inference. Public databases enable powerful correlation attacks.

8. **Firmware / OS code review**
- Where you control firmware or app code (IoT APs, vendor firmware, mobile app), review logic that constructs probe requests and PNL sharing. Check whether SSIDs are included inadvertently (e.g., when users paste SSIDs) and review randomization code.

9. **Mitigation validation**
- Toggle and test mitigations: MAC randomization frequency, disable probe-requests in background scans, enable/verify 802.11w/management frame protections, test client behaviour when SSID privacy features are on/off. Re-run clustering experiments to quantify reduction in linkability.

---

3. Artifacts to Collect (for each run)

- pcap (timestamped), CSV of observed frames (MAC, SSID, RSSI, IEs), GPS trace (wording: lat/lon/time), device setting snapshots (MAC randomization enabled?), device logs (if available), and scripts used for parsing/analysis.

---

4. Quick Checklist (priority tests)

- **Probe content leak:** any SSIDs containing PII (emails, apparent passwords, names)?
- **MAC randomization effectiveness:** how often does it rotate and when does it fall back to global MAC?
- **Re-linkability:** can you cluster randomized MACs by timing/IEs to get stable device IDs?
- **Historical correlation:** can wardriving / WiGLE data deanonymize traces into locations/POIs?
- **IoT exposures:** do IoT devices advertise default/unique SSIDs that identify device type/location? (check SSID formatting).

---

5. Mitigations to Test / Validate

- **Frequent MAC re-randomization** (session or scan-level) and ensure random MACs are not reused across contexts.
- **Omit probe SSID fields** unless user explicitly selects a network (OS level change).
- **Avoid PII in SSIDs** (UI/UX sanitization and user education).

- **Management-frame protection & reduced probe emission** (802.11w and OS scan throttling).

---

6. References

1. Ansohn McDougall, J., Burkert, C., Demmler, D., Schwarz, M., Hubbe, V., & Federrath, H. (2022). Probing for passwordsâ€“privacy implications of ssids in probe requests. In International Conference on Applied Cryptography and Network Security (pp. 376-395). Cham: Springer International Publishing.
2. Martin, J., Mayberry, T., Donahue, C., Foppe, L., Brown, L., Riggins, C., ... & Brown, D. (2017). A study of MAC address randomization in mobile devices and when it fails. arXiv preprint arXiv:1703.02874.
3. Gu, X., Wu, W., Gu, X., Ling, Z., Yang, M., & Song, A. (2020). Probe request based device identification attack and defense. Sensors, 20(16), 4620.
4. Rye, E., & Levin, D. (2024). Surveilling the masses with wi-fi-based positioning systems. In 2024 IEEE Symposium on Security and Privacy (SP) (pp. 2831-2846). IEEE.
5. Thomas, A. M., Kumaran, G. A., Ramaguru, R., Harish, R., & Praveen, K. (2021). Evaluation of wireless access point security and best practices for mitigation. In 2021 5th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT) (pp. 422-427). IEEE.

Security Testing Setup for Byzantine Attacks

1. Overview

Goal: assess an ecosystem resilience when some participants (cloud agents, mobile clients, IoT nodes) behave arbitrarily or maliciously (send wrong/contradictory data, refuse to follow protocol, equivocate, collude to subvert consensus or analytics). Tests should cover: consensus manipulation, data-poisoning/Byzantine behaviour in federated learning, Sybil/eclipse & partition attacks, malicious firmware/node behavior, and detection/mitigation controls.

---

2. Lab & Safety Pre-requisites

- **Isolated testbed**: separate cloud project(s), VLANs, device lab (real + emulated IoT and mobile devices), and a node orchestration environment (Docker/Kubernetes).
- **Node replicas**: provision multiple node instances to simulate honest/malicious ratios (e.g., N nodes, f malicious). Use containerized images for reproducibility.
- **Instrumentation & logging**: central telemetry (ELK/Splunk), PCAP collectors, and per-node logs. Enable debug/tracing in consensus stacks.
- **Rollback & snapshots**: VM/container snapshots, firmware backups for IoT devices.
- **Rules of engagement**: written authorization, time windows, and radio/regulatory compliance for any wireless experiments.
- **Safety**: do not run disruptive network partitioning tests against production or third-party networks.

---

3. High-level Testing Categories

1. **Malicious-node simulation** — create nodes that send conflicting messages, incorrect state, or random/stale data.
2. **Consensus-layer attacks** — equivocation, vote withholding, message forging, view-change manipulation, leader corruption.
3. **Sybil & eclipse** — spin-up many identities or isolate nodes to bias their view of network state.
4. **Partition & network-level attacks** — simulate partitions, delays, and reorderings to test safety/liveness under asynchrony.
5. **Data-poisoning / Byzantine ML** — in federated learning or distributed analytics, inject malicious gradients or model updates.
6. **Firmware/node compromise** — emulate IoT nodes programmed to misbehave (wrong telemetry, replay, clock skew).
7. **Detection & tolerance validation** — verify reputation systems, BFT thresholds, anomaly detectors, and recovery procedures.

---

4. Practical Testing Workflow

- **Phase A — Design & provisioning**: decide N and maximum f (Byzantine nodes) for test scenarios; prepare honest and malicious node images and scripts; deploy using Docker / Kubernetes or VM pool.
- **Phase B — Baseline & functional tests**: verify consensus correctness with all honest nodes; collect baseline telemetry.
- **Phase C — Malicious behavior injection**: run repeatable scenarios: equivocation (duplicate/conflicting signed messages), inconsistent state responses, delayed votes, or arbitrary payloads. Measure safety (no divergent committed state) and liveness (progress).
- **Phase D — Network faulting & chaos experiments**: use Jepsen-like partitioning (network cut, delay, reorder) and Chaos Toolkit to observe protocol behavior under faults.
- **Phase E — Sybil/identity attacks**: spawn many identities or simulate restricted peer views (eclipse) to test resilience and peer-selection defenses.
- **Phase F — Federated learning Byzantine tests**: inject malicious updates (label-flip, scaled gradients, random updates) and measure model degradation & robustness of aggregation rules (median, Krum, trimmed mean).
- **Phase G — Detection & remediation checks**: validate reputation scores, view-change counters, quarantining, and fallback recovery procedures; test reconfiguration and re-sync flows.
- **Phase H — Reporting & hardening**: produce reproducible testcases, suggested protocol parameter changes, and detection rule tuning.

Each of the above test phases should be automated and repeatable; log seeds and random seeds for bloom and randomness reproducibility.

---

5. Short Testing Playbook

1. **Set test parameters** — choose N (total nodes) and f (max Byzantine nodes) per scenario; document expected safety/liveness bounds.
2. **Deploy baseline network** — deploy N honest nodes (use BFT-SMaRt or Tendermint) and verify correct commits under normal operations.
3. **Inject Byzantine behavior** — replace f nodes with malicious behavior modules (equivocation, delayed replies, conflicting proposals). Log the system outputs and measure if forks/divergence occur.
4. **Chaos experiments** — use Jepsen or Chaos Toolkit to partition nodes, reorder messages, or drop messages; observe protocol reactions (timeouts, view-changes).
5. **Sybil & eclipse testing** — spin many fake peers (Docker) and attempt to isolate target nodes (limit its peer set) to bias its view. Measure impact on consensus & state.
6. **Byzantine federated learning** — use TensorFlow Federated to emulate clients; instruct a subset to send poisoned gradients (scaling/flip). Test aggregation defenses (Krum, median, trimmed mean) and quantify model degradation.
7. **Protocol fuzzing** — fuzz message formats and fields (Scapy, AFL harnesses) to find equivocation/ parsing pitfalls.
8. **Detection & recovery tests** — verify reputation systems, automatic exclusion/quarantine, reconfiguration, and data reconciliation after faults.
9. **Reporting** — produce reproducible testcases (container images, test scripts, random seeds, logs) and recommended hardening (protocol parameter tuning, signature/non-repudiation, peer diversity).

---

6. Security Testing Approaches & tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Black-box / Gray-box	DAST	Malicious node simulation / equivocation	Custom node scripts (Docker) + BFT-SMaRt harness	BFT-SMaRt harness	Cloud / IoT (containerized nodes)
Gray-box	DAST / Functional	Practical Byzantine / consensus testing	Tendermint / Cosmos SDK (testnets)	Tendermint	Cloud (consensus layer)
Gray-box / White-box	SAST / DAST	Consensus load & benchmark testing	Hyperledger Caliper (benchmarks)	Hyperledger Caliper	Cloud / Blockchain stacks
Black-box	DAST / Network	Network partitioning / chaos experiments	Jepsen (or Chaos Toolkit)	Jepsen	Cloud / distributed nodes
Gray-box	DAST	Network emulation (latency, reorder, loss)	Mininet / ns-3	Mininet / ns-3	IoT / mobile network scenarios
Gray-box / White-box	DAST / Fuzzing	Message / protocol fuzzing	Scapy / AFL for message fuzz harness	Scapy / AFL	Cloud / IoT protocols
Gray-box	DAST / ML	Byzantine/federated learning attacks & defense testing	TensorFlow Federated / PySyft (federated frameworks)	TensorFlow Federated / PySyft	Mobile / cloud for federated learning
White-box / Gray-box	SAST / Behavioral	Replica instrumentation & trace analysis	PROMETHEUS / ELK + distributed tracing	PROMETHEUS / ELK	Cloud & IoT telemetry
Black-box	DAST / Recon	Peer discovery & Sybil stress tests	Custom scripts + Docker swarm/k8s to spawn identities	— (custom, repo links in your project)	Cloud / mobile / IoT
Gray-box / White-box	SAST / Formal	Model checking & formal verification of consensus	TLA+ / PlusCal	TLA+	Protocol design / cloud
Gray-box / DAST	Detection validation	Anomaly & reputation simulation	Scikit-learn / PyTorch	Scikit-learn / PyTorch	Cloud analytics / detection

---

6. References

1. Li, Z., & Xu, Y. (2023). Byzantine fault-tolerant consensus algorithms: A survey. *Electronics*, 12(18), 3801.

2. Bouhata, D., Moumen, H., Mazari, J. A., & Bounceur, A. (2024). Byzantine fault tolerance in distributed machine learning: a survey. *Journal of Experimental & Theoretical Artificial Intelligence*, 1-59.
3. Marcozzi, M., Gemikonakli, O., Gemikonakli, E., Ever, E., & Mostarda, L. (2023). Availability evaluation of IoT systems with Byzantine fault-tolerance for mission-critical applications. Internet of Things, 23, 100889.
4. Lamport, L., Shostak, R., & Pease, M. (1982). The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3), 382-401.
5. Song, A., Wang, J., Yu, W., Dai, Y., & Zhu, H. (2019, December). Fast, dynamic and robust byzantine fault tolerance protocol for consortium blockchain. In *2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)* (pp. 419-426). IEEE.
6. Ta, N., Shi, E., & Xiao, L. (2020). Optimal consensus with dual abnormality mode for cellular IoT. *Sensors*, 20(8), 2378.
7. Ji, J. C., Lam, C. T., & Ng, B. (2025, January). A survey on consensus algorithms for distributed wireless networks. In *The International Conference Optoelectronic Information and Optical Engineering (OIOE2024)* (Vol. 13513, pp. 294-303). SPIE.
8. Kerrakchou, I., Chadli, S., Kharbach, A., Saber, M. (2021). Simulation and Analysis of Jamming Attack in IoT Networks. In: Motahhir, S., Bossoufi, B. (eds) Digital Technologies and Applications. ICDTA 2021. *Lecture Notes in Networks and Systems*, vol 211. Springer, Cham.

Security Testing for Malicious Insider Attacks

1. Overview

Insider threats (malicious or negligent) pose major risks in cloud/mobile/IoT environments because insiders often already have valid access, and their attacks may span devices (mobile, IoT), apps, network, and cloud services. Detection is hard because activities can look legitimate. Studies highlight the increased complexity when IoT devices and mobile endpoints are involved.

---

2. High-level Testing Workflow / Setup

1. **Define scope & roles**: Identify devices (IoT sensors, gateways, mobile clients), mobile apps (Android/iOS), cloud services/accounts, user roles (employees, contractors), access levels, data flows.
2. **Baseline behaviour & logs**: Capture normal user/device behaviour: login patterns, file accesses, device onboarding, firmware updates, cloud API calls, mobile app telemetry.
3. **Access control and role review**: Examine permissions, user-roles, least-privilege compliance, mobile/IoT device enrolment and provisioning processes.
4. **Static analysis (SAST)**: Review code/configs for mobile apps & IoT firmware, check for excessive privileges, hard-coded credentials, insecure APIs, backdoors.
5. **Dynamic testing (DAST/eb)**: Simulate insider activity: elevated access, data exfiltration via mobile or IoT, lateral movement from IoT to cloud, unauthorized firmware changes. Monitor detection.
6. **Network & endpoint monitoring tests**: Use network sniffing and endpoint logging to see if unusual insider-type behaviours are captured (large file transfers, unusual login times, mobile device abnormal use).
7. **Cloud component tests**: Test compromised cloud credentials or insider misuse of APIs (mobile backend, IoT device management), see how telemetry/alerting responds.
8. **IoT / mobile device tests**: Simulate an insider leveraging mobile or IoT devices (for example, a mobile app pre-installed with extra privileges, an IoT device compromised by a trusted insider).
9. **Reporting and remediation**: Identify control gaps (policy, logging, detection, alerting, privilege management), recommend improvements (user & entity behavior analytics, fine-grained role separation, mobility/IoT device management).

---

3. Security Testing Aproach & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
White-box	SAST	Code review / Privilege & role permissions review	SonarQube / CodeQL	SonarQube / CodeQL	Both (mobile + cloud backend)
Gray-box	DAST	Mobile app instrumentation & runtime behaviour monitoring	Frida	Frida	Android, iOS
Gray-box	DAST	Endpoint monitoring & user activity simulation	OSQuery		Both (devices & gateways)
Black-box	DAST	Network packet sniffing & unusual transfer detection	Wireshark / Zeek	Wireshark / Zeek	Both

Gray-box	SAST	Cloud function & API scanning & review	Snyk / Semgrep	Snyk / Semgrep	Cloud + Backend
Gray-box	DAST	User & Entity Behaviour Analytics (UEBA) simulation	Splunk UBA / Elastic UEBA	Splunk UBA" / Elastic UEBA	Both
White-box	SAST	IoT firmware & device configuration review	Binwalk / Firmadyne	Binwalk / Firmadyne	IoT
Black-box	DAST	Simulated insider data exfiltration via mobile/IoT device	Netcat / Curl / Custom script	Netcat / Curl	Both

---

4. Practical Testbed / Setup Checklist

- **Identity & access logs**: collect login/logoff events, privilege escalations, API usage, device enrolment/dis-enrolment, file access logs.
- **Mobile device lab**: invest in Android and iOS test devices; install endpoint monitoring agents; simulate user roles (regular user vs privileged).
- **IoT device lab**: capture firmware images, device logs, connectivity patterns to cloud; simulate insider modifying device config or firmware.
- **Cloud backend**: enable full audit logs (API calls, resource provisioning, data movement, identity changes). Ensure logs are forwarded to SIEM/UEBA system.
- **Behavioral analytics setup**: configure UEBA engine to learn baseline for users/devices; feed logs from mobile/IoT/cloud; simulate insider scenarios (late-night access, large downloads, device provisioning and deprovisioning) to test detection.
- **Network monitoring**: mirror mobile/IoT network traffic to sniffers (Zeek, Wireshark) for unusual patterns (large outbound transfers, odd protocols).

  Simulated insider attacks:

  A user with privileged access exports data via mobile device or IoT gateway to external destination.

- A contractor installs malicious firmware or config change on IoT device that communicates with cloud.
- A mobile app acting as a legitimate client is used to change backend settings (via cloud API) that facilitate data siphoning.
- **Policy & control validation**: test least-privilege enforcement, device enrolment/un-enrolment workflows, cloud change-management auditing, mobile & IoT device hardening.
- **Reporting**: summary of gaps (logs missing, user behaviour not baseline-profiled, mobile/IoT devices not monitored, cloud API changes not audited). Recommend remediation: stronger role separation, device management, behavioural monitoring, cloud audit pipelines.

---

5. References

1. Kim, A., Kim, H., & Choi, I. (2020). Kim, A., Oh, J., Ryu, J., & Lee, K. (2020). A review of insider threat detection approaches with IoT perspective. *IEEE Access*, 8, 78847-78867.
2. Ali, A., Husain, M., & Hans, P. (2025). Real-time detection of insider threats using behavioral analytics and deep evidential clustering. *arXiv preprint arXiv:2505.15383*.
3. Callegati, F., Giallorenzo, S., Melis, A., & Prandini, M. (2018). Cloud-of-Things meets Mobility-as-a-Service: An insider threat perspective. *Computers & Security*, 74, 277-295.
4. Duncan, A. J., Creese, S., & Goldsmith, M. (2012, June). Insider attacks in cloud computing. In *2012 IEEE 11th international conference on trust, security and privacy in computing and communications* (pp. 857-862). IEEE.
5. Khan, A. Y., Latif, R., Latif, S., Tahir, S., Batool, G., & Saba, T. (2019). Malicious insider attack detection in IoTs using data analytics. IEEE Access, 8, 11743-11753.

security Testing Setup for Sniffing Attacks

1. Overview

Sniffing attacks capture network traffic (cleartext credentials, tokens, telemetry) between IoT devices, mobile clients and cloud services — test to find misconfigured links, weak crypto, or exposed telemetry.

---

2. Quick Test Workflow

1. **Scope & inventory** — list device types, network paths (device→gateway, gateway→cloud, mobile→cloud), protocols (HTTP, MQTT, CoAP, plain TCP/UDP).
2. **Baseline capture** — passively capture normal traffic at gateway & cloud ingress (pcap) to learn expected flows.
3. **Active sniff tests (lab)** — run controlled MITM/sniffing (proxy, ARP/ND spoofing, rogue AP) in isolated lab to see if secrets leak.
4. **Protocol checks** — verify TLS on all links, certificate validation, MQTT over TLS, MQTT client auth, and avoid plaintext protocols.
5. **Detection validation** — ensure IDS/Zeek/ELK detect unauthorized sniffing patterns (ARP spikes, new DHCP leases, TLS strip attempts).

6. **Remediate & retest** — enforce encryption, mutual auth, certificate pinning, and harden network segmentation; repeat captures.

---

3. Security Test Approach & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST	Passive packet capture	Wireshark / tshark	Wireshark	Both
Black-box	DAST	Network sniff + MITM (ARP/ND)	Bettercap / Ettercap	Bettercap / Ettercap	Both
Gray-box	DAST	HTTP/HTTPS proxy & request manipulation	mitmproxy / Burp Suite	mitmproxy / Burp Suite	Both
Gray-box	DAST	Wireless sniffing & rogue AP	Kismet / hostapd (rogue AP)	Kismet	Both (Wi-Fi)
Gray-box	DAST	BLE sniffing (mobile/IoT)	Ubertooth / nRF Sniffer	Ubertooth	Both (BLE)
White-box	SAST	Code/config review for insecure transports	Semgrep / CodeQL	Semgrep / CodeQL	Cloud & mobile
Gray-box	DAST	Network monitoring / detection	Zeek / Suricata + ELK	Zeek / Suricata	Both (network)
Black-box	DAST	Traffic replay / fuzzing	tcpreplay / scapy	tcpreplay / scapy	Both

---

4. Minimal Testbed & Checklist

- Isolated lab VLAN or physical air-gap.
- Capture points: near device (Wi-Fi/BLE sniffer), at gateway uplink, at cloud ingress.
- Test devices: representative IoT nodes, Android phone (with/without root), staging cloud service.
- Logging: PCAPs, Zeek logs forwarded to ELK/Splunk.
- Safety: never sniff on public/production networks without written permission.

---

5. References

1. Sicari, S., Rizzardi, A., Grieco, L. A., & Coen-Porisini, A. (2015). Security, privacy and trust in Internet of Things: The road ahead. *Computer Networks*, 76, 146-164.
2. Miorandi, D., Sicari, S., De Pellegrini, F., & Chlamtac, I. (2012). Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7), 1497-1516.
3. Wu, T., Breitinger, F., & Niemann, S. (2021). IoT network traffic analysis: Opportunities and challenges for forensic investigators?. *Forensic Science International: Digital Investigation*, 38, 301123.
4. Almutairi, M., & Sheldon, F. T. (2025). IoT-Cloud Integration Security: A Survey of Challenges, Solutions, and Directions. *Electronics*, 14(7), 1394.

Security Testing for Man-in-the-Middle Attacks

1. Overview

Man-in-the-Middle (MiTM) attacks allow an adversary to intercept, modify or inject communications between endpoints. In a combined cloud-mobile-IoT environment, the breadth of endpoints (IoT sensors, mobile clients, gateways, cloud APIs) increases the attack surface, especially when devices use weak protocols, unverified certificates, or are on untrusted networks. For example, many IoT clouds accept weak TLS versions, making MiTM easier.

---

2. High-level Testing Workflow / Setup

1. **Scope & threat modelling**

- Map all communication channels: IoT devices → gateways → cloud, mobile apps ↔ cloud, mobile apps ↔ IoT/gateway.
- Identify susceptible protocols/network segments: e.g., WiFi, MQTT, HTTP/HTTPS, Bluetooth, cellular.
- Identify trust boundaries and certificate/identity management across devices, mobile apps and cloud.

- Define possible MiTM vectors: rogue access points, ARP/DNS spoofing on IoT/edge networks, Proxy apps on mobile, compromised gateway, unsecured mobile hotspot, weak-TLS IoT cloud endpoint.

2. **Baseline behaviour capture**

- Record normal traffic flows: device registration, telemetry sends, mobile ↔ cloud API calls, IoT ↔ gateway communications.
- Check certificate validation behaviour, handshake protocols, TLS versions, whether devices accept self-signed certs or skip validation.

3. **Static analysis (SAST) & configuration review**

- Examine firmware/mobile app/cloud backend source/config: verify certificate pinning, TLS protocol enforcement, secure defaults, verification of server identity.
- Check IoT device firmware for insecure fallback (plaintext, weak encryption), check mobile app network libraries, cloud API endpoints for insecure defaults.

4. **Dynamic testing (DAST) / MiTM simulation**

- Set up a rogue network (e.g., WiFi access point) to simulate mobile MiTM: mobile connects, attacker intercepts traffic, tries SSL stripping, DNS spoofing, proxying mobile traffic (e.g., mitmproxy).
- For IoT/gateway: use ARP spoofing, DNS redirect, downgrade TLS, intercept MQTT or CoAP traffic between device and cloud/gateway. See if device accepts compromised cert or unverified endpoint.
- On cloud side: intercept mobile/gateway requests, attempt injection/modification of telemetry or commands, test if backend validates identity of source.

5. **Network & telemetry monitoring tests**

- Capture network traffic (Wireshark, Zeek) from devices/gateway/mobile while under MiTM to see if anomalies are logged.
- Use monitoring/alerting to check for certificate mismatches, repeated TLS renegotiation, unusual endpoints, or decrypted traffic being forwarded.

6. **Mobile & IoT integration tests**

- Mobile: Install a proxy certificate or set up a custom CA on device to simulate MiTM; test mobile app response.
- IoT: Insert a test rogue gateway or rogue device that captures/intercepts device â†” cloud communications; test deviceâ€™s behavior when gateway is malicious.

7. **Reporting & remediation**

- Identify weak points: unverified certificates, weak TLS versions, lack of certificate pinning on mobile, insecure IoT protocol, network segments lacking encryption or authentication, lack of detection/alerts for MiTM.
- Recommend controls: enforce TLS 1.2/1.3, certificate pinning/mobile secure libraries, mutual authentication for IoT devices and gateways, network segmentation, monitoring/UEBA for unusual flows, mobile/hotspot policy enforcement.
- Validate by retesting after mitigation.

---

3. Security Test Approach & tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST	Network packet sniffing & MiTM traffic intercept	Wireshark / Zeek	Wireshark / Zeek	Both
Black-box	DAST	Proxying mobile traffic / SSL-strip simulation	mitmproxy	Proxying	Android, iOS
Gray-box	SAST	Mobile app / IoT firmware review for certificate validation	Ghidra / Binwalk (firmware) / Static code analysis	Ghidra / Binwalk	Android, IoT
Gray-box	DAST	Rogue access point / ARP-spoof/ DNS-spoof tests on IoT/gateway network	ettercap / ARP-poison scripts	ettercap	IoT/gateway network
Gray-box	DAST	Mobile app instrumentation for certificate pinning & trust issues	Frida	Frida	Android, iOS
White-box	SAST	Cloud backend API review for TLS enforcement / mutual auth	Semgrep / CodeQL	Semgrep / CodeQL	Cloud
Gray-box	DAST	Monitoring & anomaly detection of MiTM behaviour in IoT/mobile network flows	Elastic Stack (beats + SIEM) / Splunk	Elastic Stack / Splunk	Both

White-box	SAST	IoT device firmware review of insecure network protocol usage	Binwalk + radare2	radare2	IoT
Black-box	DAST	Mobile hotspot MiTM simulation (rogue AP) for mobileâ€“cloud traffic	Hostapd + WiFi pineapple	Hostapd	Android, iOS

---

4. Practical Testbed / Setup Checklist

- **Network lab environment**: Set up IoT devices, gateways, mobile clients, cloud backend connections. Include typical deviceâ†”gatewayâ†”cloud flows.
- **Rogue network equipment**: Use a WiFi Pineapple or hostapd-WPE (wireless rogue AP), or a switch-span port + ARP-spoof tool to act as intermediary.
- **Traffic capture & analysis**: Place span port/mirror in gateway network; capture IoT device â†” gateway, mobile â†” cloud traffic using Wireshark/Zeek.
- **Mobile device instrumentation**: Use Android (rooted/emulated) and iOS (developer mode) devices. Install a custom CA certificate for interception. Use mitmproxy to inspect mobile app outbound traffic.
- **IoT device firmware inspection**: Extract firmware (Binwalk), analyze for usage of plain-text protocols, unverified TLS, insecure fallback. Simulate MiTM by intercepting device â†” cloud traffic and attempt injection/modification of commands/data.
- **Backend/cloud API review**: Check TLS enforcement, certificate validation, mutual authentication, endpoint whitelist, monitoring of new client IPs, new device registrations.
- **Detection & logging setup**: Configure SIEM/UEBA on cloud and network segments to alert on anomalies: e.g., device connecting from unexpected MAC, IoT device sending commands with altered payloads, mobile app sessions with invalid cert chain.

  Simulation of MiTM attacks:

  Mobile: Connect to rogue AP, intercept mobile app communications, attempt injection/modification of requests, observe backend and mobile app behaviour.

- IoT/gateway: ARP spoof gateway, redirect IoT device communications to malicious gateway, alter telemetry or commands, observe device authentication failures or backend detection.
- Cloud: Insert proxy between mobile/gateway and cloud API, modify TLS handshake or downgrade TLS version, see if cloud logs detect certificate anomalies.
- **Reporting and remediation**: Document vulnerable device types, network segments, apps lacking certificate pinning, backend APIs allowing weak TLS. Recommend mitigation: enforce TLS1.2+, certificate pinning on mobile, mutual auth for IoT devices, network segmentation, anomaly detection for MiTM. Retest after applying fixes.

---

5. References

1. Fereidouni, H., Fadeitcheva, O., & Zalai, M. (2025). IoT and manâ€inâ€theâ€middle attacks. *Security and Privacy*, 8(2), e70016.
2. Aoueileyine, M. O. E., Karmous, N., Bouallegue, R., Youssef, N., & Yazidi, A. (2024, April). Detecting and mitigating MiTM attack on IOT devices using SDN. In *International Conference on Advanced Information Networking and Applications* (pp. 320-330). Cham: Springer Nature Switzerland.
3. Thankappan, M., et al. (2023). Multi-channel Man-in-the-Middle Attacks Against Protected Wi-Fi Networks and Their Attack Signatures. In: Mercier-Laurent, E., Fernando, X., Chandrabose, A. (eds) Computer, Communication, and Signal Processing. AI, Knowledge Engineering and IoT for Smart Systems. ICCCSP 2023. IFIP Advances in Information and Communication Technology, vol 670. Springer, Cham. https://doi.org/10.1007/978-3-031-39811-7_22.
4. Alrubayyi, H., Alshareef, M. S., Nadeem, Z., Abdelmoniem, A. M., & Jaber, M. (2024). Security threats and promising solutions arising from the intersection of AI and IoT: a study of IoMT and IoET applications. *Future Internet*, 16(3), 85.

Security Testing for Eavesdropping Attacks

1. Overview

Eavesdropping attacks involve the interception and analysis of data transmitted over insecure communication channels. In the **cloud-mobile-IoT ecosystem**, these attacks often exploit:

- Unencrypted Wi-Fi or BLE (Bluetooth Low Energy) traffic.
- Insecure cloud API transmissions.
- Weak TLS/SSL configurations in mobile apps.
- Compromised IoT gateways leaking telemetry data.

Testing Objectives

- Evaluate end-to-end encryption between IoT devices, mobile clients, and cloud APIs.
- Detect insecure transmission protocols (HTTP, MQTT without TLS).
- Analyze wireless traffic for unprotected credentials or sensitive payloads.
- Validate the robustness of key management, certificate pinning, and session handling.

---

2. Testing Environment Configuration

- **Cloud Layer:** Deploy API servers and IoT brokers (e.g., AWS IoT Core, Azure IoT Hub) for data exchange testing.
- **Mobile Layer:** Install test apps with varying TLS configurations; intercept traffic via proxy tools.
- **IoT Layer:** Use IoT devices with Wi-Fi, BLE, and Zigbee for wireless transmission tests.
- **Network Layer:** Set up controlled Wi-Fi access point and packet sniffers for traffic capture.
- **Monitoring Layer:** Implement intrusion detection and SSL inspection to analyze captured traffic.

---

3. Testing Workflow

1. **Threat Modeling:** Identify components and protocols susceptible to interception.
2. **Static Code Analysis (SAST):** Review code for insecure cryptographic APIs, hardcoded keys, or missing encryption calls.
3. **Dynamic Analysis (DAST):** Intercept network traffic using proxies and sniffers.
4. **Wireless Security Testing:** Monitor RF signals for BLE/Zigbee/Wi-Fi data leakage.
5. **Protocol Validation:** Check for SSL/TLS misconfigurations, weak ciphers, or missing certificate validation.
6. **Reporting:** Document all intercepted sensitive data and recommend encryption or authentication hardening.

---

4. Security Testing Approach & Tools

```html

| Test Approach | Analysis Type | Approach Name | Testing Tool | Tool Hyperlink | Platform |
|---|---|---|---|---|---|
| Black-box | DAST | Network Packet Sniffing | Wireshark | Wireshark | Both |
| Gray-box | DAST | Traffic Interception via Proxy | Burp Suite | Burp Suite | Both |
| Gray-box | DAST | HTTPS Proxy and TLS Testing | OWASP ZAP | OWASP ZAP | Both |
| Black-box | DAST | Wireless Sniffing and Packet Capture | Aircrack-ng | Aircrack-ng | IoT |
| Gray-box | DAST | BLE and Zigbee Traffic Capture | Ubertooth One | Ubertooth One | IoT |
| White-box | SAST | Code Review for Cryptography | SonarQube | SonarQube | Both |
| Black-box | DAST | SSL/TLS Vulnerability Scanning | testssl.sh | testssl.sh | Both |
| Gray-box | DAST | Network Protocol Analysis | Ettercap | Ettercap | Both |
| Black-box | DAST | Man-in-the-Middle Testing | Bettercap | Bettercap | Both |
| Gray-box | DAST | Wi-Fi Traffic Monitoring | Kismet | Kismet | IoT |

```

---

5. References

1. Alaba, F. A., Othman, M., Hashem, I. A. T., & Alotaibi, F. (2017). Internet of Things security: A survey. *Journal of Network and Computer Applications, 88*, 10-28. https://doi.org/10.1016/j.jnca.2017.04.002.
2. Sadeghi, A. R., Wachsmann, C., & Waidner, M. (2015). Security and privacy challenges in industrial Internet of Things. *Proceedings of the 52nd Annual Design Automation Conference (DAC)*, 1-6. https://doi.org/10.1145/2744769.2747942.
3. Yang, Y., Wu, L., Yin, G., Li, L., & Zhao, H. (2017). A survey on security and privacy issues in Internet-of-Things. *IEEE Internet of Things Journal, 4*(5), 1250-1258. 10.1109/JIOT.2017.2694844.

4. Liu, Y., Li, Z., Shin, K. G., Yan, Z., & Liu, J. (2024). iCoding: Countermeasure against interference and eavesdropping in wireless communications. *IEEE Transactions on Information Forensics and Security*.
5. OWASP Foundation. (2023). *OWASP Mobile Security Testing Guide - Network Communication Testing.* https://owasp.org/www-project-mobile-security-testing-guide/.

Security Testing for CSRF Attacks

1. Overview

Cross-Site Request Forgery (CSRF) is an attack where an authenticated userâ€™s browser or app unintentionally executes unwanted actions on a web application in which they are authenticated.

In a **cloud-mobile-IoT ecosystem**, CSRF vulnerabilities may appear in:

- **Cloud APIs / Web Services:** Poor token validation in REST or GraphQL endpoints.
- **Mobile Apps:** Embedded WebViews or hybrid frameworks executing cloud commands with user credentials.
- **IoT Devices:** Web admin interfaces exposed without CSRF tokens or proper authentication controls.

Testing Objectives

- Verify implementation of CSRF tokens and SameSite cookies.
- Test mobileâ€“cloud API interactions for state-changing requests.
- Evaluate IoT web interfaces for anti-CSRF protections.
- Identify improper CORS (Cross-Origin Resource Sharing) configurations.
- Assess impact of credential reuse and session mismanagement.

---

2. Testing Environment Configuration

- **Cloud Layer:** Web applications and APIs deployed in a controlled cloud test environment (e.g., AWS, Azure, or Kubernetes).
- **Mobile Layer:** Android/iOS hybrid app (e.g., React Native, Flutter) that performs authenticated cloud operations.
- **IoT Layer:** Web interface (firmware emulation or device web admin panel) accessible via local network for CSRF token validation.
- **Proxy & Interceptor:** Traffic interception tools (Burp Suite, OWASP ZAP) to test forged requests.
- **Static & Dynamic Tools:** Code analysis tools (SonarQube, MobSF) to review anti-CSRF code patterns and configurations.

---

3. Testing Workflow

1. **Threat Modeling:** Identify all endpoints performing state-changing operations (e.g., POST, PUT, DELETE).
2. **SAST Analysis:** Review source code for missing CSRF tokens, weak session handling, or improper cookie attributes.
3. **DAST Testing:** Attempt to submit unauthorized requests from malicious domains.
4. **Proxy Testing:** Intercept API calls and replay with modified referer/origin headers.
5. **Mobile Testing:** Validate WebViews or embedded browsers for cross-origin requests.
6. **IoT Interface Testing:** Simulate forged admin requests (e.g., password change) through crafted HTML forms.
7. **Remediation Validation:** Confirm server validation for CSRF tokens and double-submit cookies.

---

4. Security Testing Approach & Tools

Test Approach	Analysis Type	Approach Name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST	Web Application Penetration Testing	OWASP ZAP	OWASP ZAP	Both
Gray-box	DAST	Proxy Interception and CSRF Token Testing	Burp Suite	Burp Suite	Both
White-box	SAST	Code Review and Token Validation Analysis	SonarQube	SonarQube	Both
White-box	SAST	Static Security Scanning for Mobile Apps	MobSF (Mobile Security Framework)	MobSF	Both

Black-box	DAST	Web Vulnerability Scanning	Acunetix	Acunetix	Both
Gray-box	DAST	Automated Web Fuzzing and Request Forgery	WFuzz	WFuzz	Both
White-box	SAST	Code Injection and CSRF Source Analysis	Checkmarx SAST	Checkmarx SAST	Both
Black-box	DAST	IoT Interface Security and Request Forgery	Firmware Analysis Toolkit	Firmware Analysis Toolkit	IoT
Gray-box	DAST	Request Monitoring and Header Validation	Wireshark	Wireshark	Both

---

5. References

1. Calzavara, S., Conti, M., Focardi, R., Rabitti, A., & Tolomei, G. (2020). Machine learning for web vulnerability detection: the case of cross-site request forgery. *IEEE Security & Privacy*, 18(3), 8-16.
2. Shahriar, H., & Zulkernine, M. (2010, November). Client-side detection of cross-site request forgery attacks. In *2010 IEEE 21st International Symposium on Software Reliability Engineering* (pp. 358-367). IEEE.
3. Biswas, J., Hasan, M., Saiful, M., Mim, F. T., & Tasnim, N. (2023, December). A Review on Mitigating Security Risks: Effective Strategies to Prevent Cross-Site Request Forgery Vulnerabilities. In *International Conference on Cyber Intelligence and Information Retrieval* (pp. 309-317). Singapore: Springer Nature Singapore.
4. Singh, Y., Goel, P., Aggarwal, S., Chaudhary, R., & Budhiraja, I. (2024, December). Mitigating Cross-Site Request Forgery Vulnerabilities: An Examination of Prevention Systems. In *2024 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)* (pp. 55-60). IEEE.
5. Singh, R., Gupta, M. K., Patil, D. R., & Patil, S. M. (2024, April). Analysis of web application vulnerabilities using dynamic application security testing. In *2024 IEEE 9th International Conference for Convergence in Technology (I2CT)* (pp. 1-6). IEEE.

Security Testing for SQL Injection

1. Overview

A SQL Injection attack is a code injection technique that allows attackers to manipulate a database through insecure user input. Security testing is essential to detect and prevent these vulnerabilities, protecting sensitive data and ensuring application integrity.

---

2. Security Test & Approach

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST	Automated SQLi discovery & exploitation	sqlmap	sqlmap	Both
Gray-box	DAST	Intercept & manipulate requests (manual testing)	Burp Suite (Intruder, Repeater)	Burp Suite	Both
Black-box	DAST	Automated web scanning (includes SQLi checks)	OWASP ZAP	OWASP ZAP	Both
Gray-box	DAST	Web app vulnerability discovery (fuzzing)	w3af / nikto	w3af / nikto	Both
White-box	SAST	Source review for unsafe DB calls / concatenation	Semgrep / CodeQL	Semgrep / CodeQL	Cloud & mobile backend
Gray-box	DAST	API fuzzing & parameter testing	Postman + Fuzzers (Boomerang, RESTler)	Postman / Fuzzers	Both (APIs)

Gray-box	DAST	Database & query monitoring (detect abnormal queries)	DB audit logs / SIEM (Elastic/Splunk)	DB audit logs / SIEM	Cloud

---

3. Minimal Testbed & Quick Steps

1. **Scope & inventory** — list endpoints that accept input: web forms, API params, MQTT/CoAP fields forwarded to DB, mobile app inputs, IoT gateway admin interfaces.
2. **Backup & isolate** — run tests in staging or isolated environment; snapshot DBs and apps.
3. **Baseline capture** — enable DB auditing (slow query log, general log) and capture normal traffic.
4. **Automated scan** — run `sqlmap` and ZAP against targets to find obvious SQLi. Use authenticated scans for API endpoints (bearer tokens / session cookies).
5. **Manual verification** — use Burp Suite (Repeater/Intruder) to craft payloads, test blind/time-based SQLi, boolean blind, and stacked queries. Observe DB/log responses & side effects.
6. **API/mobile checks** — fuzz API parameters (Postman + RESTler/Boomerang) and test mobile app inputs (intercept with Burp/mitmproxy or instrument mobile app).
7. **DB & app log review** — correlate suspicious requests with DB logs; verify whether parameterized queries are used or unsafe concatenation.
8. **Fix & retest** — apply prepared statements/ORM parameterization, input validation, least privilege DB accounts, and retest. Use WAF as compensating control if immediate code fixes are infeasible.

---

4. References

1. Alsalamah, M., Alwabli, H., Alqwifli, H., & Ibrahim, D. M. (2021). A review study on SQL injection attacks, prevention, and detection.
2. Paul, A., Sharma, V., & Olukoya, O. (2024). SQL injection attack: Detection, prioritization & prevention. *Journal of Information Security and Applications*, 85, 103871.
3. Ojagbule, O., Wimmer, H., & Haddad, R. J. (2018, April). Vulnerability analysis of content management systems to SQL injection using SQLMAP. *In SoutheastCon 2018* (pp. 1-7). IEEE.
4. Bouafia, R., Benbrahim, H., & Amine, A. (2023, October). Automatic protection of web applications against sql injections: An approach based on acunetix, burp suite and sqlmap. In *2023 9th International Conference on Optimization and Applications (ICOA)* (pp. 1-6). IEEE.
5. Liu, M., Li, K., & Chen, T. (2020, July). DeepSQLi: Deep semantic learning for testing SQL injection. *In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 286-297).

Security Testing for Cross-Site Scripting

1. Overview

XSS remains one of the most common web vulnerabilities (reflected, stored, DOM), and it affects cloud apps, webviews inside mobile apps and browser-based UIs of IoT devices. Use both SAST (to find bad sanitization/templating) and DAST (to find runtime/DOM issues).

---

2. Security Test Approaches & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST	Automated XSS scanning	Burp Suite (Scanner)	Burp Suite	Both
Black-box / Gray-box	DAST	Parameter & DOM XSS scanning	DalFox	DalFox (GitHub)	Both
Black-box	DAST	Smart XSS fuzzing & payload generation	XSStrike	XSStrike (GitHub)	Both
Gray-box	DAST / Dynamic	Client-side / DOM instrumentation	Burp DOM Invader (or DOM Inspector)	DOM Invader (PortSwigger)	Both (webviews included)
Gray-box	SAST	Static analysis of templating/escaping	CodeQL, SonarQube	CodeQL	Both
Gray-box	SAST / Mobile	Mobile app scanning (webview sources)	MobSF (static + dynamic)	MobSF	Android, iOS

Black-box	DAST	Automated crawler/fuzzer for web panels of IoT	OWASP ZAP (active scan + fuzzer)	OWASP ZAP	Both
Gray-box	DAST / Monitoring	Browser automation for payload execution & verification	Selenium + headless browsers (Puppeteer)	Puppeteer	Both
Black-box	DAST	Reflected/Stored XSS PoC & exploitation	Custom Scapy/requests scripts, XSS payload libraries	custom	Both
White-box	SAST / Controls	Policy & mitigation review (CSP, sanitizers)	Manual checklists + automated CSP scanners	OWASP CSP Cheat Sheet	Both

3. Short Testing Setup — Practical steps

1. **Scope & authorization** — define target (cloud web app, mobile app webviews, IoT device web UI) and obtain written permission for each environment.
2. **Inventory inputs & sinks** — enumerate all user-controllable inputs (query parameters, POST bodies, headers, cookies, stored fields, webview `addJavascriptInterface`, message handlers) and all sinks (innerHTML, document.write, eval, setTimeout/src=, location, DOM APIs). Use automated crawling + manual review. ([PortSwigger][2])
3. **SAST (white-box)** — run CodeQL / SonarQube on server & frontend code to find insecure encoding/templating, use of unsafes like `innerHTML` or `eval`, and missing output encoding for contexts (HTML, attribute, JS, URL, CSS). Inspect mobile source (or decompiled APK/IPA) for webview APIs exposing JS interfaces. ([cheatsheetseries.owasp.org][3])
4. **DAST — automated scanning** — run Burp/ZAP scans and DalFox/XSStrike against parameter lists and known pages (including login flows). Use Burpâ€™s DOM Invader to find client-side sinks and try DOM payloads. Verify findings with headless browsers (Puppeteer) to ensure payload executes in real client context. ([Dalfox][4])
5. **Manual validation & exploit dev** — craft context-aware payloads (HTML, attribute, JavaScript, event handlers). Try stored XSS chains (submit payload to persistent fields) and propagate to other user roles. For mobile, test webviews (in-app browser) by embedding payload in expected inputs and monitor console/alert/exfil events.
6. **IoT UI testing** — many IoT admin panels are simple web appsâ€”scan with ZAP/Arachni, fuzz form fields and firmware update parameters, and attempt stored XSS in device status pages or logs. Use network captures and serial logs where possible.
7. **Mitigation verification** — verify proper output encoding per context, use of secure templating libraries, HTTPOnly for session cookies, proper CSP headers, and that mobile webviews disable dangerous flags (e.g., `setAllowFileAccessFromFileURLs`, unnecessary JS bridges). Validate CSP policy effectiveness by attempting allowed payloads. ([cheatsheetseries.owasp.org][5])
8. **Reporting & remediation steps** — for each confirmed XSS: include PoC, affected contexts, exploited path, remediation (contextual encoding, sanitize/escape, CSP recommendations), and regression tests.

4. Quick Checklist (priority test cases)

- Reflected XSS: test URL params, Referrer, headers.
- Stored XSS: form inputs, profile fields, device logs, firmware metadata.
- DOM XSS: client-side sinks (`innerHTML`, `outerHTML`, `insertAdjacentHTML`, `eval`, `new Function`, `setAttribute` with untrusted input). Use DOM Invader to identify sinks.
- Webview XSS: ensure in-app webviews are configured securely (disable remote debugging in production, limit JS bridges). Use MobSF to find webview exposures.
- CSP & sanitizers: check page CSP headers and verify that output encoding libraries (e.g., DOMPurify) are used correctly.

5. References

1. Altulaihan, E. A., Alismail, A., & Frikha, M. (2023). A survey on web application penetration testing. Electronics, 12(5), 1229.
2. Nagpure, S., & Kurkure, S. (2017, August). Vulnerability assessment and penetration testing of web application. In 2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA) (pp. 1-6). IEEE.
3. Goutam, A., & Tiwari, V. (2019, November). Vulnerability assessment and penetration testing to enhance the security of web application. In 2019 4th International Conference on Information Systems and Computer Networks (ISCON) (pp. 601-605). IEEE.

Security Testing for Server-Side Request Forgery

1. Overview

Server-Side Request Forgery (SSRF) lets an attacker coerce a server to make network requests it should not (internal services, cloud metadata, IoT endpoints), which is especially dangerous in cloud environments (IMDS / instance metadata).

2. Security Test Approach & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST	Automated SSRF fuzzing	SSRFmap	SSRFmap	Both
Gray-box	DAST	Blind SSRF detection (out-of-band)	Burp Collaborator / Interactsh	Burp Collaborator / Interactsh	Both
Gray-box	DAST	Manual request inspection & manipulation	Burp Suite / OWASP ZAP	Burp Suite / OWASP ZAP	Both
White-box	SAST	Code review for unsafe URL handling	Semgrep / CodeQL	Semgrep / CodeQL	Cloud & backend
Gray-box	DAST	Internal port / service discovery via SSRF	Custom HTTP probes + timing/response analysis (curl, httpx)	httpx	Both
White-box	SAST/DAST	Cloud metadata / IMDS safety checks	Cloud vendor docs & IMDS probes	See AWS / Azure IMDS docs	Cloud
Gray-box	DAST	Network detection / logging	Zeek / ELK / Splunk	Zeek / ELK	Both

3. Minimal Testbed & Quick Procedure

1. **Authorization & isolation** — run all tests in staging / isolated VLAN; snapshot systems and get written approval.
2. **Inventory** — list endpoints that accept external URLs or hostnames: image fetchers, webhooks, URL previews, redirects, server-side fetch endpoints on mobile/gateways.
3. **Baseline logging** — enable HTTP logs, backend request logs, and cloud audit logs; configure an Interactsh/Burp Collaborator domain to catch OOB callbacks.
4. **Automated fuzz** — run SSRFmap (or similar) against parameters identified as URL/host inputs to detect open fetch behaviors and common bypasses.
5. **Blind SSRF detection** — inject Collaborator / Interactsh payloads and monitor for DNS/HTTP/SMTP callbacks (use for blind SSRF where response not returned to user).
6. **Metadata & internal service checks** — probe for cloud metadata endpoints (e.g., AWS IMDS v1/v2), internal hosts (`localhost`, `169.254.169.254`, `169.254.169.254/latest/meta-data/`), and common internal ports/services via SSRF to confirm exposure (do not request sensitive data unless authorized).
7. **Manual exploit & bypass testing** — use Burp Suite/ZAP to try different payload encodings, alternate protocols (file://, gopher://, ftp://), DNS rebinding techniques and filter bypasses.
8. **Detection validation** — ensure SIEM detects unusual outbound internal requests, repeated parametrized fetches, or OOB callbacks; log findings and assign severity.
9. **Remediate & retest** — apply allow-lists, require URL validation, enforce network egress controls, enforce IMDS v2 and metadata access restrictions, and retest.

4. References

1. Wang, E., Chen, J., Xie, W., Wang, C., Gao, Y., Wang, Z., ... & Wang, B. (2024, May). Where urls become weapons: Automated discovery of ssrf vulnerabilities in web applications. In 2024 IEEE Symposium on Security and Privacy (SP) (pp. 239-257). IEEE.
2. Altulaihan, E. A., Alismail, A., & Frikha, M. (2023). A survey on web application penetration testing. Electronics, 12(5), 1229.
3. Luo, H. (2019, July). SSRF vulnerability attack and prevention based on php. In 2019 International Conference on Communications, Information System and Computer Engineering (CISCE) (pp. 469-472). IEEE.
4. Jabiyev, B., Mirzaei, O., Kharraz, A., & Kirda, E. (2021, March). Preventing server-side request forgery attacks. In Proceedings of the 36th Annual ACM Symposium on Applied Computing (pp. 1626-1635).

Security Testing for Command Injection Attacks

1. Overview

Command Injection occurs when user-controlled input is passed to a system command interpreter (e.g., shell, CLI, OS command) without adequate validation or sanitization.

In a **cloud-mobile-IoT** ecosystem, it can occur in:

- **Cloud layer:** APIs or serverless functions that execute shell commands (e.g., log parsing, file management).
- **Mobile layer:** input fields or WebViews sending unfiltered data to back-end services.
- **IoT layer:** device firmware or web interfaces that execute system commands (e.g., ping, traceroute, diagnostic commands).

Testing Objectives

- Detect command injection vulnerabilities in cloud APIs, mobile applications, and IoT firmware.
- Validate input handling and command execution boundaries.
- Verify that user data cannot modify command logic or system calls.
- Ensure serverless and IoT runtime environments apply proper isolation (sandboxing, least privilege).

---

2. Testing Environment Configuration

- **Cloud Testbed:** Deploy a microservice-based API (e.g., Node.js, Python Flask) inside a sandboxed container (Docker/Kubernetes).
- **Mobile Application:** Build a client app (Android/iOS) communicating with the cloud API; test input sanitization and command triggers.
- **IoT Device Simulation:** Emulate IoT firmware (QEMU/Firmadyne) with a command execution interface (e.g., ping.cgi, traceroute.cgi).
- **Network Proxy Setup:** Use Burp Suite or OWASP ZAP to intercept traffic between app and cloud.
- **Static/Dynamic Scanners:** Employ SAST and DAST to detect risky `system()`, `exec()`, or similar functions.

---

3. Testing Workflow

1. **Threat modeling:** identify modules using command interpreters.
2. **Static testing (SAST):** scan source code for OS command execution functions and tainted inputs.
3. **Dynamic testing (DAST):** inject payloads (e.g., `; ls`, `&& whoami`, `| cat /etc/passwd`) via API requests and mobile inputs.
4. **Fuzzing:** test APIs and IoT endpoints with malformed commands.
5. **Validation:** verify results using sandbox logs and alerting systems (no real system harm).
6. **Remediation review:** recommend input whitelisting, escaping, and use of parameterized APIs.

---

4. Security Testing Approach & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST	Web/Mobile API Penetration Testing	OWASP ZAP	OWASP ZAP	Both
Gray-box	DAST	Intercept & Injection Testing via Proxy	Burp Suite	Burp Suite	Both
White-box	SAST	Code Review / Static Analysis	SonarQube	SonarQube	Both
White-box	SAST	Source Code Security Scanning	Checkmarx SAST	Checkmarx SAST	Both
Black-box	DAST	Web Vulnerability Scanning	Acunetix	Acunetix	Both
Gray-box	DAST	Fuzzing for Command Injection	Boofuzz	Boofuzz	Both
White-box	SAST	Mobile Source Analysis	MobSF	MobSF	Both
Gray-box	DAST	Firmware Emulation & Command Testing	Firmadyne / Binwalk	Firmadyne	IoT

Black-box	DAST	Network Packet Sniffing & Response Monitoring	Wireshark	Wireshark	Both
Gray-box	DAST	Runtime Application Protection	Appdome / RASP Tools	Appdome	Both

5. References

1. Antunes, N., & Vieira, M. (2015). Benchmarking vulnerability detection tools for web services. *IEEE Transactions on Services Computing*, 8(2), 269-283.
2. Fonseca, J., Vieira, M., & Madeira, H. (2008). Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks. *Proceedings of the IEEE Pacific Rim Dependable Computing Conference (PRDC)*, 365-372.
3. Chimuco, F.T., Sequeiros, J.B.F., Lopes, C.G. et al. Secure cloud-based mobile apps: attack taxonomy, requirements, mechanisms, tests and automation. *International Journal of Information Security* 22, 833-867 (2023). https://doi.org/10.1007/s10207-023-00669-z.
4. Noman, H. A., & Abu-Sharkh, O. M. (2023). Code injection attacks in wireless-based Internet of Things (IoT): A comprehensive review and practical implementations. *Sensors*, 23(13), 6067.
5. Benkhelifa, E., Zhukabayeva, T., & Ennaji, S. (2025). Next-generation penetration testing: a cross-domain review of challenges, trends, and taxonomy for urban digital ecosystems. *Computing*, 107(12), 224.

Security Testing for Code Injection Attacks

1. Overveiw

Scenario: Code Injection attacks in a *cloud-mobile-IoT* ecosystem occur when untrusted data is interpreted as executable code within one of the following layers:

- **Cloud layer:** APIs, microservices, serverless functions, or backend databases (e.g., SQL injection, template injection).
- **Mobile layer:** application code (Android/iOS) consuming user input or remote APIs (e.g., JavaScript injection, intent injection).
- **IoT layer:** embedded firmware, edge controllers, and gateways (e.g., command injection, buffer overflow via payload).

Testing Objectives -

- Identify and mitigate injection points in communication and code paths.
- Validate sanitization and input-validation mechanisms across components.
- Confirm backend API and database query security.
- Verify runtime protection mechanisms (e.g., WAF, RASP, sandboxing).

---

2. Testing Environment

- **Mobile app sandbox:** Android Studio Emulator or iOS Simulator, configured with proxy (Burp / OWASP ZAP).
- **Cloud backend:** Containerized environment (Docker + API endpoints) deployed in test VPC.
- **IoT devices:** Simulated using Raspberry Pi or ESP32 with MQTT/HTTP clients; run firmware-emulation tools like QEMU or Firmadyne.
- **Network monitoring:** MITM proxy (Burp Suite), API Gateway logs, network packet capture (Wireshark).
- **Static/dynamic analyzers:** SAST and DAST tools for code scanning, runtime behavior tracing.

---

3. Testing Workflow

1. **Threat modeling:** identify trust boundaries where unvalidated input may execute.
2. **Static analysis (SAST):** scan application source and infrastructure-as-code for injection vulnerabilities.
3. **Dynamic analysis (DAST):** execute fuzzing and runtime request injection via proxies.
4. **Hybrid/Gray-box:** combine both SAST and DAST results for correlation and remediation.
5. **Reporting:** document vulnerable endpoints, risk rating (e.g., CVSS), and exploit proof-of-concept (PoC) in a safe environment.

---

3. Security Testing Approach & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST	Web/Mobile API Penetration Testing	OWASP ZAP	OWASP ZAP	Both

Gray-box	DAST	Intercept & Injection via Proxy	Burp Suite	Burp Suite	Both
White-box	SAST	Static Code Analysis / Review	SonarQube	SonarQube	Both
White-box	SAST	Code Security Scanning (Cloud & Mobile)	Checkmarx SAST	Checkmarx SAST	Both
Black-box	DAST	Automated Vulnerability Scanning	Nessus	Nessus	Both
White-box	SAST	Mobile Source Code Review	MobSF	MobSF	Both
Gray-box	DAST	Fuzzing Inputs and API Endpoints	Boofuzz / Peach Fuzzer	Boofuzz	Both
White-box	SAST	IoT Firmware Static Analysis	Firmadyne / Binwalk	Firmadyne	IoT
Gray-box	DAST	Runtime Protection & Code Injection Detection	Appdome / RASP tools	Appdome	Both
Black-box	DAST	Network Traffic Analysis	Wireshark	Wireshark	Both

4. References

1. Halfond, W.G., Viegas, J., & Orso, A. (2006). A Classification of SQL Injection Attacks and Countermeasures. *International Symposium on Signals, Systems, and Electronics*.
2. Fonseca, J., Vieira, M., & Madeira, H. (2007, December). Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks. In *13th Pacific Rim international symposium on dependable computing (PRDC 2007)* (pp. 365-372). IEEE.
3. Singh, R., Gupta, M. K., Patil, D. R., & Patil, S. M. (2024, April). Analysis of web application vulnerabilities using dynamic application security testing. In *2024 IEEE 9th International Conference for Convergence in Technology (I2CT)* (pp. 1-6). IEEE.
4. Ghaffarian, S. M., & Shahriari, H. R. (2017). Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. *ACM computing surveys (CSUR)*, 50(4), 1-36.

Security Testing Setup for Audit-Log Manipulation Attacks

1. Overview

Attackers or malicious insiders can delete, alter, truncate, inject, or replay audit logs to hide activity or create false evidence. Robust logging practices (remote/immutable storage, integrity checks, tamper detection) are required to preserve forensic value.

2. Security Test Approach & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Gray-box	DAST	Auditd / journald tamper tests (stop/rotate/delete)	auditd, journalctl, wevtutil (Windows)	auditd / journalctl	Both
White-box	DAST	Filesystem integrity & tamper detection	AIDE, Tripwire	AIDE / Tripwire	Both
Gray-box	DAST	Log forwarding & ingestion tamper / replay	Elastic Stack (Filebeat/Logstash), Splunk (UF/HEC)	Elastic Stack / Splunk	Cloud

Black-box	DAST	Log injection & log forging (newline/format attacks)	Burp Suite / custom HTTP payloads / scapy	Burp Suite / custom HTTP payloads	Both
White-box	SAST/DAST	Append-only / WORM enforcement & verification	immudb / S3 Object Lock tests	immudb / S3 Object Lock tests	Cloud
White-box	SAST	Forward-integrity & signed logs	Custom signing (HMAC/RSA) + verification scripts	See Bellare & Yee (Forward Integrity) and implementation guides	Both
Gray-box	DAST	Timeline reconstruction & missing-entries detection	Plaso / The Sleuth Kit / auditbeat + ELK	/	Both

3. Minimal Testbed & Components

- **Staging hosts:** representative cloud VM, IoT gateway, Android/iOS test clients.
- **Local audit agents:** auditd (Linux), systemd-journal, Windows Event logging.
- **Log pipeline:** Filebeat/Logstash â†' Elastic / Splunk forwarder/HEC (staging).
- **Integrity tools:** AIDE/Tripwire for FIM; immudb or S3 Object Lock for immutable storage tests.
- **Detection & analytics:** Wazuh / Elastic / Splunk rules for missing entries, unusual rotations, ingestion gaps.
- **Forensics:** The Sleuth Kit / Plaso for timeline reconstruction.

4. Short Test Workflow

1. **Baseline & logging hardening**

- Ensure audit policies are enabled (what to log, retention). Collect baseline event rates and typical log sizes.

2. **Simulate tamper techniques (in isolated lab)**

- **Service stop / disable:** stop auditd/journald or turn off Windows Event logging and observe detection/alerting.
- **Log deletion/truncation:** delete/truncate local files, remove rotated archives, attempt to tamper with archived logs.
- **Log rotation abuse:** modify rotation scripts to prematurely rotate/compress or overwrite logs.
- **Timestamp manipulation:** change host clock (NTP) to alter timestamps or cause reordering.
- **Log injection / forging:** send specially crafted inputs (HTTP fields, device telemetry) that inject spoofed log lines or create fake entries; test whether parsers are vulnerable.
- **Replay & resend:** re-send old log batches or replay events to SIEM to simulate replay attacks.
- **Log forwarding compromise:** simulate compromised forwarder (Filebeat / Splunk UF) that filters out events before shipping.

For each simulation, capture: what was altered, do local logs show deletion, does the remote collector still have originals, do integrity checks detect differences.

3. **Integrity validation & detection tests**

- **FIM:** verify AIDE/Tripwire alerts on file modifications.
- **Signed logs:** verify forward-integrity/signed log chains (HMAC chaining) detect alterations. (Bellare & Yee forward-integrity technique.)
- **Remote immutable storage:** check immudb or S3 Object Lock prevents deletion/alteration and that verification scripts detect tamper attempts.
- **SIEM correlations:** check for gaps in expected sequence numbers / heartbeat events; configure SIEM to alert on missing periodic events.

4. **Forensic reconstruction test**

- Use Plaso / Sleuth Kit to reconstruct timeline from remaining artifacts; confirm manipulability affects investigations and measure residual evidence left by each tamper pattern.

5. **Remediation & retest**

- Apply mitigations (remote append-only logging, signed logs, restricted forwarder creds, enforce immutability) and re-run tamper scenarios confirming detection or inability to delete/alter.

5. References

1. Kent, K., & Souppaya, M. (2006). Guide to computer security log management. *NIST special publication*, 92, 1-72.
2. Bellare, M., & Yee, B. (1997). Forward integrity for secure audit logs (Vol. 184). *Technical report, Computer Science and Engineering Department*, University of California at San Diego.
3. Zawoad, S., Dutta, A. K., & Hasan, R. (2013, May). SecLaaS: secure logging-as-a-service for cloud forensics. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security* (pp. 219-230).

Security Testing set-up for Brute-Force Attacks

1. Overview

Validate how resilient your cloud APIs, mobile apps and IoT gateways/devices are to automated guessing (credential brute-force, credential-stuffing, PIN/PATTERN attempts, protocol-level password guessing) and verify detection + throttling/lockout controls.

---

2. Security Test Approach & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST	Online credential brute-force / credential stuffing	Hydra (THC-Hydra)	Hydra	Both
Black-box	DAST	Protocol login brute (SSH/Telnet/FTP)	Ncrack	Ncrack	Both (IoT heavy)
Black-box	DAST	Web UI / API fuzzing & automated attack	Burp Suite (Intruder) / OWASP ZAP	Burp Suite / OWASP ZAP	Both
Gray-box	DAST	Custom login flows & throttling test	Patator (multi-module brute)	Patator	Both
Gray-box	SAST	Static review for auth logic & rate-limiting bugs	Semgrep / CodeQL	Semgrep / CodeQL	Cloud & mobile
Gray-box	DAST	Offline hash cracking (captured hashes)	Hashcat / John the Ripper	Hashcat / John the Ripper	Both
Gray-box	DAST	Mobile PIN/credential test & instrumentation	Frida / Drozer (Android) / TestFlight + MDM (iOS)	Frida / Drozer	Android, iOS
White-box	DAST	IoT default creds & factory password scanning	Shodan (discovery) + custom scripts	Shodan	IoT
Gray-box	DAST	Detection & monitoring validation	Zeek / Suricata + ELK / Splunk	Zeek / Suricata / Splunk	Cloud / Network

---

3. Minimal Testbed & Quick Step-by-step

Preconditions (must): written authorization, staging environment that mirrors production auth flows (rate limits, DB, captive portals), backups/snapshots, and escalation/kill procedure.

1. **Inventory & threat modelling**
   - Enumerate endpoints that accept credentials (web logins, APIs, device management ports, telnet/SSH, mqtt broker logins, mobile PIN entry flows, OTA agent endpoints). Note account lockout/policy settings.
2. **Baseline & logging**
   - Enable authentication logging, WAF logs, and SIEM ingestion. Record normal failed/successful login rates and typical IP ranges.
3. **Automated credential stuffing (low-rate)**
   - Use a curated test credential list (do not use real stolen credentials) and run Hydra/Patator against staging login endpoints, starting with low request rates to validate rate-limit handling. Monitor for account lockouts and SIEM alerts.
4. **Protocol brute & default credential checks (IoT)**
   - Test device protocols (Telnet/SSH/FTP/HTTP admin) with Ncrack/Hydra and common default credential lists (manufacturer defaults). For discovery use controlled Shodan queries in permitted scope.
5. **Mobile PIN & instrumentation tests**
   - For Android: use Frida or Drozer to instrument and attempt automated PIN entry on test devices or verify lockout thresholds. For iOS use MDM test profiles to verify lockout and wipe policies (iOS blocks brute for PIN via hardware limits).
6. **Offline hash cracking (if hashes available in scope)**

- If you have database dumps in scope (sanitized/test data), run Hashcat/John with appropriate wordlists and rules to measure password strength and expected compromise time.

7. **Rate-limit & throttling verification**

- Verify backend enforces exponential backoff, per-account and per-IP throttling, CAPTCHAs after threshold, progressive delays, and account lockout policies (with safe rollback for tests).

8. **Detection validation**

- Confirm SIEM/WAF/IDS alerts on sudden spike of failed auth attempts, IP reputation hits, many credential fails across many accounts (credential stuffing), or unusual geo-pattern (rapid geolocation changes).

9. **Remediation testing**

- Verify MFA enrollment/requirement prevents takeover, implement IP reputation blocking, enforce strong password policies and breach-credential checking (haveibeenpwned API or similar), then re-run tests to ensure mitigation.

---

4. References

1. Florencio, D., & Herley, C. (2007, May). A large-scale study of web password habits. In *Proceedings of the 16th international conference on World Wide Web* (pp. 657-666).
2. Sequeiros, J. B., Chimuco, F. T., Samaila, M. G., Freire, M. M., & InÃ¡cio, P. R. (2020). Attack and system modeling applied to IoT, cloud, and mobile ecosystems: Embedding security by design. *ACM Computing Surveys (CSUR)*, 53(2), 1-32.
3. Bonneau, J., & Preibusch, S. (2010, June). The Password Thicket: Technical and Market Failures in Human Authentication on the Web. In *WEIS*.

Security Testing Setup for Cryptanalysis Attacks

1. Overview

Evaluate whether cryptographic primitives, implementations, keys and protocols used across cloud, mobile and IoT are vulnerable to practical cryptanalysis (mathematical attacks, brute-force/offline key recovery, side-channel/fault attacks, randomness weakness, protocol misuse), and verify detection & remediation.

---

2. Security Testing Approach & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
White-box	SAST	Static crypto-use & API misuse review (key handling, RNG use)	Semgrep, CodeQL	Semgrep / CodeQL	Both
White-box	SAST	Key management & provisioning review	NIST guidance checks, manual review, cryptotooling	NIST guidance checks	Cloud / Both
Black-box	DAST	Offline key recovery / brute-force & dictionary attacks	Hashcat / John the Ripper / RsaCtfTool	Hashcat / John the Ripper / RsaCtfTool	Both
Gray-box	DAST	Mathematical factorization/discrete-log experiments	msieve / yafu / SageMath / PARI-GP	msieve / yafu / SageMath	Both
Gray-box	DAST	Randomness tests (PRNG/entropy validation)	dieharder / NIST STS / ent	dieharder / NIST STS	Both
Gray-box	DAST	Protocol & implementation fuzzing (TLS/crypto APIs)	tlsfuzzer / boofuzz / OSS-Fuzz (for libraries)	tlsfuzzer / boofuzz	Both
Black-box	DAST	Side-channel / power & EM analysis	ChipWhisperer / Riscure tools	ChipWhisperer / Riscure tools	IoT / mobile / Both
Black-box	DAST	Fault injection & glitching tests (fault-based cryptanalysis)	ChipWhisperer / FGPA glitcher / voltage glitch rig	ChipWhisperer	IoT / Both

Gray-box	SAST/DAST	Firmware/Library extraction & crypto primitive verification	Binwalk / Ghidra / radare2 / OpenSSL test vectors	Binwalk / Ghidra	IoT / Both
White-box	SAST	Entropy source & seed-provision auditing	Source review + test harness (openssl, rdrand checks)	openSSL	Both

3. Minimal Testbed & Quick Workflow

1. **Scope & approvals** — define targets (cloud services, mobile apps, IoT firmware), get written authorization, isolate testbed and backups.
2. **Inventory crypto usage** — enumerate algorithms, key sizes, RNGs, certs, key stores (HSM, keystore, TPM, Secure Enclave).
3. **Static checks (SAST)** — run Semgrep/CodeQL for API misuse (e.g., `RAND_bytes` misuse, ECB mode, hard-coded keys), review key lifecycle and storage, confirm TLS configurations and cert validation.
4. **Randomness testing** — collect entropy outputs and run dieharder / NIST STS to detect weak PRNGs or low entropy seeds.
5. **Offline cryptanalysis** — collect ciphertexts / public keys (within scope) and attempt practical attacks: Hashcat / John against captured hashes; RsaCtfTool, msieve/yafu/Sage for weak RSA/DSA keys; attempt small-exponent attacks or reused-nonce attacks (e.g., ECDSA nonce reuse).
6. **Protocol & implementation fuzzing** — fuzz TLS endpoints, crypto library APIs and parsing code (tlsfuzzer, boofuzz, OSS-Fuzz where applicable).
7. **Side-channel & fault lab (lab only)** — in shielded bench, perform power/EM traces and fault injection on IoT devices or secure elements to attempt key recovery (ChipWhisperer). Log traces, run CPA/DPA analysis.
8. **Firmware reverse & verification** — extract firmware, locate crypto code paths, verify use of constant-time primitives and safe libraries.
9. **Report & remediate** — list vulnerable primitives/parameters, weak randomness, poor key handling, exploitable side-channels/faults; recommend mitigations (use vetted libraries, HSMs/secure enclaves, constant-time code, larger keys, proper seeding, disable insecure curves). Retest after fixes.

References

1. Menezes, A. J., Van Oorschot, P. C., & Vanstone, S. A. (2018). Handbook of applied cryptography. *CRC press*.
2. Egele, M., Brumley, D., Fratantonio, Y., & Kruegel, C. (2013, November). An empirical study of cryptographic misuse in android applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (pp. 73-84).
3. Shuai, S., Guowei, D., Tao, G., Tianchang, Y., & Chenjie, S. (2014, August). Modelling analysis and auto-detection of cryptographic misuse in android applications. In *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing* (pp. 75-80). IEEE.
4. Muslukhov, I., Boshmaf, Y., & Beznosov, K. (2018, May). Source attribution of cryptographic api misuse in android applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security* (pp. 133-146).

Security Testing Setup for Phishing Attacks

1. Overview

Security testing against **Phishing Attacks** in the cloud-mobile-IoT ecosystem focuses primarily on two areas: **User Resilience** (simulating campaigns to measure human failure rates) and **Technical Defense** (testing mobile applications and cloud infrastructure ability to detect, block, and mitigate compromised credentials).

The setup below models both the behavioral and technical defense aspects of phishing resilience.

2. Security Testing Approach & Tools

Test Approach	Analysis Type	Approach Name	Testing Tool	Tool Hyperlink	Platform
Gray-box	DAST	Ethical Phishing Campaign Simulation	GoPhish	GoPhish	Both (User-facing)
Black-box	DAST	URL/Domain Reputation Testing	Google Safe Browsing API	Google Safe Browsing	Both (OS/Browser)
White-box	SAST	Code Review (Credential Leakage)	TruffleHog	TruffleHog	Android, iOS, Cloud Backend

Gray-box	DAST	Credential Handling Validation	Mobile Proxy (Burp Suite Pro/OWASP ZAP)	Burp Suite Pro	Both (Mobile App)
White-box	SAST	UI Spoofing Defense Review	Manual Code Review	N/A (Manual process)	Android, iOS

3. Detailed Testing Setup

The testing setup models the three phases of a phishing attack: delivery, compromise, and post-compromise mitigation.

A. Simulating the Attack (Ethical Phishing Campaign)

- **Procedure:** A **Red Team** sets up a controlled phishing environment using a framework like **GoPhish**. Emails or SMS messages are crafted to look like they are from the target organization (e.g., cloud provider login, mobile app verification) and sent to a defined pool of employees.
- **Goal:** Measure the **Click Rate** (number of users who click the link) and the **Credential Entry Rate** (number of users who submit credentials on the fake login page). This is primarily a **user-resilience metric**.

B. Client-Side Defense Testing (Black-box/DAST)

- **Procedure:** The phishing link created by **GoPhish** is fed into the **Google Safe Browsing API** (or similar third-party domain reputation services) to test the security filters embedded in the user mobile browser or operating system.
- **Goal:** Verify that the built-in phishing protection features of the OS (Android/iOS) or browser successfully detect the malicious domain and display a **warning page** before the user can interact with the phishing content, thus neutralizing the attack.

C. Post-Compromise Mitigation and Credential Handling (White-box/Gray-box)

- **Procedure (SAST):** Tools like **TruffleHog** are run against the cloud code repositories, configuration files, and mobile app source code (before compilation) to detect any hardcoded credentials, API keys, or private URLs that could be exploited if a developer machine were compromised via phishing.

  Procedure (DAST): Using a **Mobile Proxy** (**Burp Suite**), the tester simulates a successful compromise where a user session token or credential has been stolen.
  - **Goal:** Validate the effectiveness of **Multi-Factor Authentication (MFA)** enforcement, ensuring that the stolen credential/token cannot be used to gain unauthorized access without a second factor. Also, test if the application implements **session validity checks** (e.g., tying the session to a specific IP address or device identifier).

D. Mobile UI Spoofing Defense (White-box/SAST)

- **Procedure: Manual Code Review** of the mobile application UI rendering process.
- **Goal:** Ensure the app cannot be easily manipulated by external input (e.g., from a deep link or push notification) to display a **fake login prompt** or to accept user input into a malicious field, a technique known as "in-app phishing."

4. References

1. Chimuco, F. T., Sequeiros, J. B., Lopes, C. G., SimÃµes, T. M., Freire, M. M., & Inacio, P. R. (2023). Secure cloud-based mobile apps: attack taxonomy, requirements, mechanisms, tests and automation. *International Journal of Information Security*, 22(4), 833-867.
2. Abbas, S. G., Vaccari, I., Hussain, F., Zahid, S., Fayyaz, U. U., Shah, G. A., Bakhshi, T., & Cambiaso, E. (2021). Identifying and Mitigating Phishing Attack Threats in IoT Use Cases Using a Threat Modelling Approach. *Sensors*, 21(14), 4816. https://doi.org/10.3390/s21144816.
3. Wu, L., Du, X., & Wu, J. (2015). Effective defense schemes for phishing attacks on mobile computing platforms. *IEEE Transactions on Vehicular Technology*, 65(8), 6678-6691.
4. Jain, A. K., & Gupta, B. B. (2022). **A survey of phishing attack techniques, defence mechanisms and open research challenges**. *Enterprise Information Systems*, 16(4), 527-565.

Security Testing for Pharming Attacks

1. Overview

Pharming redirects legitimate user traffic to attacker-controlled sites (or services) by tampering with name resolution or local host mappings (DNS cache poisoning, router DNS setting changes, host-file modification) so victims unknowingly give credentials or the attacker injects malicious payloads. This is more scalable than classic phishing because it affects many users at once. In cloudâ€“mobileâ€“IoT ecosystems pharming can be especially damaging because: devices and mobile apps often rely on DNS, device provisioning and OTA endpoints; compromise of DNS or local resolver can redirect device/cloud traffic (telemetry, firmware updates, auth endpoints) to malicious servers. Testing must therefore cover DNS, routers, devices (mobile & IoT), and cloud backend behaviour and telemetry.

2. Security Test Approaches & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST	DNS cache poisoning & response tampering tests	Scapy, dnschef	Scapy	Both
Black-box	DAST	Router / DHCP/DNS config tamper simulation	Router firmware testbench / OpenWrt + scripting	OpenWrt	Both
Gray-box	DAST	Local host file & resolver poisoning tests	Metasploit modules, custom host-file scripts	Metasploit	Android (rooted), iOS (jailbroken), Desktop
Gray-box	SAST/DAST	IoT firmware review for hard-coded DNS / insecure resolver policy	Binwalk, Ghidra	Binwalk / Ghidra	IoT
Gray-box	SAST	Mobile app network validation & hostname verification	MobSF, Frida	MobSF / Frida	Android, iOS
White-box	SAST	Cloud API & TLS / certificate validation review	Semgrep, CodeQL, sslscan	Semgrep / CodeQL / sslscan	Cloud
Black-box	DAST	Network sniffing & DNS anomaly detection	Zeek, Wireshark, Elastic Stack	Zeek / Wireshark / Elastic Stack	Both
White-box	SAST	Resolver & DNSSEC/DoT/DoH configuration audit	dnssec-tools, doh-proxy, validators	dnssec-tools	Both
Gray-box	DAST	Pharming detection using ML on DNS logs	Python, ELK + ML modules	scikit-learn / scikit-learn	Cloud

2. Testbed & Step-by-step Testing Workflow

A. Testbed components (minimal)

- **Isolated lab network** (VLAN or air-gapped) with: DNS resolver(s), DHCP server, test router (OpenWrt), gateway that IoT devices use.
- **Test devices**: representative IoT devices (ESP32, Linux gateways), Android (emulator or rooted device), iOS test device (developer/jailbroken if needed), desktops.
- **Cloud test instance**: staging API endpoints, TLS certs, telemetry ingestion (Elastic / Splunk).
- **Tools**: Scapy/dnschef for DNS spoofing, Wireshark/Zeek for capture, Binwalk/Ghidra for firmware, MobSF/Frida for mobile app checks, Semgrep/CodeQL/sslscan for cloud code/TLS checks.

B. Steps (ordered, safe: run inside isolated test environment)

1. **Baseline collection**

   Capture normal DNS answers, device hostnames, endpoints, and TLS cert chains. Record device DNS settings (e.g., DHCP vs static) and any hard-coded resolver in firmware. 2. **Host-file pharming test (host-based)**

   On test mobile/desktop, modify the hosts file (or simulate via an MDM policy) to point a trusted hostname to a malicious staging IP. Verify the app & OS perform proper certificate/hostname verification and fail when server cert mismatch occurs. Use Frida/MobSF to instrument mobile app flows to see if hostname verification is enforced. 3. **Local router / DHCP DNS tampering**

   Configure OpenWrt test router to hand out malicious DNS server (via DHCP) or to rewrite DNS responses (dnsmasq rules). Observe whether devices accept new resolver and whether traffic goes to attacker staging server. Monitor cloud backend logs for unexpected client IPs/requests. 4. **DNS cache poisoning simulation**

Using dnschef/Scapy, craft spoofed DNS responses for frequently requested domains (test-only) and inject into resolver cache. Validate whether devices receive poisoned answers and whether detection (Zeek/IDS/ELK) flags unusual TTLs or multiple authoritative answers. 5. **Firmware/hard-coded DNS review**

Extract firmware images (Binwalk) from IoT devices; search for hard-coded hostnames or resolvers, backdoor update endpoints, or lack of TLS pinning. If firmware hard-codes a server IP, test what happens when that IP is hijacked. ([iosrjournals.org][3]) 6. **Cloud API resilience checks**

Use sslscan / semgrep to confirm TLS configuration (cert pinning, HSTS) and backend rejects requests when Host header mismatch or client cert absent. Try replaying captured requests with redirected Host header to staging server—verify server checks Host and rejects. 7. **Detection & ML experiments**

Feed DNS logs to Elastic/Zeek and run anomaly detection: unusual NXDOMAIN patterns, sudden changes in resolver usage, or TTL anomalies. Optionally run ML detection experiments per literature (ensemble learning on DNS features).

C. Test scenarios (examples)

- **Scenario 1 — Host-file pharming on mobile**: alter host file or emulator DNS config; verify mobile wallet or IoT provisioning app rejects mismatched certificates or prompts for re-auth.
- **Scenario 2 — Rogue DHCP/DNS from compromised gateway**: router gives out malicious DNS server; verify devices DNS queries are resolved to attacker server; observe cloud telemetry anomalies.
- **Scenario 3 — Firmware-level hardcoded DNS redirect**: change targeted IP behind hardcoded name; see whether devices accept update or telemetry from attacker server; verify firmware signing prevents malicious updates.
- **Scenario 4 — DNS cache poisoning (simulated)**: poison resolver cache with fake answers for a test domain and check how quickly systems detect & recover.

3. References

1. Azeez, N. A., Oladele, S. S., & Ologe, O. (2022). **Identification of pharming in communication networks using ensemble learning**. *Nigerian Journal of Technological Development*, 19(2), 172-180.
2. Chimuco, F.T., Sequeiros, J.B.F., Lopes, C.G. et al. **Secure cloud-based mobile apps: attack taxonomy, requirements, mechanisms, tests and automation**. *International. Journal of Information Security*. 22, 833â€“867.
3. Singh, N., Buyya, R., & Kim, H. (2024). **Securing cloud-based internet of things: challenges and mitigations**. *Sensors*, 25(1), 79.
4. Krishna, T. B. M., Praveen, S. P., Ahmed, S., & Srinivasu, P. N. (2023). **Software-driven secure framework for mobile healthcare applications in IoMT**.*Intelligent Decision Technologies*, 17(2), 377-393.

Security Testing for Spoofing Attacks

1. Overview

Spoofing attacks impersonate legitimate devices, services, or network elements (ARP/DNS/GPS/BLE/ID spoofing) to intercept, redirect or falsify communicationsâ€”test all network and identity touchpoints (devices, gateway, mobile, cloud).

2. Security Test Approach & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST	ARP spoofing / MITM	Bettercap	Bettercap	Both
Black-box	DAST	DNS spoofing / fake resolver	DNSChef	DNSChef	Both
Gray-box	DAST	Rogue access point / Wi-Fi spoof	Kismet / hostapd	Kismet	Both (Wi-Fi)
Black-box	DAST	BLE device / beacon spoofing	Ubertooth / nRF Sniffer	Ubertooth	Both (BLE)
White-box	SAST	Code review for identity validation & TLS checks	Semgrep / CodeQL	Semgrep , CodeQL	Cloud & mobile
Gray-box	DAST	Network anomaly detection (spoof indicators)	Zeek / Suricata	Zeek , Suricata	Both (network)

Gray-box	DAST	GPS / GNSS spoof simulation (lab)	GPS-SDR-SIM / GNSS-SDR (lab + shielded)	GPS-SDR-SIM , GNSS-SDR	Both

3. Minimal Testbed

- Isolated test VLAN or lab (no tests on production).
- Capture points: device side, gateway uplink, cloud ingress; Wi-Fi/BLE sniffers near devices.
- Test devices: representative IoT nodes, Android/iOS test devices, test Wi-Fi AP, BLE beacons, staging DNS resolver.
- Tools: bettercap, dnschef, kismet, ubertooth, Zeek, Semgrep/CodeQL.

4. Quick Test Steps

1. **Inventory & threat modeling** — list identity/auth points: MAC, IP, certificate, GPS, BLE IDs, cloud tokens.
2. **Baseline capture** — collect normal traffic & telemetry (pcap, logs).
3. **ARP/DNS spoof test** — in lab, run Bettercap (ARP) and DNSChef (fake resolver) to see if devices accept spoofed replies and whether TLS/HTTP host validation prevents redirect.
4. **Rogue AP & BLE tests** — deploy rogue AP and BLE beacon to test automatic joins and beacon acceptance; detect auto-connect and credential leakage.
5. **GNSS/GPS spoof (lab only)** — use GPS-SDR-SIM inside a shielded chamber to evaluate device tolerance to spoofed location/time. **Do not transmit RF publicly.**
6. **Code/config review** — check resolver policies, TLS pinning, certificate validation, and proper endpoint authentication. Use Semgrep/CodeQL to find insecure hostname checks or disabled cert validation.
7. **Detection validation** — ensure Zeek/Suricata and SIEM rules alert on ARP storms, unexpected DNS server change, duplicate MACs/IPs, sudden GPS/time jumps, or suspicious BLE activity.

5. References

1. Vajrobol, V., Saxena, G. J., Pundir, A., Singh, S., B. Gupta, B., Gaurav, A., & Rahaman, M. (2025). Identify spoofing attacks in Internet of Things (IoT) environments using machine learning algorithms. *Journal of High Speed Networks*, 31(1), 61-70.
2. Khan, F., Al-Atawi, A. A., Alomari, A., Alsirhani, A., Alshahrani, M. M., Khan, J., & Lee, Y. (2022). Development of a model for spoofing attacks in internet of things. Mathematics, 10(19), 3686.
3. Jinhua, G., & Kejian, X. (2013, January). ARP spoofing detection algorithm using ICMP protocol. In *2013 international conference on computer communication and informatics* (pp. 1-6). IEEE.
4. Grabski, S., & Szczypiorski, K. (2013, September). Network steganalysis: Detection of steganography in IEEE 802.11 wireless networks. In *2013 5th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)* (pp. 13-19). IEEE.
5. Yacchirena, A., Alulema, D., Aguilar, D., Morocho, D., Encalada, F., & Granizo, E. (2016, October). Analysis of attack and protection systems in Wi-Fi wireless networks under the Linux operating system. In *2016 IEEE International Conference on Automatica (ICA-ACCA)* (pp. 1-7). IEEE.

Security Testing Setup for Session Fixation Attacks

Overview

To evaluate the resilience of cloud-mobile-IoT applications against **Session Fixation Attacks**, where attackers force victims to use a known session ID, thereby gaining unauthorized access once authentication occurs.

2. Testing Environment Setup

- **Cloud Layer:** Deploy microservices on AWS, Azure, or GCP with authentication APIs (OAuth 2.0, JWT) using containerized environments (Docker/Kubernetes).
- **Mobile Layer:** Use Android and iOS apps connected to the cloud backend via RESTful or MQTT protocols. Implement both HTTP and HTTPS sessions for testing.
- **IoT Layer:** Include IoT gateways (e.g., Raspberry Pi, ESP32) communicating with the cloud through MQTT/TLS.
- **Attack Simulation:** Use proxy-based manipulation (Burp Suite, OWASP ZAP) to intercept and fixate session tokens pre- and post-authentication.
- **Detection & Monitoring:** Configure ELK Stack or Splunk to monitor unusual session reuse, duplicate session IDs, and concurrent user logins.
- **Mitigation Validation:** Implement and test secure cookie flags (`HttpOnly`, `Secure`), session regeneration after login, and token expiration policies.

3. Security Testing Approach & Tools

Test Approach	Analysis Type	Approach Name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST	Web Security Scanner / Pentesting	OWASP ZAP	OWASP ZAP	Both
Gray-box	DAST	Proxies / Session Manipulation	Burp Suite	Burp Suite	Both
White-box	SAST	Code Review / Token Handling Analysis	SonarQube	SonarQube	Both
Black-box	DAST	Vulnerability Scanning	Acunetix	Acunetix	Both
Gray-box	DAST	Network Packet Sniffing / Session Tracking	Wireshark	Wireshark	Both
White-box	SAST	Code Security Scanner	Checkmarx	Checkmarx	Both
Gray-box	DAST	API Security Testing	Postman + OWASP API Security Checklist	Postman	Both

---

4. Testing Phases

- **1. Reconnaissance:** Identify session management mechanisms across cloud, mobile, and IoT interfaces. |
- **2. Attack Simulation:** Use proxy tools to fix session IDs before and after authentication. Attempt to reuse sessions post-login.
- **3. Code Audit:** Analyze backend code for improper session lifecycle management and missing session regeneration calls.
- **4. Logging & Detection:** Use Splunk dashboards or ELK Stack to monitor for repeated session IDs or concurrent sessions.
- **5. Mitigation & Hardening** Implement session invalidation on logout and rotate session IDs post-authentication. Re-test using automated scripts.

---

6. References

1. Johns, M., Braun, B., Schrank, M., & Posegga, J. (2011, March). Reliable protection against session fixation attacks. In *Proceedings of the 2011 ACM Symposium on Applied Computing* (pp. 1531-1537).
2. Chimuco, F. T., Sequeiros, J. B., Lopes, C. G., SimÃµes, T. M., Freire, M. M., & Inacio, P. R. (2023). Secure cloud-based mobile apps: attack taxonomy, requirements, mechanisms, tests and automation. *International Journal of Information Security*, 22(4), 833-867.
3. LaBarge, R., & McGuire, T. (2013). Cloud penetration testing. *arXiv preprint arXiv*:1301.1912.
4. Kankhare, D. D., & Manjrekar, A. A. (2016, December). A cloud based system to sense security vulnerabilities of web application in open-source private cloud IAAS. In *2016 International Conference on Electrical, Electronics, Communication, Computer and Optimization Techniques (ICEECCOT)* (pp. 252-255). IEEE.
5. Casola, V., De Benedictis, A., Rak, M., & Villano, U. (2018, June). Towards automated penetration testing for cloud applications. In *2018 IEEE 27th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)* (pp. 24-29). IEEE.

# Security Testing Setup for Session Hijacking Attacks

1. Overview

To evaluate and enhance the resilience of **cloud-mobile-IoT systems** against **Session Hijacking Attacks**, where attackers intercept, steal, or predict valid session tokens to impersonate legitimate users.

---

2. Testing Environment Setup

- **Cloud Layer:** Cloud services (AWS, Azure, GCP) hosting APIs and authentication mechanisms (OAuth2, JWT, OpenID Connect). Enable HTTPS and API gateways with session management logs.
- **Mobile Layer:** Android and iOS applications communicating via RESTful APIs or MQTT over TLS. Integrate token-based authentication and session cookies.
- **IoT Layer:** IoT devices (Raspberry Pi, ESP8266, sensors) linked to cloud via MQTT/CoAP. Configure TLS for communication but allow temporary disabling to test hijacking feasibility.
- **Attack Simulation:** Simulate session hijacking using network sniffers, proxies, and replay scripts. Analyze token reuse, header manipulation, and TLS stripping.
- **Detection & Monitoring:** Monitor via ELK Stack or Splunk for duplicate session IDs, concurrent logins, or geolocation anomalies.
- **Mitigation Validation:** Test defense mechanisms: secure cookies, HSTS, session regeneration, token revocation, and behavioral anomaly detection.

---

3. Security Testing Tools Table (HTML)

Test Approach	Analysis Type	Approach Name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST	Web Security Scanner / Pentesting	OWASP ZAP	OWASP ZAP	Both
Gray-box	DAST	Proxies / Traffic Interception	Burp Suite	Burp Suite	Both
Black-box	DAST	Network Packet Sniffing / Replay	Wireshark	Wireshark	Both
Gray-box	DAST	Session Hijack Simulation	Ettercap	Ettercap	Both
White-box	SAST	Code Review / Token Handling	SonarQube	SonarQube	Both
Gray-box	DAST	Network and Web Pentesting	Metasploit Framework	Metasploit Framework	Both
White-box	SAST	API Security Review	Postman + OWASP API Security Checklist	Postman	Both

4. Testing Phases

- **1. Reconnaissance:** Identify session management mechanisms (cookies, tokens, headers). Map APIs and authentication flows.
- **2. Exploitation Simulation:** Use Wireshark, Burp Suite, or Ettercap to capture valid session tokens or cookies. Attempt replay and impersonation.
- **3. Code Review:** Use SonarQube or Checkmarx to detect insecure session token generation and lack of session invalidation after logout.
- **4. Mitigation Validation:** Implement secure session handling (HTTPS-only cookies, session rotation). Verify through repeated hijacking attempts.
- **5. Monitoring & Reporting:** Analyze captured traffic and system logs for duplicated tokens, user anomalies, and concurrent access patterns.

5. Key Metrics

- Session reuse rate
- Percentage of successful hijack attempts
- Token regeneration frequency
- Detection time of session anomalies
- API response integrity under attack conditions

6. References

1. Al-Ahmad, A. S., Kahtan, H., Hujainah, F., & Jalab, H. A. (2019). Systematic literature review on penetration testing for mobile cloud computing applications. *IEEE Access*, 7, 173524-173540.
2. Siddiqui, S., & Khan, T. A. (2019). Test patterns for cloud applications. *IEEE access*, 7, 147060-147080.
3. OS, J. N., & Bhanu, S. M. S. (2018). A survey on code injection attacks in mobile cloud computing environment. In *2018 8th International Conference on Cloud Computing, Data Science & Engineering (Confluence)* (pp. 1-6). IEEE.

Security Testing Setup for Access Point Hijacking Attacks

1. Overview

A detailed security testing setup for **Access Point (AP) Hijacking Attacks** requires simulating the malicious environment to test the defensive capabilities of client devices (Mobile and IoT) against traffic interception and manipulation. The focus is on validating the implementation of **Certificate Pinning** and **secure authentication protocols**.

2. Detailed Testing Setup and Procedures

The testing process is divided into simulating the attack and verifying the client defenses.

Rogue AP Simulation and Connection Testing (Black-box)

- **Procedure:** Set up a laptop running a Linux distribution like **Kali Linux** with an external Wi-Fi adapter. Use tools like `hostapd` or `airbase-ng` to create an **Evil Twin** AP with the exact same SSID and security settings as a trusted network (e.g., a corporate or home Wi-Fi).
- **Goal:** Observe whether the mobile/IoT client automatically connects to the rogue AP. Once connected, confirm if the client attempts to transmit any data (especially credentials or initial authentication tokens) before TLS handshaking is completed.

MITM and Certificate Pinning Validation (Gray-box/White-box)

- **Procedure:** Route all client traffic through a tool like **Burp Suite Professional** or **mitmproxy** running in transparent proxy mode. The proxy is configured to intercept and forge the SSL/TLS certificate used by the target cloud service (acting as the MITM).
- **Goal (Gray-box/DAST):** If the client is successfully connected to the rogue AP, attempt to access the cloud API. A secure client implementing **Certificate Pinning** should immediately **terminate the connection** and fail with a certificate error (e.g., `SSLHandshakeException`), preventing the MITM from capturing or altering data.
- **Goal (White-box/SAST):** Use **SonarQube** or manual review to inspect the source code. Verify that the client application network library is configured to perform explicit pinning (i.e., comparing the server public key or certificate hash against a hardcoded value) and that the exception handling for a pinning failure is secure (e.g., stops execution, does not fall back to unencrypted communication).

Protocol Downgrade and DNS Hijacking Testing (Black-box)

Procedure:
- **Downgrade:** Use **mitmproxy** to actively strip the HTTPS connection, forcing the client to communicate over unencrypted HTTP.
- **DNS Hijacking:** Configure the rogue AP DHCP server to issue a malicious DNS server address (using **DNSMasq**). This malicious DNS server is set to redirect the target domain (e.g., `api.cloudservice.com`) to the attacker server IP address.

- **Goal:** Ensure the client application **refuses to communicate** over the downgraded (HTTP) protocol. Verify that even when the client resolves the API domain to the attacker IP (via the malicious DNS), the subsequent secure connection attempt fails due to **Certificate Pinning** rejecting the attacker forged TLS certificate.

IoT Firmware and Gateway Vetting (White-box/DAST)

- **Procedure:** Directly scan the IP address of the target IoT gateway (AP) using a comprehensive **vulnerability scanner** like **Nessus**.
- **Goal:** Identify default credentials, unpatched firmware vulnerabilities, or open management ports that an attacker could exploit to gain administrative control over the legitimate AP. This models the initial compromise phase of a DNS Hijacking attack on the router itself.

---

3. Security Testing Approach & Tools

Test Approach	Analysis Type	Approach Name	Testing Tool	Hyperlink for Tool	Platform
Black-box	DAST	Rogue AP/Evil Twin Simulation	Kali Linux (using `airbase-ng` or `hostapd`)	Kali Linux	Both
Gray-box	DAST	MITM Traffic Interception/Validation	Burp Suite Professional	Burp Suite Pro	Both
White-box	SAST	Code Review (Certificate Pinning Enforcement)	SonarQube	SonarQube	Android, iOS
Black-box	DAST	Protocol Downgrade Testing (SSL Stripping)	mitmproxy	mitmproxy	Both
Black-box	DAST	DNS Hijacking Testing	DNSMasq	DNSMasq	Both
White-box	SAST/DAST	Firmware Vulnerability Scanning (IoT APs)	Nessus	Nessus	IoT Gateway/Router

Gray-box	DAST	Session Integrity Check (Post-MITM)	Wireshark	[Wireshark](#)	Both

4. References

1. LeBlanc, D., & Howard, M. (2002). *Writing Secure Code* (2nd ed.). *Microsoft Press*.
2. Xia, H., Brustoloni, J. (2004). Detecting and Blocking Unauthorized Access in Wi-Fi Networks. In: Mitrou, N., Kontovasilis, K., Rouskas, G.N., Iliadis, I., Merakos, L. (eds) Networking 2004. NETWORKING 2004. *Lecture Notes in Computer Science*, vol 3042. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-24693-0_65.
3. Golightly, L., Chang, V., Xu, Q.A. (2021). Towards Ethical Hackingâ€"The Performance of Hacking a Router. In: Jahankhani, H., Kendzierskyj, S., Akhgar, B. (eds) Information Security Technologies for Controlling Pandemics. Advanced Sciences and Technologies for Security Applications. Springer, Cham. https://doi.org/10.1007/978-3-030-72120-6_17.

Security Testing Setup for Cellular Rogue Base Station Attacks

1. Overview

Objective: Emulate, detect, measure impact of, and harden against rogue base station attacks that perform one or more of the following:

- Intercept or collect identifiers (IMSI/IMEI), downgrade security (force weaker ciphering), perform man-in-the-middle (MITM) for signaling or data, or perform denial-of-service (force detach/connection rejection). Test detection and mitigation across device (Android/iOS), network (local eNodeB/gNB), and cloud (backend services that rely on mobile telemetry).

2. Required Lab & Equipment

- **Shielded RF test environment** (Faraday cage) or written regulatory permission for test frequencies.
- **Software-Defined Radios (SDRs)** (for research-grade control): USRP family (Ettus), bladeRF, HackRF One (for low-power RX/TX in shielded lab).
- **Software base station stacks** (for setting up test BTS/eNodeB/gNB): srsRAN (srsLTE), YateBTS, OpenAirInterface, OpenBTS.
- **GSM/LTE analysis / sniffing tools**: gr-gsm / Airprobe (GSM), Wireshark (S1/RRC decode where possible), mobile logs (adb/logcat / iOS sysdiagnose).
- **IMSI-catcher & detector apps** (lab verification & detection): AIMSICD, SnoopSnitch, and other detector toolkits for ground truth comparison (note: these apps have limitations).

3. High-level Testing Categories

1. **Rogue BTS setup & configuration** — create controlled BTS (2G/3G/4G) and configure to accept test UEs.
2. **IMSI/IMEI harvesting & identity requests** — test whether your test stack can request subscriber identity, force null-ciphering, or request silent SMS.
3. **Security downgrades & cipher forcing** — test if the UE falls back to unencrypted or weaker crypto modes.
4. **MITM / data interception** — where lawful/contained: validate if signaling user plane can be intercepted in lab and whether applications leak sensitive data.
5. **Detection validation** — run IMSI-catcher detector apps and compare to ground truth from the test BTS (assess false negatives / bypass approaches).
6. **Cloud/service impact** — measure mobile app behavior, telemetry loss, backend connection anomalies, and forensic traces in cloud logs.
7. **Mitigation testing** — evaluate UE hardening, operator-side mitigations, and detection rules.

3. Stepwise Testing Playbook

Phase 0 — plan & authorize

- Written authorization (scope, time, frequencies), pre-registered test plan, emergency rollback.

Phase 1 — baseline & instrumentation

- Deploy a test base station (YateBTS / srsRAN) inside a Faraday cage; attach test UEs (spare phones/emulators). Collect baseline KPIs and logs (UE radio logs, Wireshark S1/RRC traces, SDR IQ).

Phase 2 — passive observation / recon

- Use gr-gsm, Airprobe, and SDR RX (RTL-SDR / HackRF) to passively observe neighbor cells and signaling. Record broadcasts (MIB/SIB/PBCH) for comparison.

Phase 3 — controlled rogue configuration (lab only)

- Configure test BTS to advertise stronger RSSI and accept registration from UEs; test identity request messages and null-ciphering scenarios. Log all signaling and compare to expected behaviour. **Never** do this outside shielded/test bands.

Phase 4 — active tests & attack variants (incremental)

- Basic IMSI collection (simulated), null ciphering, silent SMS, downgrade attempts.
- Protocol quirks: test disguised/mimicked real cell parameters to evaluate detector bypass (see White-Stingray evaluation).

Phase 5 — detection instrumentation & validation

- Run AIMSICD and SnoopSnitch on UEs; compare app alerts against ground-truth logs from the test BTS (assess detection latency and blindspots).

Phase 6 — cloud & service effects

- Measure mobile app retry behavior, backend session churn, and SIEM alerts. Verify whether cloud logs contain sufficient telemetry for attribution.

Phase 7 — report & remediation

- Produce clearly reproducible test cases (SDR IQ files, base station configs, device logs), prioritized findings, and recommended mitigations (UE hardening, carrier detection, operator configuration changes).

---

4. Security Testing Approaches & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Gray-box / Physical	DAST	Controlled rogue BTS / test eNodeB	srsRAN (srsLTE)	srsRAN	Both (Android, iOS; test UEs)
Gray-box / Physical	DAST	Open BTS (GSM) testbed	YateBTS / Yate	YateBTS	Both (legacy GSM & test UEs)
Black-box / Recon	DAST / Passive	GSM/LTE sniffing & broadcast analysis	gr-gsm / Airprobe	gr-gsm	Both (passive SDR receive)
Black-box / Physical	DAST / RF	SDR hardware for RX/TX (research & lab only)	USRP (Ettus) / bladeRF / HackRF	USRP / bladeRF / HackRF	Both
Gray-box / Detection	DAST / Endpoint	IMSI-catcher detector apps (compare GT)	AIMSICD / SnoopSnitch	AIMSICD / SnoopSnitch	Android
White-box / Simulation	SAST / Model	Cell/UE protocol simulation (no TX)	ns-3 / MATLAB / GNUradio	ns-3 / GNUradio	Both
Gray-box / Network	DAST / Traffic	Control channel & signaling analysis	Wireshark (LTE dissectors)	Wireshark	Both
Black-box / Research	DAST / RF	Research jamming / stealth techniques (lab only)	JamRF / GNUradio flows (research repos)	JamRF	Both
Gray-box / Detection	DAST / Analytics	Signal-feature & ML detection (spectrogram/IQ)	TensorFlow / PyTorch (custom ML)	TensorFlow / PyTorch	Both

---

5. References

1. Park, S., Shaik, A., Borgaonkar, R., Martin, A., & Seifert, J. P. (2017). {White-Stingray}: Evaluating {IMSI} Catchers Detection Applications. In *11th USENIX workshop on Offensive Technologies* (WOOT 17).
2. Tucker, T., Bennett, N., Kotuliak, M., Erni, S., Capkun, S., Butler, K., & Traynor, P. (2025). Detecting IMSI-Catchers by Characterizing Identity Exposing Messages in Cellular Traffic. In *Proceedings of the ISOC Networking and Distributed Systems Security (NDSS) Symposium*. San Diego, CA, USA.
3. Saedi, M., Moore, A., Perry, P., Shojafar, M., Ullah, H., Synnott, J., ... & Herwono, I. (2020, June). Generation of realistic signal strength measurements for a 5G Rogue Base Station attack scenario. In *2020 IEEE Conference on Communications and Network Security (CNS)* (pp. 1-7). IEEE.
4. Shaik, A., Borgaonkar, R., Park, S., & Seifert, J. P. (2018, June). On the impact of rogue base stations in 4g/lte self organizing networks. In *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks* (pp. 75-86).
5. Golde, N., Redon, K., & Borgaonkar, R. (2012, February). Weaponizing Femtocells: The Effect of Rogue Devices on Mobile Telecommunications. In *NDSS*.

Security Testing for GPS Spoofing Attacks

1. Overview

GPS spoofing attacks involve broadcasting counterfeit GPS signals to deceive receivers (e.g., IoT trackers, mobile apps, drones, and vehicles).

In a **cloud-mobile-IoT ecosystem**, GPS spoofing can:

- Mislead IoT location data and route tracking.
- Compromise mobile app geolocation features.
- Disrupt cloud-based logistics and fleet monitoring systems.
- Enable time synchronization attacks on cloud services relying on GNSS timestamps.

Testing Objectives

- Detect and evaluate vulnerabilities to GPS spoofing in IoT and mobile systems.
- Assess accuracy and tamper-resistance of GNSS modules.
- Test the robustness of cloud-based location validation and anomaly detection logic.
- Evaluate defenses like multi-sensor fusion (GPS + Wi-Fi + accelerometer).

---

2. Testing Environment Configuration

Cloud Layer: Simulated GPS data ingestion APIs and storage for IoT devices; anomaly detection logic deployed.
Mobile Layer: Android and iOS apps consuming GPS location services via SDKs (Google Maps, Core Location).
IoT Layer: GPS-enabled IoT devices (e.g., Raspberry Pi + GPS receiver) in a controlled testbed with signal simulator. **Spoofing Simulation:** GNSS simulators and SDR-based tools to transmit controlled fake GPS signals.
Monitoring: Use GPS integrity detection algorithms and signal analysis to monitor deviation from real coordinates.

---

3. Testing Workflow

1. **Threat Modeling:** Identify devices and systems relying on GPS for operation.
2. **Static Code Review (SAST):** Inspect location API usage, permission handling, and signal validation routines.
3. **Dynamic Testing (DAST):** Inject spoofed GPS signals to assess how apps and devices respond.
4. **Anomaly Detection:** Evaluate timestamp drifts, signal inconsistencies, or erratic path data.
5. **Cloud Verification:** Test back-end detection algorithms using spoofed data streams.
6. **Result Analysis:** Document drift tolerance, false positives, and mitigation performance.

---

4. Security Testing Approach & Tools

Test Approach	Analysis Type	Approach Name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST	GPS Spoofing Simulation	gps-sdr-sim	gps-sdr-sim	IoT
Gray-box	DAST	RF Signal Analysis	GNSS-SDR	GNSS-SDR	Both
Black-box	DAST	Wireless Signal Monitoring	HackRF One + SDRangel	HackRF One + SDRangel	IoT
White-box	SAST	Code Review for Location Validation	SonarQube	SonarQube	Both
Gray-box	DAST	Mobile App Reverse Engineering	MobSF	MobSF	Both
Black-box	DAST	Geolocation Integrity Testing	Scapy	Scapy	Both
Gray-box	DAST	Cloud Data Validation Testing	Postman	Postman	Both
Black-box	DAST	Location Spoof Detection Evaluation	GPS Test (Android)	GPS Test	Android

Gray-box	DAST	Network Packet Sniffing	Wireshark	Wireshark	Both
White-box	SAST	Static Analysis for Data Integrity	Checkmarx SAST	Checkmarx SAST	Both

5. References

1. Kerns, A. J., Shepard, D. P., Bhatti, J. A., & Humphreys, T. E. (2014). Unmanned aircraft capture and control via GPS spoofing. *Journal of Field Robotics*, 31(4), 617-636.
2. Tippenhauer, N. O., PÃ¶pper, C., Rasmussen, K. B., & Capkun, S. (2011). On the requirements for successful GPS spoofing attacks. *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS)*, 75-86.
3. Psiaki, M. L., & Humphreys, T. E. (2016). GNSS spoofing and detection. *Proceedings of the IEEE, 104*(6), 1258-1270.
4. Nighswander, T., Ledvina, B., Brumley, B. B., Brumley, D., & Clancy, T. C. (2012). GPS software attacks. *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, 450-461.
5. Jafarnia-Jahromi, A., Broumandan, A., Nielsen, J., & Lachapelle, G. (2012). GPS vulnerability to spoofing threats and a review of antispoofing techniques. *International Journal of Navigation and Observation*, 2012(1), 127072.
6. Alanda, A., Satria, D., Mooduto, H. A., & Kurniawan, B. (2020, May). Mobile application security penetration testing based on OWASP. In IOP Conference Series: Materials Science and Engineering (Vol. 846, No. 1, p. 012036). IOP Publishing.

Security Testing Setup for Tampering Attacks

1. Overview

Tampering involves unauthorized modification of code, firmware, configurations, or physical components. In cloud-mobile-IoT ecosystems, testing must span:

- **Mobile apps**: Detect code injection, runtime manipulation, and unauthorized access.
- **IoT devices**: Identify firmware tampering, debug port abuse, and physical bypass.
- **Cloud services**: Validate API integrity, configuration hardening, and deployment security.

Recommended Testing Layers

1. **Static Analysis (SAST)**: Review source code and binaries for vulnerabilities.
2. **Dynamic Analysis (DAST)**: Monitor runtime behavior and responses to inputs.
3. **Physical Inspection**: Evaluate tamper-resistance of hardware and enclosures.
4. **Network Monitoring**: Detect unauthorized traffic and protocol manipulation.
5. **Penetration Testing**: Simulate real-world attacks across all layers.

2. Security Testing Approach & Tools

Test Approach	Analysis Type	Approach Name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST	Pentesting	Burp Suite	Burp Suite	Both
Gray-box	SAST	Code Security Scanner	MobSF	MobSF	Both
White-box	SAST	Code Review	SonarQube	SonarQube	Both
Black-box	DAST	Web Security Scanner	OWASP ZAP	OWASP ZAP	Both
Gray-box	DAST	Network Packet Sniffing	Wireshark	Wireshark	Both
Black-box	DAST	Fuzzing	Peach Fuzzer	Peach Fuzzer	Both
White-box	Physical Review	Physical Security Measures Review	IoT Inspector	IoT Inspector	IoT

References

1. Sequeiros, J. B. F., Chimuco, F. T., Samaila, M. G., Freire, M. M., & InÃ¡cio, P. R. M. (2020). Attack and system modeling applied to IoT, cloud, and mobile ecosystems: Embedding security by design. *ACM Computing Surveys*, 53(2), Article 25. https://doi.org/10.1145/3376123
2. Chimuco, F. T., Sequeiros, J. B. F., Lopes, C. G., SimÃµes, T. M. C., Freire, M. M., & InÃ¡cio, P. R. M. (2023). Secure cloud-based mobile apps: Attack taxonomy, requirements, mechanisms, tests and automation. *International Journal of Information Security*, 22, 833â€“867. https://doi.org/10.1007/s10207-023-00669-z
3. Bella, G., Biondi, P., Bognanni, S., & Esposito, S. (2023). Petiot: Penetration testing the internet of things. *Internet of Things*, 22, 100707.
4. Yadav, G., Allakany, A., Kumar, V., Paul, K., & Okamura, K. (2019, July). Penetration testing framework for iot. In *2019 8th International Congress on Advanced Applied Informatics (IIAI-AAI)* (pp. 477-482). IEEE.

Security Testing for Node Tampering Attacks

1. Overview

In an IoT ecosystem (mobile clients, edge gateways, nodes/sensors, cloud services) a "node tampering" attack means that an attacker physically accesses, replaces, modifies, or reprograms an IoT node (sensor, actuator, gateway) so that it misbehaves: leaking keys, injecting false data, disrupting flows, or acting as a malicious pivot. For example: a sensor replaced with one that has back-door firmware; a gateway whose firmware is modified; a mobile client certified node tampered with. Research classifies this under the physical layer attacks for IoT. Because such attacks involve both hardware/firmware and their integration with mobile/cloud infrastructure, testing must encompass firmware/code review, physical security reviews, dynamic behavior monitoring, and cloud/mobile backend analytics.

---

2. High-level Testing Workflow / Setup

1. **Scope & threat modelling:**
- Inventory nodes: sensors, actuators, gateways, mobile client devices which may host or interface with nodes, cloud backend services.
- Identify assumed physical security, tamper-resistance, firmware update channels, cryptographic key storage, root of trust.
- Identify key data flows: node gateway ↔ cloud, node ↔ mobile client, mobile ↔ cloud.
- Define tampering vectors: full node replacement, firmware modification, memory extraction, side-channel attack, malicious re-programming.
2. **Baseline capture / instrumentation:**
- Record expected behaviour of nodes: firmware version, memory checksum/hash, boot up logs, cryptographic key presence, secure boot status, remote attestation if any.
- Mobile and cloud: log node registration/identity, firmware version check, heartbeat or telemetry data, anomaly metrics (data drift, behavior change).
3. **Static (SAST) & configuration review:**
- Review firmware or code for nodes (if available), look for secure boot, integrity checks, memory protection, key storage, absence of debug interfaces.
- Review mobile app/gateway code for verifying node identity, firmware version, attestation, remote enrollment.
- Review cloud backend processes that accept node registration, version checks, firmware update enforcement, revocation of compromised nodes.
4. **Dynamic testing (DAST) / Tampering simulation:**
- In a lab testbed: take a node, physically open it, swap memory modules, alter firmware, or simulate an attacker modifying keys or injecting malicious code.
- Deploy the modified node into the system (gateway/mobile/cloud) and observe: does the system accept version, does mobile app/gateway detect anomaly, does cloud backend flag the node?
- Use debugging tools/firmware tools to simulate memory extraction or debug interface abuse.
- On mobile/gateway side: attempt to inject a compromised node or emulate tampered behavior (e.g., send bogus sensor data) and check how the system handles it.
5. **Physical security & side-channel review:**
- Check enclosure tamper-evidence, tamper switches, sensors (shock, light, opening).
- Inspect memory/flash via tools (JTAG, BDM) to see if attacker can extract keys/hardware secrets.
- Use side-channel or fault injection tools to test memory/firmware integrity (optional advanced).
6. **Monitoring, telemetry & detection:**
- Monitor for anomalies: node firmware version change, boot counts, unusual behaviour (very high/low sensor values), mismatch between node identity and behavior.
- Inspect cloud logs for nodes with modified firmware, dropouts, inconsistent data, remote attestation failures.
- Setup alerts: node identity changed, firmware version unregistered, duplicate serial numbers, memory checksum mismatches.
7. **Reporting & remediation:**
- Produce findings: which nodes lacked tamper-detection, which firmware lacked integrity checks, mobile/gateway code lacked version enforcement, backend lacked revocation list.
- Provide mitigations: tamper-resistant hardware, secure boot, firmware integrity verification, remote attestation, regular inventory of node firmware versions, anomaly detection for node behavior, device key rotation, physical site inspection.
- Retest after fixes: use same tampered node and confirm detection or rejection.

---

3. Security Test Approaches & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
White-box	SAST	Firmware / node software code review	Ghidra / Binwalk	Ghidra	IoT node / gateway
Gray-box	SAST	Mobile/gateway app review for node enrolment & identity verification	MobSF	MobSF	Android, iOS
Gray-box	DAST	Simulated node replacement & behavior deviation testing	Custom test harness (Raspberry Pi/ESP32 nodes + tooling)	— (custom scripts)	IoT network
Black-box	DAST	Physical security / tamper-evidence testing	Tamper switch test rigs / torque testers / enclosure inspectors	— (hardware test equipment)	IoT node
White-box	SAST	Cloud backend code review for node firmware version control & revocation logic	CodeQL / Semgrep	CodeQL	Cloud backend
Gray-box	DAST	Telemetry anomaly detection for tampered nodes	Elastic Stack / Splunk	Elastic Stack	Both (cloud + mobile/gateway)
White-box	SAST	Remote attestation & hardware integrity check review	RADIS	RADIS	IoT node / gateway
Gray-box	DAST	Memory/key extraction via debug interfaces / side-channel simulation	JTAGulator / ChipWhisperer	JTAGulator	IoT node

4. Practical Testbed / Setup Checklist

- **Node test-fleet:** select representative IoT nodes (sensors/actuators) with known firmware, key storage, logging. Deploy them in lab environment, connected to gateway and cloud backend.
- **Mobile/gateway clients:** include mobile apps or gateway devices that enrol nodes, monitor node state, send telemetry to cloud.
- **Baseline capture:** record firmware version, boot logs, node IDs, sensor reading patterns, node registration events, mobile/gateway and cloud logs.

Tampering simulation: in lab:

Physically open node, swap memory card/flash, alter firmware, change keys, disable tamper switches; redeploy node and observe system.

- On node: simulate firmware downgrade, custom firmware that misreports sensor data or drops encryption.
- On mobile/gateway: attempt to register a node with altered identity or firmware version which is known â€œcompromisedâ€.

Observe system reaction:

Does mobile/gateway detect unexpected version or identity?

- Does cloud backend flag firmware version or node behaviour anomaly?
- Are sensor readings inconsistent with other nodes (e.g., drift, flat-line)?

Physical security test:

Inspect nodes deployed in field (if feasible) for tamper evidence: seals broken, casing open, unauthorized access.

- Use tamper-switch triggers to test if node logs â€œtamper eventâ€ or if system alerts.

Memory/key extraction lab test (optional advanced):

Using JTAGulator/ChipWhisperer to test whether secrets can be extracted from node hardware; simulate attacker re-use of extracted keys.

Detection & monitoring test:

Feed tampered node behaviour into system, verify logs, alerting, revocation path.

- Confirm that cloud backend prevents compromised node from further operation or quarantines it.

Remediation validation:

After implementing mitigations (secure boot, tamper switch, remote attestation, anomaly detection), retest with tampered node and confirm detection or rejection.

---

5. References

1. Sharma, G., Vidalis, S., Anand, N., Menon, C., & Kumar, S. (2021). A Survey on Layer-Wise Security Attacks in IoT: Attacks, Countermeasures, and Open-Issues. *Electronics*, 10(19), 2365.
2. Conti, M., Dushku, E., & Mancini, L. V. (2019, June). RADIS: Remote attestation of distributed IoT services. In *2019 Sixth International Conference on Software Defined Systems (SDS)* (pp. 25-32). IEEE.
3. Laguduva, V., Islam, S. A., Aakur, S., Katkoori, S., & Karam, R. (2019, July). Machine learning based iot edge node security attack and countermeasures. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* (pp. 670-675). IEEE.
4. Enow, M. A. (2018). An Effective Scheme to Detect and Prevent Tampering on the Physical Layer of WSN. International Journal of Sciences: Basic and Applied Research (IJSBAR), 39(2), 116-128.

Security Testing for Botnet attacks

1. Overview

Purpose

Emulate how an attacker would recruit and control devices (IoT and mobile) and abuse cloud services for C2, propagation, DDoS, data exfiltration and persistence. Use combined Black-box / Gray-box / White-box methods across layers: device firmware/app, network, cloud APIs, and backend infra.

Test environment & safety

- Isolate a lab environment (air-gapped or VLANs + NAT) that mirrors production: virtual cloud project(s) (AWS/Azure/GCP sandbox), mobile device farm (emulators + real devices), IoT device lab (real devices or firmware images), and an internal attacker host. Never scan or attack third-party networks without permission.
- Create representative assets: sample Android/iOS apps, firmware images, simulated home routers, cameras, and cloud APIs that your mobile/IoT fleet uses.
- Logging & monitoring: centralize packet capture (PCAP), system logs, cloud logs (CloudTrail/Activity Log), IDS/IPS sensors, and telemetry from devices.
- Baseline: run inventory + asset discovery (identify reachable devices and cloud endpoints) before active tests.
- Legal/ethical: signed authorization (scope, duration, toolset), and rollback/restore plan.

Test categories (what to test):

1. **Discovery & reconnaissance** (Shodan, Nmap, service banners).
2. **Vulnerability scanning & configuration** (Nessus / OpenVAS, Cloud Inspector).
3. **Mobile/firmware static & dynamic analysis** (SAST with SonarQube, MobSF, Apktool, Frida).
4. **Network behavior & C2 simulation** (Metasploit auxiliary modules, Scapy, Wireshark packet captures).
5. **Web/API / cloud DAST + proxy testing** (Burp Suite, OWASP ZAP).
6. **Fuzzing for protocol/firmware bugs** (Peach / fuzzers).
7. **Traffic manipulation / MitM / proxying** (Burp / ZAP / proxychains / mitmproxy).
8. **Persistence & lateral movement simulation** (Metasploit modules in controlled lab).
9. **Telemetry & detection testing** (validate detection signatures on IDS and cloud SIEM).
10. **Physical / supply-chain considerations** (review physical access, default credentials, bootloader locks).

---

2. Testing Set-up Details

1. **Prepare lab**

- Build a cloud sandbox (separate account/project); create sample backend APIs, message queues, and storage used by devices.
- Set up a device pool: several Android devices (real + emulator), sample iOS devices/emulators, IoT devices or firmware images.
- Central logging (ELK/Splunk) + packet capture point.

2. **Recon & inventory**

- Use **Shodan** and Nmap to enumerate exposed devices and services reachable from outside and internal network. (helps find default passwords, open telnet/ssh, or exposed management interfaces).

3. **Automated scanning & SAST**

- Run **Nessus/OpenVAS** on cloud hosts and device gateways.
- Run **SonarQube** on server and backend code.

4. **Mobile & firmware analysis**

- Decompile Android APKs with **Apktool**; run **MobSF** for static/dynamic mobile analysis; instrument suspicious calls with **Frida** to observe runtime behavior and potential botnet code (command parsing, C2 callbacks).

5. **Network & C2 emulation**

- Capture device traffic with **Wireshark**. Use **Scapy** to craft C2 packets / simulate botnet commands to see how devices respond. Use **Metasploit** modules in a fully controlled lab to simulate exploit and post-exploitation steps to test detection and containment.

6. **Web/API & Cloud tests**

- Use **Burp Suite** or **ZAP** to test backend APIs for authentication flaws, injection, or command execution that could be abused to orchestrate botnets (e.g., insecure firmware update endpoints).

7. **Fuzzing**

- Use a protocol/firmware fuzzer (Peach / equivalents) against custom services (device management, update protocols) to find crashes that could be exploited to install bot code.

8. **Detection validation**

- Replay found malicious traces against IDS, SIEM, and threat detection pipelines. Ensure cloud logs show useful telemetry (suspicious IPs, anomalous API calls).

9. **Report & fix**

- Map findings to CVEs, OWASP Mobile Top 10, and IoT security recommendations; provide prioritized remediation and detection tuning.

Short mapping: When to use which approach (quick guide)

- **Shodan / Nmap**: reconnaissance & exposed service discovery.
- **Nessus / OpenVAS**: broad vulnerability scanning of hosts and devices.
- **MobSF / Apktool / Frida**: mobile app reverse engineering and runtime instrumentation for mobile botnet components.
- **Burp / ZAP**: API/backend fuzzing and exploitation to see if cloud APIs can stage bot control.
- **Wireshark / Scapy**: observe or craft network traffic and emulate C2 to test device reactions.
- **Metasploit**: controlled exploit & post-exploit simulation (privilege escalation, persistence).

---

3. Security Testing Approach & Tools

Test approach	Analysis Type	Approach name	Testing Tool (link)	Platform
Black-box	DAST	Pentesting / Exploit simulation (C2, lateral movement)	Metasploit Framework	Both (server, network) / clients via payloads
Gray-box	DAST	Web/API proxying & manipulation	Burp Suite	Both (mobile app â‡„ backend APIs)
Black-box	DAST	Automated web app / API scanning	OWASP ZAP	Both (web / cloud APIs)
Gray-box / White-box	SAST / DAST	Mobile app static & dynamic analysis, malware scan	MobSF (Mobile Security Framework)	Android, iOS
Black-box	DAST / Passive	Network packet capture & protocol analysis	Wireshark	Both (network level)
Black-box	DAST	Host & port discovery, service fingerprinting	Nmap	Network / both

Gray-box / Dynamic	Dynamic instrumentation	Runtime instrumentation & function hooking	Frida	Android, iOS
White-box	SAST	Static code analysis (security / quality)	SonarQube	Both (server & app source)
Black-box	DAST / Vulnerability scanning	Full vulnerability assessment	Nessus (Tenable)	Network / Cloud / hosts
Black-box	DAST / Vulnerability scanning	Open source vulnerability management	OpenVAS / Greenbone	Network / hosts / IoT
Black-box / Recon	DAST / Recon	Internet-wide discovery of exposed IoT / cloud devices	Shodan	IoT / Cloud endpoints
Dynamic / Scripted	Network manipulation / active testing	Packet crafting & C2 simulation	Scapy	Network / Both
White-box / Static	Reverse engineering	APK reverse / resource inspection	Apktool	Android
Black-box / Gray-box	Fuzzing (protocols / firmware)	Protocol & firmware fuzzing	Peach Fuzzer (community)	IoT firmware, network services

3. References

1. Farina, P., Cambiaso, E., Papaleo, G., & Aiello, M. (2016). *Are mobile botnets a possible threat? The case of SlowBot Net*. Computers & Security, 58, 268-283.
2. Hamzenejadi, S., Ghazvini, M., & Hosseini, S. (2023). *Mobile botnet detection: a comprehensive survey*. International Journal of Information Security, 22(1), 137-175.
3. Abdullah, Z., Saudi, M. M., & Anuar, N. B. (2014, August). *Mobile botnet detection: Proof of concept*. In 2014 IEEE 5th control and system graduate research colloquium (pp. 257-262). IEEE.
4. Bernardeschi, C., Mercaldo, F., Nardone, V., & Santone, A. (2019). *Exploiting model checking for mobile botnet detection*. Procedia Computer Science, 159, 963-972.
5. Anwar, S., Zain, J. M., Inayat, Z., Haq, R. U., Karim, A., & Jabir, A. N. (2016, August). *A static approach towards mobile botnet detection*. In 2016 3rd International Conference on Electronic Design (ICED) (pp. 563-567). IEEE.
6. Mohammadi, H., & Hosseini, S. (2025). *Mobile botnet attacks detection using supervised learning algorithms*. Security and Privacy, 8(2), e494.

Security Testing for Malware-as-a-Service Attacks

1. Overview

Malware-as-a-Service (MaaS) commoditizes malware (ransomware, spyware, botnets, credential stealers, infostealers, etc.), letting less-skilled actors buy or rent turnkey attacks. That means mobile and IoT endpoints — often under-protected and widely deployed — become attractive delivery targets and footholds for cloud compromise or large-scale botnets. Detection, containment, and attribution across device-mobile app-cloud pipelines must therefore be tested end-to-end.

2. High-level Testing Workflow

1. **Scope & threat modelling** — enumerate device classes (constrained IoT, gateways, Android/iOS apps), cloud ingestion points, third-party services, and likely MaaS payload types (e.g., mobile spyware, IoT botnet binaries, cross-platform infostealers).
2. **Baseline & golden telemetry** — collect normal device, app, and cloud telemetry (process lists, network flows, logs, CPU/IO patterns, user behaviour) to detect subtle changes once malware is introduced.

3. **Static analysis (SAST)** — source & binary scanning for known indicators, insecure libraries, suspicious obfuscation, code signing checks, and manifest/permission abuses. Tools: MobSF, static analyzers, SCA tools.
4. **Dynamic analysis (DAST)** — execute suspected samples in sandboxes/containment (Cuckoo, Tamer or dedicated ARM IoT sandboxes) and observe persistence, network behavior, C2 patterns, and cloud-side effects.
5. **Network & telemetry fuzzing / simulation** — emulate command & control (C2), use service emulators (INetSim/FakeNet) to force different malware behaviors, and simulate large-scale infection to validate cloud detection/IOC pipelines.
6. **Mobile runtime instrumentation** — dynamic hooking, runtime API tracing and behavior inference on Android/iOS (Frida, MobSF dynamic, emulator + instrumentation).
7. **IoT firmware & binary analysis** — extract and analyze firmware images (Binwalk, Firmadyne), run in emulators, or instrument real devices in isolated lab.
8. **Adversary emulation / red-team** — use MaaS-like payloads in controlled fashion (benign-simulating payloads, telemetry-only agents, or offline samples) to validate detection, containment and incident responses.
9. **Reporting & remediation** — triage by impact category (data exfiltration, persistence, lateral movement, cloud compromise), remediate vulnerabilities, harden telemetry and patch/update cadence.

---

3. Security Testing Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
White-box	SAST	Static binary/source analysis	MobSF	MobSF	Android, iOS
Gray-box	DAST	Automated dynamic malware sandbox	Cuckoo Sandbox	Cuckoo Sandbox	Both (Windows/Linux/ARM targets via agents)
Gray-box	DAST	IoT / ARM sandbox & dynamic analysis	Tamer (IoT sandbox) / Firmadyne (emulation)	Tamer , Firmadyne	IoT (ARM/mips)
Black-box	DAST	Network traffic capture & C2 detection	Bro/Zeek, Suricata, Wireshark	Suricata	Both
Gray-box	DAST	Service emulation / Fake Internet	INetSim, FakeNet-NG	INetSim	Both
Gray-box	DAST	Runtime instrumentation / hooking	Frida	Frida	Android, iOS
White-box	SAST	Cloud function & CI/CD scanning	Snyk, Trivy, Semgrep	Snyk	Both (cloud)
Black-box	DAST	Sample collection & triage	VirusTotal, Hybrid-Analysis	VirusTotal	Both
Gray-box	SAST	Firmware unpack & binary analysis	Binwalk, radare2, Ghidra	Binwalk	IoT (firmware)
Gray-box	DAST	EDR validation & detection engineering	Atomic Red Team, Caldera	Atomic Red Team	Both
Black-box	DAST	Network sandbox / traffic replay	tcpreplay, Zeek + ELK stack	tcpreplay	Both

---

4. Practical Testing Set-up / Lab Checklist

- **Isolated network lab**: air-gapped or VLANed lab with internet simulation (INetSim / FakeNet) and strict egress filtering.
- **Virtualization**: ESXi / KVM hosts for Windows/Linux/Android VMs and snapshots; dedicated physical ARM boards (Raspberry Pi, test IoT devices) for real device behaviour.
- **Automated sandboxes**: Cuckoo Sandbox for Windows/Linux/Android; integrate with network capture (Zeek, Suricata) and centralized logging (ELK/Splunk).
- **Mobile devices**: instrumented Android (rooted/emulator) and iOS (jailbroken or diagnostic builds where legally allowed) with Frida and MobSF for dynamic analysis.
- **Firmware & IoT**: binwalk + Firmadyne for firmware extraction and emulation; hardware debug tools (UART/serial, JTAG) for deeper analysis.
- **Sample handling & enrichment**: VirusTotal / Hybrid-Analysis / private sample repositories; use cued triage to avoid executing live destructive payloads unintentionally.

- **Threat emulation**: Atomic Red Team, MITRE CALDERA, and scripted MaaS-like payloads (benign or telemetry-only) to validate detections.
- **Monitoring & telemetry**: centralize logs (ELK/Splunk), instrument cloud functions (CloudWatch/Stackdriver/Azure Monitor) for anomalous outbound connections, unusual CPU spikes, or sudden config changes.
- **Legal/safety**: legal approvals, data-handling procedures, and safe disposal for real malware samples. Use isolated lab with no uncontrolled internet egress.

---

5. Example Test Scenarios (practical)

- **Infostealer dropper via MaaS kit**: deploy a controlled dropper in a VM sandbox pointing to INetSim C2; validate that EDR/IDS triggers, cloud logs detect exfil attempts, and that telemetry contains IOCs for rapid triage.
- **Mobile spyware emulation**: instrument an Android app with Frida to observe API calls (telephony, contacts, geolocation); run in emulator, confirm detection by mobile threat hunting rules.
- **IoT botnet sample**: extract firmware, run in Firmadyne, execute bot behavior in sandbox to observe scanning/C2 beaconing; ensure cloud-side rate-limiting and blacklisting rules detect anomalous device traffic.
- **MaaS supply chain / CI pipeline test**: run SAST on cloud function repos and scanning on container images (Trivy/Snyk); attempt to inject a benign test payload via CI to validate preventive controls.

---

6. References

1. atsakis, C., Arroyo, D., Casino, F. (2025). The Malware as a Service Ecosystem. In: Gritzalis, D., Choo, KK.R., Patsakis, C. (eds) Malware. *Advances in Information Security*, vol 91. Springer, Cham. https://doi.org/10.1007/978-3-031-66245-4_16.
2. Yonamine, S., Taenaka, Y., & Kadobayashi, Y. (2022). Tamer: A Sandbox for Facilitating and Automating IoT Malware Analysis with Techniques to Elicit Malicious Behavior. In ICISSP (pp. 677-687).
3. Jamalpur, S., Navya, Y. S., Raja, P., Tagore, G., & Rao, G. R. K. (2018). Dynamic malware analysis using cuckoo sandbox. In *2018 Second international conference on inventive communication and computational technologies (ICICCT)* (pp. 1056-1060). IEEE.
4. Shahriar, H., Zhang, C., Talukder, M.A., Islam, S. (2021). Mobile Application Security Using Static and Dynamic Analysis. In: Maleh, Y., Shojafar, M., Alazab, M., Baddi, Y. (eds) Machine Intelligence and Big Data Analytics for Cybersecurity Applications. *Studies in Computational Intelligence*, vol 919. Springer, Cham. https://doi.org/10.1007/978-3-030-57024-8_20.

Security Testing Setup for Flooding / DDoS Attacks

---

1. Overview

Simulate and detect flooding / DDoS vectors (volumetric, protocol, application-layer) against cloud services, mobile backends and IoT gateways; validate monitoring, auto-scale and mitigation controls without harming production.

---

2. Security Testing Approach & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST	Application-layer HTTP flood (legitimate-looking requests)	wrk / Locust / Tsung	wrk	Both
Black-box	DAST	Low-and-slow / connection exhaustion	slowhttptest / Slowloris (testing mode)	slowhttptest	Both
Black-box	DAST	Protocol & SYN/UDP/TCP floods	hping3 / mausezahn / Scapy	hping3	Both (network)
Gray-box	DAST	IoT device flood / simulated botnet behavior	custom device-simulators (Python/async) / MQTT flood scripts	— (in-house scripts)	IoT
Gray-box	SAST/DAST	Load & infrastructure resilience testing (scale, autoscaling)	k6 / JMeter / chaos-engineering tools (Chaos Mesh)	k6	Cloud
White-box	DAST	Network & packet inspection / detection validation	Zeek / Suricata + ELK / Grafana	Zeek	Both

Gray-box	DAST	Upstream scrubbing & CDN validation	Test with provider-supplied DDoS test services (requires approval)	— (contact CDN/DDoS provider)	Cloud
White-box	SAST	Code review for expensive operations & rate-limit gaps	Semgrep / CodeQL / manual review	Semgrep	Cloud & mobile

3. Minimal Testbed & Safety Rules

- **Staging-only:** run all active flooding tests in isolated staging environments, not production, unless you have explicit written permission from cloud provider and stakeholders.
- **Traffic control & kill-switches:** have bandwidth caps, rate-limit guards, and a manual/automated kill switch to stop tests.
- **Provider coordination:** for cloud workloads, coordinate with the cloud/CDN/DDoS provider (AWS/Azure/GCP/Cloudflare etc.) before any volumetric tests — they often require notification/permission.
- **Monitoring:** centralize metrics (CloudWatch/Prometheus), network telemetry, IDS logs (Zeek/Suricata), and application logs to measure impact.
- **Snapshots/backups:** prepare snapshots and autoscaling rollback policies.
- **Ethics & compliance:** never generate unwarranted traffic that can affect third parties or upstream networks.

4. Short Step-by-step Test Workflow

1. **Define scope & get approvals** — targeted endpoints, allowed traffic types, max request rates, time windows, provider approvals.
2. **Baseline & instrumentation** — capture baseline latency, CPU/memory, connection counts, request rates; enable detailed logging and metrics dashboards.
3. **Micro-load (application) tests** — run `wrk` or `k6` with realistic user patterns (ramped concurrency) to validate app-level thresholds and autoscaling behaviour. Monitor error rates, latency, DB load.
4. **Slow / resource exhaustion tests** — run `slowhttptest` to test socket exhaustion and server worker limits; verify webserver config (worker limits, timeouts, keepalive) defends.
5. **Protocol flood experiments (lab net only)** — small-scale SYN/UDP/TCP bursts via `hping3` / `mausezahn` to verify network stack (SYN cookies, conn table) and firewall behaviour. Keep rate low and controlled.
6. **IoT botnet simulation** — use scripted IoT device-emulators to generate many simultaneous lightweight connections (MQTT publishes) to the broker to measure broker/edge resilience and detection.
7. **Autoscale / CDN / upstream validation** — verify autoscaling triggers correctly, test WAF/ratelimit rules, and coordinate with CDN/scrubbing provider to ensure protected traffic is routed to scrubbing centers.
8. **Detection & alert validation** — verify Zeek/Suricata and SIEM rules alert on volumetric spikes, connection anomalies, SYN floods, abnormal request headers or repeated malformed requests.
9. **Mitigation tests** — validate success of mitigations: WAF rules block malicious patterns, rate-limits throttle, SYN cookies protect TCP stack, CDN caches absorb traffic, and scrubbing mitigates volumetrics.
10. **Report & hardening** — produce findings: thresholds to tune, WAF rules to add, autoscale thresholds, network ACLs, and runbook for incident response.

5. References

1. Mirkovic, J., & Reiher, P. (2004). A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2), 39-53.
2. Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., ... & Zhou, Y. (2017). Understanding the mirai botnet. In *26th USENIX security symposium (USENIX Security 17)* (pp. 1093-1110).
3. Agrawal, N., & Tapaswi, S. (2019). Defense mechanisms against DDoS attacks in a cloud computing environment: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 21(4), 3769-3795.

Security Testing for Bypassing Physical Security Attack

1. Overview

A Bypassing Physical Security Attack involves gaining unauthorized access to restricted areas or assets by exploiting weaknesses in physical barriers or protocols. Security testing is vital to uncover these vulnerabilities and prevent real-world breaches that digital defenses alone cannot stop.

Why Is Security Testing for Physical Bypass Attacks Important?

1. Reveals Overlooked Vulnerabilities;
2. Protects Critical Assets;
3. Supports Regulatory Compliance;

4. Improves Incident Response Planning;
5. Enhances Overall Security Posture.

---

2. Lab & Safety Setup

1. **Authorization & ROE**: signed Rules of Engagement (scope, dates, allowed tools, safety contacts, rollback).
2. **Isolated environment**: VLAN / air-gapped physical test zone and separate cloud test project/account; do not run RF transmission tests outside allowed frequency/power limits and local regulatory constraints.
3. **Representative assets**: test badges/cards, badge readers, IoT devices, mobile devices, endpoints, cloud test APIs and backend instances that mirror production (but are not production).
4. **Instrumentation**: CCTV (or simulated camera inputs), PCAP capture points (mirror/SPAN), endpoint logging agents, and cloud audit logs enabled. Use packet capture tools to record attack behavior for detection validation.
5. **Safety & PPE**: ESD strap, insulated tools when opening hardware, documented reboot/restore procedures, firmware backups.
6. **Legal**: ensure compliance with radio transmission laws when using SDR (HackRF/Flipper transmissions) and do not replay signals in public spaces.

---

3. High-level testing categories (what to test)

- Recon & mapping: badge readers, cameras, external IoT endpoints, network hosts.
- RFID/NFC: sniff, read, emulate, clone, relay, and replay tests.
- RF/Sub-GHz: sniff and test replay/rolling-code vulnerabilities (SDR).
- USB / HID: BadUSB, Rubber Ducky, Bash Bunny payload tests (keystroke injection, network emulation).
- Hardware ports & debug interfaces: UART, JTAG, SPI — attempt controlled firmware reads / serial consoles.
- Tampering & supply-chain: open enclosures, inspect for debug pads and unprotected storage.
- Post-physical compromise: use any recovered secrets to access cloud APIs, mobile apps, or to persist/exfiltrate data.
- Detection & telemetry validation: replay captured traces to SIEM/IDS and adjust detections.

---

4. Stepwise Testing Playbook (practical)

1. **Recon & mapping**
- Visual map of readers, cameras, and ports; network scanning for accessible IoT devices (Shodan + Nmap).
2. **RFID / NFC tests (lab only)**
- Passive sniff with Proxmark3; attempt read (identify tag type), emulate with ChameleonMini, and test clone/replay. Log all reader responses and timestamps to detect replay resilience.
3. **SDR / sub-GHz & RF tests**
- Use HackRF to capture candidate signals (low power, legal frequencies). In an isolated lab only, attempt controlled replay to test whether access systems accept replayed tokens or rolling code weaknesses.
4. **Bluetooth tests**
- Capture BLE advertising/packets with Ubertooth or BLE dongles; test for open pairing or insecure GATT endpoints.
5. **USB / HID attacks**
- In locked-lab endpoints, execute staged Rubber Ducky payloads (keystroke injection) and Bash Bunny multi-vector payloads to measure time-to-compromise, persistence, and telemetry. Record detection events (EDR, AV, SIEM).
6. **Hardware debug & firmware**
- Power down device, inspect PCB for UART/JTAG pads, attach serial adapter, capture boot messages; where permitted, dump firmware for offline analysis and fuzzing. (Follow ESD & warranty guidance.)
7. **Protocol & firmware fuzzing**
- Fuzz device update endpoints and management protocol (Peach / custom AFL harness) to discover crashes enabling code injection.
8. **Post-physical compromise escalation**
- If credentials or shell are obtained through physical tests, attempt to access cloud APIs / backend as scoped and log all activity. Use Metasploit for controlled exploitation only in lab.
9. **Detection & reporting**
- Replay PCAPs into IDS/SIEM and verify detection coverage; produce prioritized remediation (crypto badges, signed firmware, USB device control, disable unused debug ports, anti-relay hardware where possible).

---

5. Security Testing Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform

Black-box / Physical	DAST / Passive	RFID / NFC reading, cloning & emulation	Proxmark3	Proxmark3	Hardware (RFID/NFC) — affects IoT & mobile badges
Black-box / Physical	DAST / Multi-purpose	Multi-tool RF & peripheral testing	Flipper Zero	Flipper Zero	RFID, sub-GHz, IR, GPIO — IoT & access controls
Black-box / Physical	DAST / Emulation	NFC card emulator (badge emulation)	ChameleonMini (RevG)	ChameleonMini	RFID/NFC badges (IoT access / doors)
Gray-box / Wireless	DAST / RF analysis	Software-Defined Radio (sniff & replay)	HackRF One	HackRF One	Sub-GHz / ISM / custom protocols — IoT radios
Black-box / Wireless	DAST / Bluetooth	Bluetooth/BLE sniffing & replay	Ubertooth / BLE tools	Ubertooth / BLE tools	Bluetooth / BLE devices (IoT & mobile)
Black-box / Physical	DAST / Host compromise	Keystroke injection / BadUSB	USB Rubber Ducky	USB Rubber Ducky	Windows/macOS/Linux endpoints (via USB)
Black-box / Physical	DAST / Multi-vector USB	Complex USB multi-vector payloads	Bash Bunny	Bash Bunny	Endpoint compromise via USB/HID/network emulation
Gray-box / Hardware	SAST / DAST	UART / JTAG / serial debug access	Serial adapters / logic analysers	Serial adapters	IoT hardware (firmware access)
Gray-box / Network	DAST / Passive	Packet capture & protocol analysis	Wireshark	Wireshark	Network / IoT / mobile traffic
Black-box / Recon	DAST / Recon	Public exposure discovery (cameras, IoT endpoints)	Shodan	Shodan	Internet-accessible IoT & cloud endpoints
Black-box / Recon	DAST	Network & host discovery	Nmap	Nmap	Network devices, gateways, cloud hosts
White-box / Post-compromise	DAST / Exploit simulation	Post-physical compromise exploitation & pivoting	Metasploit Framework	Metasploit Framework	Server / host / network post-compromise (cloud & local)
Gray-box / API	DAST	API/backend tests (credential reuse)	Burp Suite	Burp Suite	Both
Gray-box / Fuzzing	DAST	Protocol & firmware fuzzing	Peach Fuzzer / AFL	Peach Fuzzer	IoT firmware, device protocols
Dynamic / Scripted	DAST / Packet crafting	Packet crafting & replay	Scapy	Scapy	Network / RF / IoT protocols

---

6. References

1. Huang, W., Zhang, Y., & Feng, Y. (2020). ACD: An adaptable approach for RFID cloning attack detection. *Sensors*, 20(8), 2378. https://doi.org/10.3390/s20082378
2. Hancke, G. P. (2006, May). Practical attacks on proximity identification systems. In *2006 IEEE Symposium on Security and Privacy (S&P06)* (pp. 6-pp). IEEE.
3. Koffi, K. A., Smiliotopoulos, C., Kolias, C., & Kambourakis, G. (2024). To (US) Be or Not to (US) Be: Discovering Malicious USB Peripherals through Neural Network-Driven Power Analysis. *Electronics*, 13(11), 2117.
4. MuÃ±oz, A., FernÃ¡ndez-Gago, C., & LÃ³pez-Villa, R. (2023). A test environment for wireless hacking in domestic IoT scenarios. *Mobile Networks and Applications*, 28(4), 1255-1264.
5. Yang, X., Shu, L., Liu, Y., Hancke, G. P., Ferrag, M. A., & Huang, K. (2022). Physical security and safety of IoT equipment: A survey of recent advances and opportunities. *IEEE Transactions on Industrial Informatics*, 18(7), 4319-4330.

Security Testing for Physical Theft Attacks

1. Overview

In IoT-mobile-cloud ecosystems, physical theft of devices (sensors, gateways, mobile devices) or mobile endpoints allows attackers to bypass many software protections: they may extract keys, remove storage media, implant malware, clone devices, or misuse devices as trusted network endpoints. Research reviews of IoT threat models highlight physical attacks and theft as major vectors. Testing the resilience of devices, encryption at rest, secure boot, tamper-evidence, remote wipe, and backend recognition of stolen nodes is therefore critical.

2. Security Test Approaches & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
White-box	SAST	Firmware code review for secure boot / key storage	Ghidra / Binwalk	Ghidra	IoT node / gateway
Gray-box	DAST	Mobile device theft simulation & remote wipe verification	Mobile Device Management (MDM) test suite / custom script	— Custom test	Android, iOS
Black-box	DAST	Physical device removal & insertion in network (stolen node test)	Test fleet of devices + network monitoring (Zeek/Wireshark)	Zeek	IoT node / gateway
Gray-box	SAST	Mobile app review for stolen device authentication bypass	MobSF / Frida	MobSF	Android, iOS
White-box	SAST	Cloud backend review for stolen-device detection & revocation logic	Semgrep / CodeQL	Semgrep	Cloud
Gray-box	DAST	Key extraction / memory dump from stolen device	ChipWhisperer / JTAGulator	ChipWhisperer	IoT node / mobile
Black-box	DAST	Physical tamper-evidence test (tamper seals, enclosure breach)	Visual inspection kit / tamper-switch test rig	— hardware test rig	IoT node / gateway
Gray-box	DAST	Network traffic monitoring for stolen-node behaviour (new MAC/IP)	Elastic Stack / Splunk	Elastic Stack	Cloud/mobile/IoT

3. Testing Setup & Workflow

Components & Setup

- **Test device pool:** IoT sensor/gateway units identical to production devices; mobile smartphones/tablets used by users; cloud backend simulation environment.
- **Physical theft scenarios:** designate a set of devices as stolen — these will be removed from lab, tampered or replaced, then reintroduced.
- **Monitoring/logging:** network monitoring (Zeek/Wireshark) at gateway; cloud logging of device IDs, firmware version, last-seen timestamp, telemetry.
- **Firmware & mobile code review:** obtain firmware images (if available) for key extraction & secure boot checks; mobile apps for remote wipe and device-loss handling.
- **Revocation & detection logic:** cloud backend should have logic to revoke lost/stolen device IDs, monitor abnormal behaviour (new IPs, duplicate IDs, out-ofâ€region connections).
- **Physical test rig:** use tamper-switch test rig, visual inspection kits, JTAG/USB debug port exposure test.

Step-by-step Workflow

1. **Baseline capture:** deploy all devices in lab; take inventory (device ID, firmware version, MAC/IP, geolocation if applicable) and capture normal telemetry & network flows.
2. **Firmware/mobile review:** run Ghidra/Binwalk on node firmware; check for secure boot, key storage, debug ports. Review mobile app for remote wipe, device registration, lost-device handling.
3. **Stolen device simulation:**

- Remove one IoT node from lab and later re-connect it (either tampered or unchanged) — monitor how the backend handles it.
- On mobile device branch: simulate lost/stolen handset; test remote wipe, account logout, device block. 4. **Key extraction test:** With stolen node in lab, attempt JTAG/USB debug access to extract keys or firmware. Use ChipWhisperer/JTAGulator in controlled environment. 5. **Reintroduction & network test:** Reconnect the stolen/tampered device to network. Monitor for abnormal behaviour (new IP, duplicate ID, unexpected traffic patterns) and ensure backend revokes/quarantines device. 6. **Tamper-evidence test:** Open device enclosure, trigger tamper switch or remove internal components, re-connect; verify if device logs a tamper event or locked. 7. **Mobile lost device test:** Simulate user lost phone; ensure remote wipe works, apps do not auto-log back in, MFA required, device cannot access IoT/gateway/cloud. 8. **Reporting & remediation:** Document which devices lacked tamper-evidence, which mobile apps lacked remote-wipe or lockout logic, and which backend lacked revocation capability. Provide remedial recommendations: asset tagging, secure boot, encrypted storage, remote wipe, tamper monitoring, endpoint inventory.

---

4. References

1. Mehari Msgna, A. (2022). Anatomy of attacks on IoT systems: review of attacks, impacts and countermeasures. *Journal of Surveillance, Security and Safety*, 3, 150-173. https://doi.org/10.20517/jsss.2022.07.
2. Kumar Sikder, A., Petracca, G., Aksu, H., Jaeger, T., & Uluagac, A. S. (2018). A Survey on Sensor-based Threats to Internet-of-Things (IoT) Devices and Applications. *arXiv preprint arXiv:1802.02041*.
3. Islam, M. R., & Aktheruzzaman, K. M. (2020). An analysis of cybersecurity attacks against internet of things and security solutions. *Journal of Computer and Communications*, 8(04), 11.

Security Testing for VM Migration Attacks

1. Overview

A VM Migration Attack targets the process of moving virtual machines (VMs) between physical hosts, exploiting vulnerabilities to compromise data, control, or system stability.

Addressing VM Migration Attack security testing is crucial for maintaining the integrity, confidentiality, and availability of cloud and virtualized environments.

2. Security Testing Approaches & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Black-box / Gray-box	DAST	Live migration protocol fuzzing (network & RPC)	FitM / custom middlebox fuzzer	FitM / network fuzzers (examples)	Both
Black-box	DAST	Migration stream tampering / MitM proxy	mitmproxy, socat, tcpreplay	mitmproxy	Both
Gray-box	DAST	Post-copy/resume stalling & resource exhaustion	custom kernel/guest workloads, stress-ng	stress-ng	Both
Gray-box	DAST / Forensics	Memory/state tampering & snapshot manipulation	QEMU snapshot tools, libvirt, vmstate-tooling	QEMU migration docs	Both
White-box	SAST	Source review for migration code paths (libvirt/QEMU)	CodeQL, SonarQube, Coverity	CodeQL	Both
Gray-box	DAST / Binary	Fuzzing of migration RPCs / RPC handlers	AFL/LibAFL harnesses targeting QEMU migration streams	AFL	Both
Black-box	DAST / Network	Network tracing / packet capture during migration	Wireshark, tcpdump, Zeek	Wireshark	Both

Gray-box	DAST / Runtime	Runtime integrity & tamper-detection testing	LibVMI, Volatility	LibVMI	Both
Black-box	DAST	Cloud orchestration / API abuse during migration	OpenStack client, AWS CLI, Terraform + Burp Suite	OpenStack	Both
White-box	SAST	Configuration & policy review (encryption/auth)	Nessus / OpenVAS, CIS Benchmarks	OpenVAS	Both

3. Brief Testing Set-up

Isolated testbed — dedicate physical hosts for source/target hypervisors (KVM/QEMU, Xen, VMware) and an isolated network to avoid impacting production. Use libvirt to orchestrate migration flows. (QEMU/KVM docs are a practical reference).

Build baseline VM images — minimal Linux/Windows guests and representative Android emulator images (QEMU-based). For IoT: use lightweight guests or container-based device images (CRIU for container migration experiments).

Enable migration modes — test pre-copy, post-copy, and block/live migration modes supported by your stack (QEMU has explicit migration states and troubleshooting tips). Record migration control channels (TCP ports / unix sockets).

Network & MitM testing — place a MitM between source and target migration endpoints and attempt: packet corruption, selective drop, replay, injection, or protocol field fuzzing. Use mitmproxy/socat/tcpreplay to manipulate streams and observe failure modes. (Empirical attacks historically exploited unsecured migration channels).

Stream / state fuzzing — fuzz the migration RPC/state machine (e.g., QEMU migration stream) with AFL/LibAFL harnesses or fuzzer-in-the-middle setups that mutate live migration messages; capture crashes and incomplete restores.

Resource exhaustion & stalling — run attacker workloads in guest (e.g., memory churn, hugedirty pages) to attempt migration stalling or amplification attacks (stalled post-copy or forced repeated rounds). Monitor migration progress and host resource consumption; reproduce KVM stalling attacks for research validation.

Snapshot & snapshot-manipulation attacks — create and modify VM snapshots / disk/image state to simulate tampering during offline/online migration; attempt rollback/resume tampering to observe integrity failures. Use QEMU snapshot tooling and libvirt APIs.

Forensic & detection instrumentation — attach LibVMI/Volatility on the target host to inspect guest memory/CPU state after migration; use logs and packet captures to triage whether tampering produced code execution or data corruption.

SAST & config review — review migration code paths (libvirt, QEMU, VMware agents) for insecure defaults, missing encryption/auth, improper length checks, and deserialization issues. Run CodeQL/SonarQube and check cloud orchestration APIs for misconfigurations.

Reproduce & PoC — for every crash/state inconsistency, collect migration logs, full VM snapshots, pcap, guest traces, and step-by-step reproduction on a clean environment. Prepare responsible disclosure or mitigation recommendations.

4. References

1. Oberheide, J., Cooke, E., & Jahanian, F. (2008, February). Empirical exploitation of live virtual machine migration. In *Proceeding of Black Hat DC convention* (pp. 1-6). Citeseer: The Pennsylvania State University.
2. Atya, A., Aqil, A., Khalil, K., Qian, Z., Krishnamurthy, S. V., & La Porta, T. F. (2017). Stalling live migrations on the cloud. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*.
3. Clark, C. et al. (2005). Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2* (pp. 273-286).
4. Casola, V., De Benedictis, A., Rak, M., & Villano, U. (2018, June). Towards automated penetration testing for cloud applications. In 2018 IEEE 27th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE) (pp. 24-29). IEEE.
5. Rakotondravony, N., Taubmann, B., Mandarawi, W. et al. Classifying malware attacks in IaaS cloud environments. J Cloud Comp 6, 26 (2017). https://doi.org/10.1186/s13677-017-0098-8

Security Testing Setup for Side-Channel Attacks

1. Overview

Goal: Measure and mitigate leakage (power, EM, timing, cache) that can reveal secrets (cryptographic keys, credentials) across cloud, mobile and IoT components.

Scope: cloud VMs/HSMs, mobile apps (Android/iOS), IoT edge devices (Raspberry Pi/ESP32), network links.

2. Key Test Categories

- **Timing analysis** — check for variable-time crypto or API calls.
- **Cache attacks** — flush+reload / prime+probe on shared hosts.
- **Power & EM** — measure device emissions to recover keys (edge devices, smartcards).
- **Network timing correlation** — infer operations via request/response timing.
- **Code review** — find non-constant time code, unsafe libraries, or shared-resource patterns.

3. Compact Testing Tools

Test Approach	Analysis Type	Approach Name	Testing Tool	Tool Hyperlink	Platform
Gray-box	DAST	Physical Security Measures Review	ChipWhisperer	ChipWhisperer	Both
White-box	SAST	Code Review / Timing Analysis	CacheAudit	CacheAudit	Both
Black-box	DAST	Fuzzing / Input Timing Testing	American Fuzzy Lop (AFL)	AFL	Both
Gray-box	DAST	Electromagnetic and Power Analysis	Riscure Inspector	Riscure Inspector	Both
White-box	SAST	Code Security Scanner / Constant-Time Verification	ctgrind (Valgrind plugin)	ctgrind	Both
Gray-box	DAST	Network Packet Sniffing / Timing Correlation	Wireshark	Wireshark	Both
White-box	SAST	Code Review for Cache Leakage	LLVM-based DataFlow Sanitizer	LLVM-based DataFlow	Both

4. Minimal Testbed & Quick Steps

1. **Isolate lab** (air-gapped or VLAN). snapshot/backup targets.
2. **Baseline**: capture normal timing/power/EM traces.
3. **Attack**: run cache/timing tests and power/EM captures (ChipWhisperer) against crypto operations.
4. **Code review**: search for non-constant time ops, secret-dependent branching. Use ctgrind / static checks.
5. **Detect**: monitor `perf` / counters, timing variance, unusual cache misses; alert via SIEM.
6. **Mitigate & retest**: apply constant-time libs, masking, noise, HSMs/secure enclaves â€” then repeat tests.

5. References

1. Mangard, S., Oswald, E., & Popp, T. (2007). Power analysis attacks: Revealing the secrets of smart cards. Boston, MA: *Springer US*.
2. Yarom, Y., & Falkner, K. (2014). {FLUSH+ RELOAD}: A high resolution, low noise, l3 cache {Side-Channel} attack. In *23rd USENIX security symposium (USENIX security 14)* (pp. 719-732).
3. Genkin, D., Shamir, A., Tromer, E. (2014). RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis. In: Garay, J.A., Gennaro, R. (eds) Advances in Cryptology â€“ CRYPTO 2014. CRYPTO 2014. *Lecture Notes in Computer Science*, vol 8616. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-44371-2_25

Security Testing Setup for Spectre Attacks

1. Overview

Test whether your systems (hypervisors, VMs, browsers, mobile apps, IoT devices) are susceptible to Spectre-class attacks and whether mitigations (retpoline, LFENCE/CSDB, compiler/firmware patches, microcode, sandbox hardening) are correctly applied and effective.

2. Security Test Approach & Tool

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
White-box	SAST	Source & compiler review for speculative-safe coding	Compiler options & static analysis	Compiler options	Both
White-box	DAST	System mitigation verification	spectre-meltdown-checker	spectre-meltdown-checker	Both (Linux/Unix)
Black-box	DAST	PoC attack runs (research PoCs)	SpectrePoC / spectrev2-poc repositories	SpectrePoC	Both (lab only)
Gray-box	DAST	Browser / JS exploitability checks	Google Spectre PoC (JS) & browser testbeds	Google Spectre PoC	Both (browser)
White-box	SAST	Kernel & hypervisor mitigation review	Linux hw-vuln docs + kernel config checks	Linux hw-vuln docs + kernel config checks	Both (cloud/hypervisor)
White-box	DAST	Mitigation technique validation (retpoline / fences)	Intel / AMD mitigation guides (retpoline, LFENCE, CSDB)	Intel / AMD mitigation guides	Both
Gray-box	DAST	Performance counter telemetry & anomaly detection	perf / PMU monitoring / SIEM rules	perf	Both

3. Minimal Testbed & Short Steps

1. **Get approval & isolate** — use air-gapped VLANs or dedicated lab hardware; snapshot/rollback images.
2. **Inventory targets** — cloud hypervisors/hosts, container hosts, browser versions, Android builds, IoT boards (with speculative-capable CPUs).
3. **Check mitigations** — run `spectre-meltdown-checker` (or vendor tools) to report mitigation state and needed updates.
4. **Run PoCs in lab only** — run vetted PoC repos (SpectrePoC, spectrev2-poc) to test exploitability; treat results carefully and stop if unsafe.
5. **Browser & mobile checks** — use Googleâ€™s JS PoC to test browser hardening and apply site-isolation / JIT mitigations where applicable.
6. **Kernel/hypervisor review** — verify retpoline/fence mitigations, microcode updates, and kernel configs (see Linux hw-vuln docs).
7. **Telemetry & detection** — monitor performance counters (`perf`) for abnormal branch/misprediction patterns and integrate SIEM alerts.
8. **Remediate & retest** — apply microcode, compiler/OS patches, enable retpoline or LFENCE/CSDB as required; retest PoCs and scanner reports.

4. References

1. Kocher, P., Horn, J., Fogh, A., et al. (2020). Spectre attacks: Exploiting speculative execution. *Communications of the ACM*, 63(7), 93-101.
2. Wang, G., Chattopadhyay, S., Gotovchits, I., Mitra, T., & Roychoudhury, A. (2019). oo7: Low-overhead defense against spectre attacks via program analysis. *IEEE Transactions on Software Engineering*, 47(11), 2504-2519.
3. Depoix, J., & Altmeyer, P. (2018). Detecting spectre attacks by identifying cache side-channel attacks using machine learning. *Advanced Microkernel Operating Systems*, 75, 48.

Security Testing for Meltdown Attacks

1. Overview

Meltdown is a hardware vulnerability that allows a process to read kernel (or other privileged) memory by exploiting out-of-order execution and side-channels. While originally documented on desktop/cloud CPUs, the risk extends to mobile/IoT devices (embedded processors with speculation or caches) and multi-tenant cloud resources. ([arXiv][1]) In a cloudâ€“mobileâ€“IoT ecosystem, an attacker could exploit Meltdown (or similar speculative execution side-channels) to leak sensitive data from other tenants, device firmware, mobile apps, or IoT gateways. So testing for it—and verifying mitigation—is important.

2. High-level Testing Workflow / Setup

1. **Scope & threat modelling**

- Identify devices/processors in your ecosystem (cloud hosts, edge gateways, IoT devices, mobile devices) that may support speculative execution / caches.
- Identify privilege boundaries (user space vs kernel space, firmware vs OS, mobile app sandbox vs native OS, IoT device kernel vs firmware).
- Map data flows: mobile â†” IoT gateway â†” cloud; multi-tenant cloud VMs/containers; shared edge devices; firmware updates.

2. **Baseline performance & telemetry capture**

- Capture baseline hardware metrics (cache miss/hit rates, branch mispredictions, performance counter data) for "ormal" operation.
- Capture IoT/gateway/mobile telemetry: firmware version, CPU microarchitecture, patch-level, kernel isolation settings (e.g., KPTI).
- Document which processors are patched, which are still vulnerable.

3. **Static analysis (SAST) & configuration review**

- Review mobile app, gateway firmware, OS kernels for speculation/side-channel aware code (e.g., avoiding vulnerable instruction sequences).
- Review cloud host configurations: Is Kernel Page Table Isolation (KPTI) or equivalent enabled? Are hyperthreading/Simultaneous MultiThreading (SMT) disabled if required? Are microcode updates applied?

4. **Dynamic testing (DAST)**

- On test machines/devices, run proof-of-concept Meltdown (or variants) in a contained lab to verify leakage is possible (only in test environment). See educational labs. ([seedsecuritylabs.org][2])
- Use hardware performance counters (HPCs) to detect abnormal cache miss/miss ratios or branch mispredictions while running suspected attack code. ([trendmicro.com][3])
- For mobile/IoT, attempt to execute code (in controlled lab) that performs transient memory reads via speculative side-channel and observe if data leakage is possible.

5. **Cloud/tenant isolation testing**

- In a multi-tenant cloud scenario, test if one tenantâ€™s process can execute speculative-execution sequence to read host or other VM memory (in lab).
- Validate hypervisor/CPU microcode/host patches are in place; test isolation boundaries.

6. **Monitoring, detection & alerting**

- Set up monitoring of hardware performance counters (cache-miss, branch mispredict, TLBS) for anomalies correlated to speculative attacks.
- Integrate logs/alerts when abnormal micro-architectural behaviour is detected. Use EDR/UEBA techniques.

7. **Reporting & remediation**

- Identify devices/hosts that remain vulnerable (old CPU, lacking microcode/OS patch).
- Recommend mitigation: microcode/firmware updates, KPTI (for x86), disable SMT/HT where necessary, apply patches on mobile/IoT OS, review code for side-channel safe patterns.
- Validate through re-testing that no leakage occurs.

---

3. Security Testing Approach & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
White-box	SAST	Code review / firmware review for speculation-safe patterns	Ghidra / Binwalk (firmware) / Static code analysis	Guidra	IoT/embedded/gateway
Gray-box	SAST	Host/OS configuration review (KPTI, microcode patch, SMT settings)	OSQuery / custom config scripts	Host/OS configuration review	Cloud/edge/gateway
Black-box	DAST	Proof-of-concept Meltdown side-channel execution	SEED Lab Meltdown VM / custom PoC code	EED Lab Meltdown VM	Cloud host/test machine
Gray-box	DAST	Hardware performance counter monitoring (cache-miss, branch mispredict)	perf (Linux) / Windows Performance Counters	perf	Cloud/host/mobile
Gray-box	DAST	Mobile/IoT speculative workload test	Custom microbenchmark code targeting speculative execution edge	— (internal)	Android, IoT
White-box	SAST	Cloud hypervisor/VM isolation review	Hypervisor audit tools / vendor guidance review	Vendor documentation	Cloud

Black-box	DAST	Monitoring anomaly detection (side-channel exploit activity)	Elastic + performance counter ingestion / Trend Micro side-channel detector	[Elastic + performance counter]()	Cloud/host

4. Practical Testbed / Setup Checklist

Test machines/devices:

Cloud/host machine: x86 processor known to be vulnerable (or deliberately unpatched) in a contained lab environment.

- IoT/edge device: ARM or x86 embedded board with OS and kernel that may allow speculation side-channels.
- Mobile device: Android (preferably debug/rooted for experimentation) or iOS if hardware supports speculative exec vulnerabilities.

Baseline measurement:

On each device, measure performance counters for normal workloads (cache misses, branch mispredicts, SMT behaviour, memory access times) and log them.

Firmware/OS configuration review:

Check that KPTI or equivalent isolation is enabled on host OS.

- Check firmware/microcode patch status on processors (Intel microcode updates, ARM equivalents).
- Check that SMT/hyperthreading settings are mitigated if required.

Attack emulation:

Use a VM/testbed (e.g., SEED Lab VM) to execute Meltdown PoC code and verify leakage of kernel memory in lab. ([seedsecuritylabs.org][2])

- On mobile or IoT device, if applicable, deploy microbenchmark code that reads privileged memory via speculative side channels (in a test environment only).

Monitoring & detection setup:

Configure performance-counter logging and ingest logs into central monitoring (Elastic / SIEM).

- Define alert thresholds: e.g., unusual spike in cache-misses, branch mispredicts, high fault counts, etc.

Isolation & containment tests:

On cloud host with multiple VMs, attempt to run attack from one VM to read memory of another (lab only).

Remediation validation:

After mitigation (patches, KPTI, microcode), retest to verify that speculative side-channel leakage is prevented or severely reduced.

Documentation & report:

Document vulnerable devices, mitigation status, residual risk, alerting/monitoring gaps, and remediation tasks.

5. References

1. Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Horn et al. (2020). Meltdown: Reading kernel memory from user space. *Communications of the ACM*, 63(6), 46-56.
2. Hill, M. D., Masters, J., Ranganathan, P., Turner, P., & Hennessy, J. L. (2019). On the spectre and meltdown processor security vulnerabilities. *IEEE Micro*, 39(2), 9-19.
3. Pan, Z., & Mishra, P. (2021, December). Automated detection of spectre and meltdown attacks using explainable machine learning. In *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (pp. 24-34). IEEE.

Security Testing for Hardware Integrity Attacks

1. Overview

Hardware integrity attacks (hardware Trojans, supply-chain implants, fault/glitch injection, side-channel exfiltration, physical tampering and counterfeiting) can be introduced during design, fabrication, distribution, or deployment and are difficult to detect with software-only testing; so hardware-focused validation and supply-chain controls are required.

2. Testing Setup

1. **Threat modelling & asset inventory**
- Inventory components (MCU/SoC, radio modules, power management, secure elements, bootloader, firmware, enclosures, connectors) and define trust boundaries (cloud, gateway, mobile client, device hardware). Use SBOMs and HW provenance records where possible.
2. **Golden-reference & baseline creation**
- Maintain golden designs/firmware images and expected side-channel / performance baselines for comparison (voltage, current, power traces, timing). Baselines enable anomaly detection for implants or tampered silicon.
3. **Pre-silicon & supply-chain controls**
- Design reviews, IP vetting, EDA flow checks, provenance tracking, secure procurement and QA audits. (Mitigates insertion during design/fab/distribution.)
4. **Static analysis (SAST) of RTL/firmware**
- RTL/netlist checks, EDA tool logs, firmware source code review, secure-boot validation, digital signatures and SBOM verification.
5. **Dynamic hardware testing (DAST / post-silicon)**
- Side-channel analysis (power, EM) versus baseline, fault-injection (voltage/clock/glitch, EM, laser if available), JTAG/Debug port probing, bus sniffing, physical tamper / enclosure stress testing.
6. **Firmware reverse engineering & runtime instrumentation**
- Extract firmware (if possible), perform binary analysis, fuzz firmware interfaces, emulate where feasible, use runtime instrumentation (Frida, dynamic hooks) on mobile apps that interact with devices.
7. **Reverse engineering & physical inspection**
- X-ray, decap / optical inspection, PCB trace audit, component authenticity checks, BGA inspection, and microprobing where resources allow.
8. **Monitoring & runtime integrity**
- Deploy runtime monitors, TPM/secure element attestation, anomaly detection in cloud telemetry (unusual behavior from a device), and continuous re-validation (periodic re-attestation).
9. **Reporting & remediation**
- Triage anomalies into firmware patch, revocation / recall, procurement policy changes, or forensics / disclosure.

---

3. Security Testing Approach & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
White-box	SAST	RTL / Netlist review	Formal/Static EDA checks (e.g., SpyGlass, Custom scripts)	Formal/Static EDA checks	Both (HW-level)
Gray-box	DAST	Side-channel analysis (SCA)	ChipWhisperer	ChipWhisperer	Both
Gray-box	DAST	Fault / Glitch injection	ChipWhisperer (glitch), Riscure Fault Injection lab (commercial)	ChipWhisperer (glitch)	Both
Black-box	DAST	Debug / JTAG port enumeration	JTAGulator	JTAGulator	Both
Gray-box	DAST / SCA	Bus sniffing / protocol analysis	Bus Pirate, Saleae Logic	Bus Pirate	Both
Gray-box	SAST / DAST	Firmware extraction & analysis	Binwalk, Firmadyne, Ghidra	Binwalk	Both
Gray-box	DAST	Runtime instrumentation / mobile â†” device	Frida		Android, iOS
White-box	SAST	Platform / OS HW checks	chipsec (Intel)/Platform diagnostics	chipsec (Intel)	Both (x86 targets; limited ARM support)
Black-box	DAST	Firmware fuzzing	TriforceAFL / RPFuzzer / GraphFuzz (research)	Relevant repos/papers (TriforceAFL variants) / arXiv (GraphFuzz)	Both

Black-box	DAST	Physical tamper & enclosure testing	EM probe, thermal imaging, mechanical tamper tools	See vendor sites (e.g., Keysight, Tektronix)	Both
Gray-box	SAST	Supply-chain provenance & SBOM checks	SBOM tools (CycloneDX/SPDX tooling), procurement auditing	SBOM tools	Both
Gray-box	DAST	Reverse engineering / PCB inspection	X-ray, microscope, decap labs, PCB inspection tools	Laboratory services & vendors (e.g., TEK/Keysight/third-party labs)	Both

4. Practical Testbed Setup

- **Hardware lab:** ChipWhisperer kit, high-speed oscilloscope (â‰¥100 MS/s for power traces), EM probe, programmable power supply, glitching module (voltage/clock), JTAGulator, Bus Pirate, Saleae logic analyzer, bench microscope, hot-air rework station, X-ray / decap access via external lab (if required).
- **Firmware & analysis workstation:** Kali/Ubuntu, Ghidra, Binwalk, Firmadyne, IDA/objdump, radare2, custom scripts.
- **Mobile test devices:** rooted/jailbroken Android and iOS test phones (for instrumentation), Frida + adb/ideviceinstaller.
- **Cloud side:** telemetry ingestion, attestation verification services, automated anomaly detection (compare device behavior vs golden baseline).
- **Supply-chain tooling:** SBOM generation (CycloneDX/SPDX), procurement provenance database, certificate-based attestation for secure elements.
- **Safety & compliance:** ESD-safe bench, documented chain of custody procedures for examined devices, legal sign-offs for destructive analysis and export-controlled equipment.

5. Quick Example Test Scenarios

1. **Side-channel detection of hardware Trojan**
- Capture power traces from device running known workload; compare statistical features to golden reference; use ChipWhisperer + PCA / ML anomaly detector.
2. **Glitch injection to bypass secure boot**
- Apply clock/voltage glitch at boot to see if bootloader signature checks can be bypassed; perform repeated tests, correlate with golden logs.
3. **JTAG port discovery & firmware dump**
- Use JTAGulator to locate debug pins, use OpenOCD to read memory/flash; analyze firmware with Ghidra and binwalk.
4. **Supply-chain & provenance audit**
- Validate SBOM; check component lot numbers and compare with expected suppliers; run counterfeit part checks (visual / X-ray if suspicious).

6. References

1. Sidhu, S., Mohd, B. J., & Hayajneh, T. (2019). Hardware security in IoT devices with emphasis on hardware Trojans. *Journal of Sensor and Actuator Networks*, 8(3), 42. https://doi.org/10.3390/jsan8030042.
2. Chen, D., Xu, Y., Liu, X., Zhang, F., He, G., & Chen, Y. (2020). Hardware Trojans in chips: A survey for detection and prevention. *Sensors* (Basel), 20(18), 5165. https://doi.org/10.3390/s20185165.
3. Rao, A., Priya, A., Richardson, M., Dorey, P. & Brown R. (2022). Securing the Internet of Things supply chain. *IoT Security Foundation*. https://www.iotsecurityfoundation.org/.
4. Chatterjee, D., Maitra, S., Mishra, N., Shukla, S., & Mukhopadhyay, D. (2025). Hardware security in the connected world. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 15(3), e70034.
5. Situ, L., Zhang, C., Guan, L., Zuo, Z., Wang, L., Li, X., Liu, P. & Shi, J. (2023). Physical devices-agnostic hybrid fuzzing of IoT firmware. *IEEE Internet of Things Journal*, 10(23), 20718-20734.

Security Testing setup for Rowhammer Attacks

1. Overview

Rowhammer is a hardware/DRAM failure phenomenon where repeatedly activating ("hammering") certain DRAM rows causes bit flips in adjacent rows. Those bit flips can be induced from software and exploited for privilege escalation, cross-VM attacks, or data corruption including practical attacks from JavaScript, Android apps, and co-located VMs. Rowhammer continues to be relevant as DRAM scales and remains a realistic attack surface in cloud, desktop, mobile and some IoT platforms.

2. High-level Testing Workflow / Objectives

1. **Scope & threat model** — enumerate devices and contexts where DRAM is present and trusted: cloud hypervisors / VMs, edge gateways, mobile devices (Android), IoT devices with DRAM, and browser-based clients. Identify assets that would be impacted by bit flips (kernel pages, crypto keys, page tables, VM page caches).
2. **Baseline & instrumentation** — ensure test systems have full logging, kernel crash dumps, performance counters available (e.g., `perf`), and remote logging to a secured collector. Snapshot images for fast rollback.
3. **Static & source review (SAST)** — review code paths that rely on DRAM integrity (hypervisor memory isolation, page deduplication, memory deduplication, kernel modules) and note high-value targets (e.g., page caches used by privileged processes).
4. **Controlled dynamic testing (DAST)** — run Rowhammer test suites (safe/authorized, in lab) to detect whether a given DRAM module / platform exhibits bit flips and whether flips can be exploited to alter privileged data. Test different hammering patterns (single-sided, double-sided, one-location) and degrees of aggressiveness.
5. **Exploitability assessment** — attempt end-to-end demonstration in a controlled environment: userland to kernel privilege escalation, VM escape (Flip Feng Shui style), or mobile privilege escalation (Drammer). Only do this with explicit authorization and in an isolated lab.
6. **Detection & monitoring tests** — validate whether available mitigations and telemetry (hardware mitigations, ECC, TRR, increased refresh, OS mitigations, performance counter anomalies) detect hammering or flips. Measure false positive/false negative tradeoffs.
7. **Reporting & remediation** — produce prioritized findings (vulnerable modules, required mitigations, compensating controls) and validate fixes (firmware updates, OS/hypervisor patches, disabling risky features like page deduplication, enabling ECC/targeted refresh).

---

3. Security Test Approaches & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Black-box	DAST	DRAM vulnerability detection (hammer tests)	Google / CMU rowhammer-test / antmicro rowhammer-tester	Google / CMU rowhammer-test	Both
Black-box	DAST	Remote JS hammer (research / PoC)	rowhammer.js (research implementation)	rowhammer.js (research implementation)	Both
Gray-box	DAST	Android deterministic Rowhammer testing / exploitability	Drammer (vusec) + Drammer repo	Drammer (vusec) + Drammer repo	Android
Gray-box	DAST	Cross-VM disturbance / Flip Feng Shui-style tests	Flip Feng Shui artifacts / testbed (research)	Flip Feng Shui artifacts	Both (cloud hypervisor)
White-box	DAST	Hardware/firmware test and mitigation validation	antmicro rowhammer-tester, CMU Rowhammer repo	antmicro rowhammer-tester	Both
Gray-box	DAST	Memory allocator shaping (force desired physical placement)	custom allocators, hugepages, memtester, stress-ng	tress-ng	Both
White-box	DAST	Performance counter & telemetry monitoring for hammering	perf / Intel PCM / OS counters / kernel instrumentation	perf	Both
White-box	SAST	Code review of OS/hypervisor features (page dedup, copy-on-write)	Semgrep / CodeQL / manual code review	Semgrep	Both

---

4. Practical Testbed — Minimum Safe Configuration

- **Isolated lab network:** fully air-gapped or VLAN + physical isolation with snapshot/rollback capability. Do **not** run any exploitative tests on production systems.
- **Test hardware:** representative systems for each target class: cloud host (hypervisor + guest VMs), Android phones (for Drammer-style tests), edge gateways/IoT devices with DRAM (if applicable). Keep identical hardware to production where possible.
- **DRAM characterization tools:** Google/CMU rowhammer-test, Antmicro rowhammer-tester, CMU Rowhammer repo.
- **Exploit PoCs (research only):** rowhammer.js (for browser tests), Drammer (Android). Only use published PoCs to evaluate whether vulnerabilities are exploitable in your environment — with permission.

- **Monitoring & telemetry:** kernel crash dumps, `perf` counters, PCM, syslogs forwarded to a secure collector (Elastic/Splunk).
- **Controlled allocation tools:** stress-ng, memtester, custom allocators/hugepages to shape physical placement and reduce noise.
- **Snapshots and rollback:** VM snapshots and physical disk images so tests are non-destructive and recoverable.

---

5. Step-by-step Test Procedures

A. Preparation

1. Get written authorization and define scope: targeted hosts, permitted PoCs, data handling rules.
2. Prepare snapshots / backups for all DUTs (devices under test). Enable full kernel logging and collect baseline performance counters.
3. Ensure lab isolation — no accidental internet exposure, and emergency kill switches (power or VM pause).

B. Characterize susceptibility (non-exploit test)

1. Run `rowhammer-test` / `antmicro rowhammer-tester` on the target host to determine whether bit-flips can be induced and at what aggressiveness thresholds (hammer count, refresh intervals). Record flip patterns (addresses, rows, timing).
2. Repeat at different temperatures, DRAM frequencies and voltages to map sensitivity (Rowhammer depends on physical conditions).

C. Allocation shaping & exploitability

1. Use memory shaping techniques (hugepages, allocation patterns) to co-locate attacker memory with victim pages (for Flip Feng Shui and VM attacks). Research artifacts (Flip Feng Shui) detail techniques for influencing physical placement.
2. Attempt controlled PoC (only inside lab): e.g., run a guest VM hammering rows to see whether you can flip bits in a co-located victim VM (if scope authorizes cross-VM tests). Log progress and abort if unintended behavior occurs.

D. Mobile tests (Android)

1. On an Android test device, run Drammer per the research instructions to test deterministic hammering on ARM/Android platforms. Verify whether privilege escalation (PoC) is possible **only** if explicitly in scope — otherwise run non-exploit detection-only mode.

E. Browser tests

1. As an additional measurement, run controlled rowhammer.js tests on a test browser/device to evaluate remote attack feasibility. Note that browser vendors mitigate aggressively; results will vary.

F. Detection validation

1. Monitor performance counters (cache miss rates, DRAM row activations) and test detection heuristics (e.g., high activation rates, abnormal `perf` metrics). Evaluate whether telemetry reliably signals hammering and whether flip events are visible in logs.

G. Mitigation verification

1. Validate deployed mitigations: ECC memory corrects single-bit flips (verify via induced flips), targeted refresh (TRR) or vendor features, disabling page deduplication (KSM), kernel mitigations, or OS / hypervisor patches. Test that mitigations prevent exploitability under comparable hammering conditions.

---

6. References

1. Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J. H., Lee, D., ... & Mutlu, O. (2014). Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. *ACM SIGARCH Computer Architecture News*, 42(3), 361-372. https://doi.org/10.1145/2678373.2665726.
2. Seaborn, M., & Dullien, T. (2015). Exploiting the DRAM rowhammer bug to gain kernel privileges. *Black Hat*, 15(71), 2.
3. Gruss, D., Maurice, C., Mangard, S. (2016). Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript. In: Caballero, J., Zurutuza, U., RodrÃguez, R. (eds) Detection of Intrusions and Malware, and Vulnerability Assessment. DIMVA 2016. *Lecture Notes in Computer Science*, vol 9721. Springer, Cham. https://doi.org/10.1007/978-3-319-40667-1_15.
4. Van Der Veen, V., Fratantonio, Y., Lindorfer, M., Gruss, D., Maurice, C., Vigna, G., Bos, H., Razavi, K. & Giuffrida, C. (2016). Drammer: Deterministic rowhammer attacks on mobile platforms. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security* (pp. 1675-1689). https://doi.org/10.1145/2976749.2978406.
5. Razavi, K., Gras, B., Bosman, E., Preneel, B., Giuffrida, C., & Bos, H. (2016). Flip feng shui: Hammering a needle in the software stack. In *25th USENIX Security Symposium (USENIX Security 16)* (pp. 1-18). https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/razavi.
6. Mutlu, O., & Kim, J. S. (2019). Rowhammer: A retrospective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(8), 1555-1571. 10.1109/TCAD.2019.2915318.

Security Testing Setup for Reverse Engineering Attacks

1. Overview

Reverse engineering involves disassembling or analyzing applications, binaries, or firmware to understand internal logic, extract sensitive data, or modify behavior. In cloud-mobile-IoT environments, attackers may:

- **Decompile mobile apps** to extract API keys or bypass logic.
- **Analyze firmware** to find hardcoded credentials or debug interfaces.
- **Intercept cloud communication** to reverse protocols or authentication flows.

Recommended Testing Layers

1. **Static Analysis (SAST)**: Disassemble and inspect code or firmware for secrets, logic flaws, or insecure configurations.
2. **Dynamic Analysis (DAST)**: Monitor runtime behavior, memory, and network traffic for tampering or reverse engineering attempts.
3. **Physical Inspection**: Evaluate hardware for debug ports, unprotected storage, or firmware extraction vectors.
4. **Penetration Testing**: Simulate reverse engineering scenarios using emulators, patching, and instrumentation.

---

2. Security Testing Approach & Tools

Test Approach	Analysis Type	Approach Name	Testing Tool	Tool Hyperlink	Platform
White-box	SAST	Code Review	Ghidra	Ghidra	Both
Gray-box	SAST	Code Security Scanner	MobSF	MobSF	Both
Black-box	DAST	Fuzzing	Peach Fuzzer	Peach Fuzzer	Both
Gray-box	DAST	Network Packet Sniffing	Wireshark	Wireshark	Both
Black-box	DAST	Pentesting	Frida	Frida	Both
White-box	SAST	Firmware Analysis	Binwalk	Binwalk	IoT
White-box	Physical Review	Physical Security Measures Review	ChipWhisperer	ChipWhisperer	IoT

---

3. References

1. Sequeiros, J. B., Chimuco, F. T., Samaila, M. G., Freire, M. M., & InÃ¡cio, P. R. (2020). Attack and system modeling applied to IoT, cloud, and mobile ecosystems: Embedding security by design. *ACM Computing Surveys (CSUR)*, 53(2), 1-32.
2. Elsayed, E. K., ElDahshan, K. A., El-Sharawy, E. E., & Ghannam, N. E. (2019). Reverse engineering approach for improving the quality of mobile applications. *PeerJ Computer Science*, 5, e212.
3. Shwartz, O., Mathov, Y., Bohadana, M., Elovici, Y., & Oren, Y. (2018). Reverse engineering IoT devices: Effective techniques and methods. *IEEE Internet of Things Journal*, 5(6), 4965-4976.
4. Franke, D., Elsemann, C., Kowalewski, S., & Weise, C. (2011, October). Reverse engineering of mobile application lifecycles. In *2011 18th Working Conference on Reverse Engineering* (pp. 283-292). IEEE.
5. Albakri, A., Fatima, H., Mohammed, M., Ahmed, A., Ali, A., Ali, A., & Elzein, N. M. (2022). Survey on Reverseâ€Engineering Tools for Android Mobile Devices. *Mathematical Problems in Engineering*, 2022(1), 4908134.
6. Chavan, N., Patel, H., & Shah, K. (2025). Comprehensive Approach to Android Penetration Testing: Techniques, Tools and Best Practices for Securing Mobile Applications. In *2025 12th International Conference on Computing for Sustainable Global Development (INDIACom)* (pp. 01-07). IEEE.

Security Testing Setup for VM Escape Attacks

1. Overview

Focus on **hypervisor/device-interface fuzzing, nested-virtualization testbeds, VM-introspection + crash triage, and targeted pentesting of virtual device paths (network, block, hypercalls, MMIO/PIO)** â€" these are where real VM-escape bugs are found.

---

2. Security Testing Approaches & Tools

Test approach	Analysis Type	Approach name	Testing Tool	Tool Hyperlink	Platform
Black-box / Gray-box	DAST (dynamic)	Hypervisor / virtual-device fuzzing	HYPERPILL	HYPERPILL (USENIX '24)	Both (QEMU/KVM/Hyper-V)

Gray-box	DAST	Nested virtualization fuzzing	hAFL2 (hypervisor fuzzer)	hAFL2 / SafeBreach writeup	Both (KVM/QEMU/Hyper-V)
Gray-box	DAST	Virtual device fuzzing (AFL-based)	AFL + AFL harnesses for QEMU	Black Hat: AFL virtual device fuzzing	Both (QEMU/Android emulator)
Black-box	DAST	Penetration testing / exploit chains	Metasploit, custom exploit modules, Immunity CANVAS	Metasploit	Both
White-box	SAST	Source code review / secure code scan	Coverity, SonarQube, CodeQL	CodeQL	Both
Gray-box	DAST / Forensics	Virtual Machine Introspection (VMI)	LibVMI, Volatility	LibVMI	Both
Black-box	DAST	Hypervisor config & surface scanning	Nmap, Shodan (discovery), Nessus/OpenVAS	OpenVAS	Both
Black-box	DAST / Network	Network monitoring / packet analysis	Wireshark, tcpdump	Wireshark	Both
Gray-box	DAST / Binary	Binary diffing & reverse	BinDiff, Diaphora, IDA/Ghidra	Ghidra	Both
Gray-box	DAST / Crash analysis	Crash triage / corpus minimization	afl-cmin/afl-tmin, GDB / WinDbg	AFL	Both
White-box	SAST	Driver/VM service code review (hypercall, VSP)	Static analyzers + manual review	SonarQube	Both

3. Short Testing Setup

1. **Testbed & isolation**
- Prepare dedicated physical lab host(s); enable nested virtualization if you plan multi-layer fuzzing (host → L1 → L2). Use QEMU/KVM on Linux for reproducibility. (many fuzzers rely on nested setups).
2. **Baseline images**
- Build minimal guest images (Linux/Windows) with test harnesses (custom hypercall handlers or guest code that triggers device paths). For mobile: use Android emulator (QEMU-based) images; for iOS, prefer macOS virtualization/hypervisor framework where applicable.
3. **Fuzzing / dynamic testing**
- Use HYPERPILL or hAFL2 to snapshot the hypervisor and fuzz hardware interfaces (MMIO/PIO/hypercalls/DMA). Corpus minimization and coverage-guided mutation are critical. Log crashes with full VM snapshots for offline triage.
4. **Pentest & exploit chaining**
- Use Metasploit or custom exploit modules to validate real escapes (if permitted by policy). Prioritize paths that cross from VM â†’ host services: paravirtual drivers, virtual NICs, shared folders, host agent services.
5. **Introspection & monitoring**
- Attach LibVMI / Volatility to detect successful guest actions and to extract memory at crash time for root cause. Use these to confirm host compromise vs. guest crash.
6. **Static analysis**
- Run SAST (CodeQL / Coverity / SonarQube) on hypervisor code or virtual-device drivers (when source is available) to find integer overflows, unchecked memcpy, etc.
7. **Triage & report**
- For each crash: collect VM snapshot, gdb/WinDbg backtrace, binary diffing (if vendor binary), reproduce, and prepare PoC with responsible disclosure steps.
8. **Hardening validation**
- Re-run fuzzing and targeted tests after mitigations (e.g., bounds checks, privilege separation, reducing shared device surface). Use moving-target or randomized builds where possible.

4. References

1. Bulekov, A., Liu, Q., Egele, M., & Payer, M. (2024). {HYPERPILL}: Fuzzing for Hypervisor-bugs by Leveraging the Hardware Virtualization Interface. In 33rd USENIX Security Symposium (USENIX Security 24) (pp. 919-935).

2. Tang, J., & Li, M. (2016). When virtualization encounter AFL. Black Hat Europe.
3. Schumilo, S., Aschermann, C., Abbasi, A., WÃ¶rner, S., & Holz, T. (2020, February). HYPER-CUBE: High-Dimensional Hypervisor Fuzzing. In NDSS.
4. Shi, H., Mirkovic, J., & Alwabel, A. (2017). Handling anti-virtual machine techniques in malicious software. ACM Transactions on Privacy and Security (TOPS), 21(1), 1-31.