You Bought It, They Bricked It: A Call for Reclaiming Local Control

Jack Tigar Humphries
jhumphri@cs.stanford.edu
Stellar Development Foundation
New York, NY, US

Abstract

Cloud-tethered devices can become e-waste when vendors retire required applications or services, even while their hardware remains functional. Vendors need not support devices forever, but ending support and blocking replacement software are distinct choices. Foundation models may reduce the cost of reconstructing proprietary management software. Using AI, collected traces, and minimal binary inspection, we recovered two versions of Apple's encrypted, undocumented AirPort device-management protocol and built a compatible client tested across six models and four firmware versions. We retrospectively estimate that repeating this recovery would require four to five days of human work, compared with the months we would expect from manual reverse engineering. This work demonstrates human-guided recovery of a local management protocol that may generalize to autonomous reconstruction.

We call for an end to non-consensual e-waste through an end-of-life control-transfer principle: when support ends, vendors should support local operation and release the information and authority required by replacement software. Because vendor incentives may conflict with continued independent operation, we argue this principle should be backed by enforceable requirements, such as regulation. To make independent recovery practical, we formulate AI-assisted de-cloudification as a closed-loop experimental process and identify the systems challenges it raises. AI-assisted recovery should remain a fallback, however, not a substitute for a deliberate vendor-provided end-of-life path.

CCS Concepts

• **Networks** → **Network protocols**; *Network components*; *Network management*.



This work is licensed under a Creative Commons Attribution 4.0 International License.

HotNets '26, Salt Lake City, UT, USA

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-3029-0/2026/11
https://doi.org/10.1145/3847290.3859946

Keywords

IoT, protocols, reverse engineering, traces

ACM Reference Format:
Jack Tigar Humphries. 2026. You Bought It, They Bricked It: A Call for Reclaiming Local Control. In *The 25th ACM Workshop on Hot Topics in Networks (HotNets '26), November 16–17, 2026, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3847290.3859946

1 Introduction

Cloud-tethered devices can fail before their hardware does. A vendor may retire an application or cloud service, or make important functionality contingent on a subscription. Supporting deployed devices indefinitely is not free, yet vendors may also benefit when discontinuation forces owners to purchase replacements. The result is non-consensual e-waste as owners must replace working hardware because a vendor withdrew the means or authorization to use it.

Spotify disabled its $90 Car Thing devices just five months after launch and directed users to recycle them [15]. Nest shut down the cloud service required by its $299 Revolv smart-home hubs, rendering them inoperable [25, 28]. Belkin ended Wemo cloud services and app support, causing affected products to lose app control, remote access, and voice-assistant integration [8, 14].

Even when devices are not immediately disabled, vendors can retain long-term control through proprietary software. Apple's AirPort routers require its proprietary AirPort Utility for configuration [2–4], and its loss of support on macOS 27 and iOS 27 threatens owners' ability to manage working hardware [16]. eero reserves daily and monthly usage histories and security analytics for paying eero Plus subscribers [22, 23]. Samsung added refrigerator advertisements through a software update [20, 48] and a later update reportedly bricked refrigerators inadvertently [42], causing food spoilage just before a major holiday. BMW offered, then abandoned, subscriptions to activate installed heated seats [29, 30]. These cases range from lost functionality to restrictions on use, but all leave vendors controlling what owners can do with their hardware.

Reconstructing closed network management protocols and software can require months of expert reverse engineering. Our AirPort experience suggests that AI can reduce this barrier. Using AI, we recovered two versions of Apple's encrypted, undocumented AirPort Configuration Protocol and built a compatible client tested across six models and four firmware versions. We complete a *human*-guided recovery of a local management protocol for one device type, but not yet an autonomous recovery across unrelated devices or reconstruction of a cloud service that no longer exists. Protocol recovery cannot by itself supply unavailable cloud state or vendor-controlled authority.

Our call to action is simple: **support may end, but control should transfer.** Vendors should not be required to operate applications and cloud services indefinitely; however, when support ends, vendors should support local operation and release the documentation and authority needed by replacement software.

Vendor cooperation cannot be assumed when continued dependence or replacement purchases benefit the vendor. While industry standards can define control-transfer mechanisms, regulation may be needed to ensure vendors provide them. We advocate enforceable end-of-life obligations for features that owners or third parties can sustain independently. For other features, consumer-protection rules should require pre-sale disclosure of cloud dependencies.

Where no exit path exists, *de-cloudification* offers a technical fallback through a local replacement management application. A central networking challenge is recovering proprietary device-management protocols. Building a compatible replacement requires reconstructing message formats and other concerns like authentication and command semantics. One alternative is to install replacement firmware, but doing so is often infeasible or unnecessarily invasive. Replacement firmware may be blocked by signature checks [26, 50] or complicated by proprietary drivers [12, 27, 36]. A compatible external management application, such as our AirPort replacement, preserves the existing Apple-signed firmware. Recovering its protocol still requires collecting traces, implementing a replacement protocol, and testing it against the original system [10, 11, 13, 32, 39, 53].

Foundation models may make this work substantially cheaper [9, 18, 24, 31, 34, 35, 40, 47, 55, 56]. Given traces and implementation artifacts like binaries, an AI system can generate plausible descriptions of the protocol and implement them with code. Yet a candidate implementation may explain every observed packet while mishandling an unobserved state. As hypothesis generation becomes cheaper, the bottleneck shifts to **knowing which hypothesis is correct**, which includes collecting representative traces, building behavior oracles, and choosing more traces to collect to clarify any unknown behavior. The central question is no longer only "What might this protocol do?" but also "What observation or trace would distinguish this explanation from the alternatives?" Even after protocol semantics are recovered, independent operation may still depend on vendor-controlled authority or unavailable cloud services.

This paper calls for end-of-life control transfer to end non-consensual e-waste, with clear pre-sale disclosure of features that cannot be controlled independently. Using AirPort as early evidence, we propose a closed-loop recovery framework and identify challenges in coverage, oracles, experiment planning, test environments, efficiency, confidence, and responsible recovery. Together with regulation for transferring control back to owners, this agenda aims to preserve independent control after vendor support ends.

2 Early Experience Reclaiming Local Control

Apple has retired its AirPort routers and announced that its proprietary management application, AirPort Utility, is no longer supported on macOS 27 or iOS 27 [16]. The AirPort ecosystem has functional black-box hardware, encrypted traffic, and undocumented protocols, and the official utility and physical devices provide ground truth for testing a replacement. Our goal was to recover enough AirPort Configuration Protocol (ACP) semantics to build a compatible AirPort Utility replacement.

Our initial approach was to ask Codex [46] with GPT-5.5 Medium [45] to reconstruct high-level code directly from the AirPort Utility binary. This approach quickly became inefficient as even a button is entangled in compiled graphics and user-interface frameworks irrelevant to reproducing its behavior. Worse, reconstructed code was difficult to connect to an observable device effect and therefore difficult to validate. We instead treated the *running* application as the specification, recovering behavior from traces and inspecting binaries only when traces did not expose information necessary for the specification.

We manually exercised AirPort Utility 6.3.9 and 5.6.1 on macOS Sequoia 15.7.7, recording timestamped screenshots, Accessibility-framework dumps of user interface elements [1], and PCAPs captured with `tcpdump` [38]. Codex inferred user actions from the graphical traces and used timestamps to correlate these actions with protocol exchanges. We attempted to explore the interface systematically and added workflows when we noticed missing behavior during testing. We began with one Time Capsule (model A1254), then expanded to two AirPort Expresses (A1088/A1392), two AirPort Extremes (A1354/A1521), and one last Time Capsule (A1470) running four firmware versions from 6.3 through 7.9.1. The A1088 uses the first (legacy) version of ACP while the other five devices use a second, substantially different version. Prior work

implemented portions of legacy ACP, including individual property reads and writes, but left many protocol features and complete management workflows unsupported [57]; we supplied this material to Codex.

The first PCAP revealed why traces alone were insufficient as ACP traffic was encrypted. We had Codex dump symbols from both AirPort Utility binaries, use Ghidra [44] and command-line object-file tools to follow relevant code into private macOS frameworks, and attach LLDB [51] to determine which cryptographic routines actually executed. Breakpoints on key-derivation and cipher-initialization routines captured per-session directional keys and associated them with individual TCP flows. Modern ACP authenticates using SRP-6a [19, 52], derives separate request and response keys using PBKDF2-HMAC-SHA1 [33], and encrypts each direction with an AES-128 counter-mode construction. Legacy ACP instead uses a 1024-bit Diffie-Hellman [21] exchange before applying the same stream construction. Codex also recovered ACP framing and serialization, enough to decrypt captures and construct a compatible client. This selective use of binary analysis was critical, but we did not otherwise need to inspect the original binary's decompiled assembly.

Codex rebuilt the interface in Swift, using Accessibility snapshots as its primary specification and screenshots when they were ambiguous. It implemented the protocol in Python so that the recovered client could also run outside macOS. It replayed each workflow through Accessibility APIs and mouse-and-keyboard automation. Each replay began from a known router configuration. A reusable Python oracle compared the decrypted packets sent and received by our implementation with those produced by AirPort Utility. It ignored differences that are nonsemantic, such as timestamps and property ordering, while requiring the relevant fields, values, and transaction groupings to agree. On a divergence, Codex revised its model and implementation and then replayed the workflow, often for several iterations before we manually inspected its changes and tested the result.

The oracle itself changed as we learned which differences mattered. During one Time Capsule user flow, AirPort Utility wrote a set of properties in one batch, whereas the replacement divided the same fields and values across three writes. Codex initially classified the outputs as semantically equivalent, but the device rejected the three-write sequence. After observing this failure, Codex revised both the implementation and the oracle to preserve the original batch. Transaction structure may therefore encode atomicity or a hidden state transition visible only on the device.

Coverage presented a separate problem. An implementation that matched the Time Capsule traces failed on the AirPort Express (A1088), which uses the older ACP protocol and exposes AirPlay [5] settings absent from the Time Capsule. We purchased four additional devices and collected new traces across hardware families, firmware versions, and capabilities. The final corpus contains roughly 184 protocol operations and 144 configuration fields across 6 AirPort devices, including about 2,700 screenshots and Accessibility snapshots and 57,000 ACP packets across 500 TCP flows. Across approximately 870 replay iterations, Codex produced roughly 50,000 lines of source and test code and 650 tests.

Within the confines of the collected traces, the replacement supports both ACP versions, device discovery and authentication, configuration of the same settings exposed by AirPort Utility, new device setup, error reporting, AirPlay and disk management, factory reset and disk erasure, and installation of Apple-signed firmware. Every recorded workflow ultimately produced semantically equivalent exchanges, and we manually tested configuration changes, restarts, resets, firmware installation, and disk erasure on physical devices. The process nevertheless required substantial human judgment as we collected traces, recognized coverage gaps, audited the oracle, inspected intermediate implementations, and physically reset devices that Codex left in invalid states. Some experiments also caused routers attached to our network to advertise weakly secured Wi-Fi SSIDs, whereas an isolated VLAN and automatic restoration of device state would have made exploration safer and cheaper.

Limitations. While this result shows human-guided recovery for the AirPort family is possible, it does not prove generalization is possible across unrelated device protocols. Furthermore, the official application and working devices remained available as references, so we did not reconstruct a discontinued cloud service or recover unavailable external state. Even within the AirPort ecosystem, we could not exercise PPPoE and sampled only a small fraction of possible network modes.

The recovery consumed billions of model tokens and we estimate that reproducing it with the described methodology requires four to five days of human work rather than the months we would expect from manual reverse engineering. Furthermore, several important parts were completed manually by a human, including collecting the traces, inspecting the oracle, resetting devices, and testing the reproduced app.

Generalizability. We think our approach could generalize to work with devices whose behavior can be seen and whose binaries are available. The core part of our approach is (1) tracing user interface actions and messages, (2) recovering limited information from the binary, and (3) comparing behavior with the original. However, it is not immediately clear how to automate all of the intensive actions like collecting traces. The following sections examine what is needed to scale this approach.

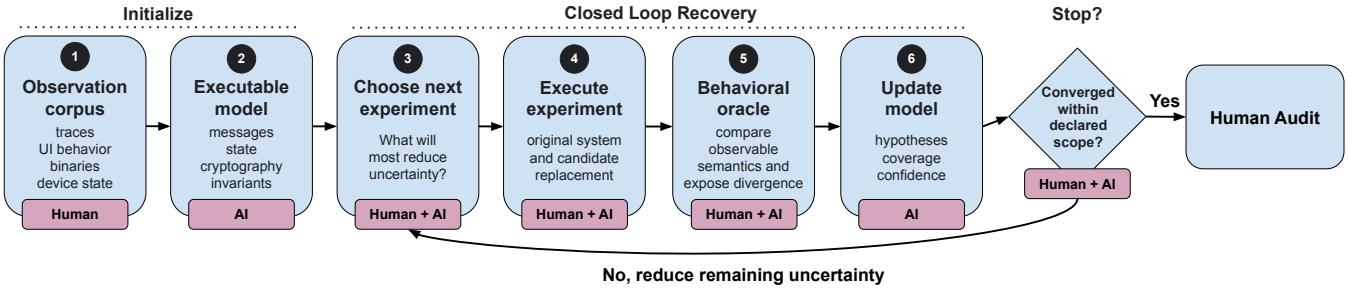


Figure 1: Our proposed closed-loop recovery framework. Labels indicate human and AI participation in the AirPort case study. Automating the remaining human tasks is a research goal.

3 Why De-Cloudification Does Not Scale

Recovery requires a behavior model that explains when operations are valid, how sessions are authorized, and how actions affect device state. A camera request that starts streaming, for example, does not reveal how the device behaves when another client is connected.

Recovering this contract requires connecting *low-level artifacts* to *high-level intent*. A packet capture shows messages without the corresponding user action that triggered them, and decompiled code similarly exposes routines without their user workflows.

Investigators must connect evidence across traces, binaries, devices, and logs, for example by following encryption through a binary before testing on hardware [10, 11, 13, 32, 39, 53]. Experiments and device differences can invalidate assumptions, so humans must decide what to investigate next. While foundation models make it easier to generate explanations and implementations [9, 18, 24, 31, 34, 35, 40, 47, 55, 56], they still require validation.

Limits of protocol recovery. What we can recover depends on where the obstacle lies. Locally accessible functionality may be recoverable through protocol reconstruction only if the owner can use a replacement application, as in AirPort. Vendor-controlled authorization may block replacement applications through device attestation, so continued operation then requires transferring authority. Firmware-signing restricts software modification rather than external applications, but it becomes a barrier if independent operation requires firmware changes the device rejects. Last, essential state or functionality may exist only in an unavailable cloud service. Recovering the cloud service's interface cannot supply missing state or compute infrastructure, so a replacement service must provide these. But all of these barriers can coexist: the proposed loop addresses missing protocol documentation while transferring control and creating replacement services address other dependencies.

4 Toward AI-Assisted De-Cloudification

AirPort suggests that de-cloudification can be organized as a closed-loop process that maintains a protocol model and candidate implementation, then asks **which next experiment will most reduce uncertainty?** The framework below is a research agenda for automating the tasks that required human intervention in our AirPort work for recovering the features an owner wants to keep using.

The loop in Figure 1 begins with an **initial corpus of traces** from users or an agent exercising the original application. Starting with common workflows, the agent varies initial states and inputs while recording the network traffic and user interface changes. Each trace then links user actions to the protocol messages that were produced.

From this trace corpus, the agent constructs an **executable model** of the protocol. The model can generate messages and predict responses. It also has internal guesses for protocol fields and other protocol features when the traces do not provide enough information.

To decide **what to do next**, the agent selects an experiment expected to get more missing information without costing too much. It can collect another trace or look at the binary, or it could try fixing code it wrote. The choices made by the agent affect how long this overall process will take and how much it costs.

Each selected experiment is **evaluated with a behavior oracle**. The agent compares the same operation on the original system and candidate. The oracle can compare semantic packet batches (as in AirPort) or other semantic behavior. It should ignore some irrelevant nondeterminism like timestamps and identify only meaningful differences.

The agent **updates the protocol model and replacement implementation**, then works on fixing the remaining differences relative to the original. The agent updates a coverage map of workflows before starting the loop again.

The loop ends with **a convergence assessment** and a human audit. Coverage and confidence remain research challenges, discussed next.

5 Research Challenges

Reliable recovery requires traces and validation beyond generating many implementations. Prior work on protocol recovery and fuzzing provides a foundation. Discoverer [17] infers message formats from network traces, while NetPlier [58] uses probabilistic inference to recover message types and protocol structure. DynPRE [37] actively interacts with protocol servers to gather additional evidence and refine protocol inference. ChatAFL [41] uses LLMs to generate protocol messages and explore stateful protocol implementations, while LLMIF [54] uses LLMs to extract protocol information and guide IoT-device fuzzing. Complementary efforts include REA [43], which connects AI agents to binary-analysis and application-inspection tools, and PhotoCraft [6], a Rust reimplementation of Photoshop functionality built from public specifications and observed behavior.

These projects provide tooling and examples for inspecting and reconstructing software, while device-management recovery also requires validating protocols and their effects on physical devices. Our focus is on combining such inference and exploration with implementation and behavioral validation to recover user workflows.

Seven challenges remain before this process can operate with limited human supervision.

1. Coverage. Ordinary traces are biased toward common, successful workflows and omit the states most likely to expose an incomplete model. Random protocol fuzzing can expand the corpus, but often produces invalid messages without reaching meaningful behavior [7]. An AI-guided system could instead generate user workflows, vary one condition at a time, and target uncertain regions of its current model. Traces from real users can also expose unexpected behaviors. The open problem is measuring coverage of user workflows beyond packet or code-path counts.

2. Behavioral oracles. Oracles must cheaply assess correctness from traces and device behavior, ignoring irrelevant nondeterminism without hiding semantic differences. Manually constructing oracles can require the same cost as protocol recovery itself. A practical system must derive oracles from repeated executions and existing interfaces, coping with partial information when necessary.

3. Experiment planning. If the replacement protocol has a bug, the agent needs to either change the code it wrote or collect more traces if it needs more information. These choices affect how long the process will take and how many resources are consumed.

4. Faithful test environments. Test environments and emulators can gracefully isolate destructive experiments such as erasing a disk or making a purchase, but building these requires first exercising this behavior on a real device. It is challenging to incrementally build these when destructive actions are necessary in the process.

5. Efficiency. Experiments may require restoring state, waiting for timeouts [49], or accessing scarce hardware. A recovery system should reuse traces rather than repeat work after every model revision. Deterministic replay and caching can preserve normalized traces and prior oracle outcomes, while independent experiments can run in parallel when they do not share device or cloud state. The challenge is deciding which results remain valid after model changes.

6. Confidence and convergence. Matching every collected trace does not establish that a re-implementation is correct because the trace corpus and oracle can both be incomplete. Confidence should therefore be framed in units of individual behaviors rather than for the protocol as a whole. Test-only traces and bugs across different device versions can be used to expose when the model is overfit to too few traces. But we can only be confident that the behavior is correct for features we have traced extensively.

7. Responsible recovery. While a process intended for letting owners control their own devices could also be misused by attackers, the research community should distinguish between these. One reasonable direction is to focus on building clean-room implementations of device applications. Vendors providing local control remains preferable, but AI-assisted recovery may be used otherwise.

6 When Support Ends, Control Should Transfer

An end-of-life control-transfer principle. Vendors may stop supporting devices, but they should provide an exit path for features that owners or third parties can sustain independently. They can preserve local operation, publish documentation, and transfer the authority needed by replacement software. Vendors already possess protocol knowledge and authority that independent recovery may never obtain.

Incentives and enforcement. Vendor cooperation cannot be assumed because preserving independent operation may conflict with revenue from subscriptions or replacement purchases. While voluntary vendor commitments can provide individual exit paths, and standards can specify interoperable interfaces and ways to transfer control, neither fixes the incentive to withhold these capabilities. We therefore advocate for enforceable regulation requiring vendors to provide an end-of-life path.

Disclosure for cloud features. When a feature cannot reasonably be sustained independently, vendors should disclose that prominently before sale. Disclosure should identify the guaranteed support period and any features that will stop working. These disclosures help with purchasing decisions but are not a solution for preserving features.

Vendor obligations are still limited. Some features depend on remote infrastructure and other ongoing expenses. As Section 3 explains, transferring authority alone cannot replace missing cloud services. Vendors cannot be required to provide perpetual free service, but they should be required to transfer control when they do end service.

The AirPort work demonstrates that a local management application can sometimes be recovered without support from vendors. However, vendors can withhold control, so recovery is not possible in all cases. The systems community should develop ways to recover protocols and transfer control, while vendors, standards bodies, and policymakers should make exit paths the default.

7 Conclusion

Cloud-tethered devices should not become e-waste merely because vendor support ends. Vendors need not operate services forever, but should support local operation and release the information and authority required by replacement software. Foundation models make independent recovery increasingly practical, but dependable de-cloudification still requires better trace coverage, behavior oracles, and convergence confidence. AI-assisted recovery should remain a fallback, but not a substitute for deliberate end-of-life design. Support may end, but control should transfer.

8 Acknowledgments

We thank our anonymous reviewers, David Mazières, Christos Kozyrakis, Ryan Stutsman, Kostis Kaffes, Mendel Rosenblum, and our shepherd, Maria Apostolaki, for their helpful feedback.

References

[1] Apple, Inc. 2015. *The OS X Accessibility Model.* https://developer.apple.com/library/archive/documentation/Accessibility/Conceptual/AccessibilityMacOSX/OSXAXmodel.html

[2] Apple Inc. 2024. *Download AirPort Utility 5.6.1 for Windows.* https://support.apple.com/en-us/106400

[3] Apple Inc. 2024. *Download AirPort Utility 6.3.1 for Mac.* https://support.apple.com/en-us/106429

[4] Apple Inc. 2026. *AirPort Utility.* https://apps.apple.com/us/app/airport-utility/id427276530

[5] Apple, Inc. n.d. *AirPlay.* https://www.apple.com/airplay/

[6] ArtCraft. 2026. *PhotoCraft.* https://github.com/storytold/photocraft

[7] Jinsheng Ba, Marcel Böhme, Zahra Mirzamomen, and Abhik Roychoudhury. 2022. Stateful Greybox Fuzzing. In *31st USENIX Security Symposium (USENIX Security 22).* USENIX Association, Boston, MA, 3255–3272. https://www.usenix.org/conference/usenixsecurity22/presentation/ba

[8] Belkin. 2026. *Wemo Support Ending - What You Need to Know.* https://www.belkin.com/support-article/?articleNum=335419

[9] Tristan Benoit, Yunru Wang, Moritz Dannehl, and Johannes Kinder. 2025. BLens: contrastive captioning of binary functions using ensemble embedding. In *Proceedings of the 34th USENIX Conference on Security Symposium* (Seattle, WA, USA) *(SEC '25).* USENIX Association, USA, Article 353, 20 pages.

[10] Ruven Brooks. 1983. Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies* 18, 6 (1983), 543–554. https://doi.org/10.1016/S0020-7373(83)80031-5

[11] Kevin Burk, Fabio Pagani, Christopher Kruegel, and Giovanni Vigna. 2022. Decomperson: How Humans Decompile and What We Can Learn From It. In *31st USENIX Security Symposium (USENIX Security 22).* USENIX Association, Boston, MA, 2765–2782. https://www.usenix.org/conference/usenixsecurity22/presentation/burk

[12] James Calligeros. 2026. *Progress Report: Linux 6.19.* https://asahilinux.org/2026/02/progress-report-6-19/

[13] Ying Cao, Runze Zhang, Ruigang Liang, and Kai Chen. 2024. Evaluating the Effectiveness of Decompilers. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024).* Association for Computing Machinery, New York, NY, USA, 491–502. https://doi.org/10.1145/3650212.3652144

[14] ClassAction.org. 2026. *Belkin Wemo End of Life Arbitration: Affected by Discontinued Belkin Wemo Support?* https://www.classaction.org/belkin-wemo-bricking-arbitration-lawsuit-info

[15] Juli Clover. 2024. *Spotify's Car Thing Accessory is Officially Dead.* https://www.macrumors.com/2024/12/10/spotify-car-thing-officially-dead/

[16] Juli Clover. 2026. *Apple Dropping AirPort Utility From the App Store With iOS 27.* https://www.macrumors.com/2026/06/22/apple-airport-utility-ios-27/

[17] Weidong Cui, Jayanthkumar Kannan, and Helen J. Wang. 2007. Discoverer: Automatic Protocol Reverse Engineering from Network Traces. In *16th USENIX Security Symposium (USENIX Security 07).* USENIX Association, Boston, MA. https://www.usenix.org/conference/16th-usenix-security-symposium/discoverer-automatic-protocol-reverse-engineering-network

[18] Cristian Curaba, Denis D'Ambrosi, Alessandro Minisini, and Natalia Pérez-Campanero Antolín. 2024. CryptoFormalEval: Integrating Large Language Models and Formal Verification for Automated Cryptographic Protocol Vulnerability Detection. In *The First Workshop on System-2 Reasoning at Scale, NeurIPS'24.* https://openreview.net/forum?id=hqvJT2pOxu

[19] D. Taylor and T. Wu and N. Mavrogiannopoulos and T. Perrin. 2007. *Using the Secure Remote Password (SRP) Protocol for TLS Authentication.* https://datatracker.ietf.org/doc/html/rfc5054

[20] Enrique Dans. 2025. *Samsung's ad-powered fridge: a masterclass in how to alienate your best customers.* https://medium.com/enrique-dans/samsungs-ad-powered-fridge-a-masterclass-in-how-to-alienate-your-best-customers-089d2856aaac Medium, accessed July 16, 2026.

[21] E. Rescorla. 1999. *Diffie-Hellman Key Agreement Method.* https://datatracker.ietf.org/doc/html/rfc2631

[22] eero. n.d. *What is eero Plus?* https://support.eero.com/hc/en-us/articles/115002766063-What-is-eero-Plus

[23] eero. n.d. *What is Wifi Radio Analytics?* https://support.eero.com/hc/en-us/articles/15282384606107-What-is-Wifi-Radio-Analytics

[24] Chongzhou Fang, Ning Miao, Shaurya Srivastav, Jialin Liu, Ruoyu Zhang, Ruijie Fang, Asmita, Ryan Tsang, Najmeh Nazari, Han Wang, and Houman Homayoun. 2024. Large language models for code analysis: do LLMs really do their job?. In *Proceedings of the 33rd USENIX Conference on Security Symposium* (Philadelphia, PA, USA) *(SEC '24).* USENIX Association, USA, Article 47, 18 pages.

[25] Arlo Gilbert. 2016. *The time that Tony Fadell sold me a container of hummus.* https://arlogilbert.com/the-time-that-tony-fadell-sold-me-a-container-of-hummus-cb0941c762c1

[26] Gadget Hacks. 2025. *Apple Locks iPhones into iOS 26 - No Downgrade Possible.* https://apple.gadgethacks.com/news/apple-locks-iphones-into-ios-26-no-downgrade-possible/

[27] Karol Herbst. 2018. Karol Herbst: Nouveau - reverse engineering Nvidia GPUs. DevConf 2018, YouTube. https://www.youtube.com/watch?v=-7SdKBUrKJ0

[28] Alex Hern. 2016. *Revolv devices bricked as Google's Nest shuts down smart home company.* https://www.theguardian.com/technology/2016/apr/05/revolv-devices-bricked-google-nest-smart-home

[29] Peter Holderith. 2022. *BMW Responds to Fury Over Heated Seats Subscription Fee.* https://www.thedrive.com/news/bmw-responds-to-fury-over-heated-seats-subscription-fee

[30] Byron Hurd. 2026. *BMW Commits to Subscriptions Even After Heated Seat Debacle.* https://www.thedrive.com/news/bmw-commits-to-subscriptions-even-after-heated-seat-debacle

[31] Michaela Mihaljević Jakić. 2026. *The SwiftUI Oracle: Measuring a Clean Room Against the Real Thing.* https://aleahim.com/blog/the-swiftui-oracle/

[32] Jiayi Jiang, Xiyuan Zhang, Chengcheng Wan, Haoyi Chen, Haiying Sun, and Ting Su. 2024. BinPRE: Enhancing Field Inference in Binary Analysis Based Protocol Reverse Engineering. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (Salt Lake City, UT, USA) *(CCS '24)*. Association for Computing Machinery, New York, NY, USA, 3689–3703. https://doi.org/10.1145/3658644.3690299

[33] K. Moriarty and B. Kaliski and A. Rusch. 2017. *PKCS #5: Password-Based Cryptography Specification Version 2.1.* https://datatracker.ietf.org/doc/html/rfc8018

[34] Donghyun Kim, Zichao Hu, Joydeep Biswas, Aditya Akella, and Daehyeok Kim. 2026. Mimesys: Generating Realistic Executable Testing Environments from Resource Usage Traces. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*. USENIX Association, Seattle, WA, 2205–2221. https://www.usenix.org/conference/osdi26/presentation/kim-donghyun

[35] Boaz Lavon, Shahar Katz, and Lior Wolf. 2025. Execution Guided Line-by-Line Code Generation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems.* https://openreview.net/forum?id=ySFDPoiANu

[36] Linux Wireless Developers. n.d. *Broadcom brcmsmac (PCIe) and brcmfmac (SDIO/USB) Drivers.* https://wireless.docs.kernel.org/en/latest/en/users/drivers/brcm80211.html

[37] Zhengxiong Luo, Kai Liang, Yan-Yang Zhao, Feifan Wu, Junze Yu, He-Yuan Shi, and Yu Jiang. 2024. DynPRE: Protocol Reverse Engineering via Dynamic Inference. *Proceedings 2024 Network and Distributed System Security Symposium* (2024). https://api.semanticscholar.org/CorpusID:267623524

[38] man7.org. 2026. *tcpdump(1) — Linux manual page.* https://man7.org/linux/man-pages/man1/tcpdump.1.html

[39] Alessandro Mantovani, Simone Aonzo, Yanick Fratantonio, and Davide Balzarotti. 2022. RE-Mind: a First Look Inside the Mind of a Reverse Engineer. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 2727–2745. https://www.usenix.org/conference/usenixsecurity22/presentation/mantovani

[40] Dylan Manuel, Nafis Tanveer Islam, Joseph Khoury, Ana Nunez, Elias Bou-Harb, and Peyman Najafirad. 2024. Enhancing Reverse Engineering: Investigating and Benchmarking Large Language Models for Vulnerability Analysis in Decompiled Binaries. arXiv:2411.04981 [cs.CR] https://arxiv.org/abs/2411.04981

[41] Ruijie Meng, Martin Mirchev, Marcel Böhme, and Abhik Roychoudhury. 2024. Large Language Model guided Protocol Fuzzing. In *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/large-language-model-guided-protocol-fuzzing/

[42] Jowi Morales. 2026. *Samsung's disastrous firmware update bricks refrigerators ahead of South Korean holiday — repair visits slip into October, customers complain of food spoilage ahead of three-day celebration.* https://www.tomshardware.com/service-providers/samsungs-disastrous-firmware-update-breaks-refrigerators-ahead-of-south-korean-holiday-customers-complain-of-non-functioning-units-food-spoilage-ahead-of-three-day-celebration

[43] Morluto. 2026. *REA: Reverse Engineer Anything.* https://github.com/morluto/rea

[44] National Security Agency. 2026. *ghidra.* https://github.com/nationalsecurityagency/ghidra

[45] OpenAI. 2026. *Introducing GPT-5.5.* https://openai.com/index/introducing-gpt-5-5/

[46] OpenAI. n.d. *Codex.* https://openai.com/codex/

[47] Thiyagu Palanisamy and Chandirasekar Thiagarajan. 2025. *Blackbox reverse engineering: Rebuilding without source code — XConf India 2025.* https://www.youtube.com/watch?v=Vl1yHlBuAlk

[48] Sasha Rogelberg. 2025. *Samsung confirms it will begin showing you advertisements on your $1,800-plus refrigerator's screen.* https://fortune.com/2025/09/19/samsung-family-hub-refrigerators-advertisements/

[49] Sergej Schumilo, Cornelius Aschermann, Andrea Jemmett, Ali Abbasi, and Thorsten Holz. 2022. Nyx-net: network fuzzing with incremental snapshots. In *Proceedings of the Seventeenth European Conference on Computer Systems* (Rennes, France) *(EuroSys '22)*. Association for Computing Machinery, New York, NY, USA, 166–180. https://doi.org/10.1145/3492321.3519591

[50] The Apple Wiki Contributors. 2026. *SHSH.* https://theapplewiki.com/wiki/SHSH

[51] The LLDB Team. 2026. *The LLDB Debugger.* https://lldb.llvm.org/

[52] Tom Wu. n.d. *SRP Protocol Design.* http://srp.stanford.edu/design.html

[53] Daniel Votipka, Seth Rabin, Kristopher Micinski, Jeffrey S. Foster, and Michelle L. Mazurek. 2020. An Observational Investigation of Reverse Engineers' Processes. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 1875–1892. https://www.usenix.org/conference/usenixsecurity20/presentation/votipka-observational

[54] Jincheng Wang, Le Yu, and Xiapu Luo. 2024. LLMIF: Augmented Large Language Model for Fuzzing IoT Devices. In *2024 IEEE Symposium on Security and Privacy (SP)*. 881–896. https://doi.org/10.1109/SP54263.2024.00211

[55] Anjiang Wei, Tarun Suresh, Jiannan Cao, Naveen Kannan, Yuheng Wu, Kai Yan, Thiago S. F. X. Teixeira, Ke Wang, and Alex Aiken. 2025. CodeARC: Benchmarking Reasoning Capabilities of LLM Agents for Inductive Program Synthesis. arXiv:2503.23145 [cs.PL] https://arxiv.org/abs/2503.23145

[56] Haiyang Wei, Ligeng Chen, Zhengjie Du, Yuhan Wu, Haohui Huang, Yue Liu, Guang Cheng, Fengyuan Xu, Linzhang Wang, and Bing Mao. 2025. Unleashing the Power of LLM to Infer State Machine From the Protocol Implementation. In *2025 IEEE/ACM 33rd International Symposium on Quality of Service (IWQoS)*. 1–10. https://doi.org/10.1109/IWQoS65803.2025.11143461

[57] x56 (Vince Cali). 2017. AirPyrt Tools. https://github.com/x56/airpyrt-tools.

[58] Yapeng Ye, Zhuo Zhang, Fei Wang, X. Zhang, and Dong-Yan Xu. 2021. NetPlier: Probabilistic Network Protocol Reverse Engineering from Message Traces. *Proceedings 2021 Network and Distributed System Security Symposium* (2021). https://api.semanticscholar.org/CorpusID:231877711