Architectural VPN Vulnerabilities, Disclosure Fatigue, and Structural Failures

William J. Tolley
Hampden-Sydney College
Breakpointing Bad
william@breakpointingbad.com

Everett Morse
Hampden-Sydney College
morsee29@hsc.edu

Gabriel Hogan
Washington & Lee University
ghogan@mail.wlu.edu

Jeffrey Knockel
Bowdoin College
j.knockel@bowdoin.edu

Jedidiah R. Crandall
Arizona State University
Breakpointing Bad
jed@breakpointingbad.com

Abstract

This experience paper recounts seven years of disclosure and re-testing of an architectural VPN vulnerability first reported in 2019. The flaw, rooted in predictable tunnel behavior, still allows blind in-path adversaries to infer and disrupt encrypted traffic on fully updated devices in 2025. Our experience shows that repeated CVEs and patch cycles create the illusion of progress while the underlying risk persists. We distill lessons about the limits of patch-based disclosure, the absence of ownership for architectural flaws, and the resulting risks to high-threat users, and propose a framework for tracking long-lived, cross-vendor vulnerabilities.

1 Introduction: Unacknowledged and Unresolved Vulnerabilities

This paper draws on seven years of coordinated disclosure, vendor engagement, and re-testing surrounding a class of architectural VPN vulnerabilities first reported in 2019. These vulnerabilities allow blind in/on-path adversaries—attackers who can observe and inject packets but cannot decrypt their contents—to infer internal VPN IP addresses, identify the existence of active encrypted connections, and inject or reset TCP streams without needing to decrypt any traffic [47]. These attacks exploit fundamental architectural behaviors, including metadata exposure, predictable NAT and routing logic, and uniform tunnel response behavior, rather than implementation flaws or cryptographic weaknesses.

The vulnerability class was first disclosed on the oss-security mailing list [46] and later assigned CVE-2019-9461 [28] and CVE-2019-14899 [27]; subsequent peer-reviewed work characterized both client- and server-side variants [47]. Vendors treated the client-side variant as a patchable defect and the server-side variant as "out of scope." Over the following years, partial fixes appeared, new CVEs were issued, and the same behavior resurfaced [4–6, 16, 18, 27, 29]. No party claimed ownership of the underlying architectural condition.

These observations are less a new vulnerability discovery than evidence of a governance failure. Current disclosure infrastructure, especially the CVE system, cannot track cross-vendor, architectural flaws. Once a CVE is closed, persistent risks vanish from public view, giving the appearance of progress while exploitability endures. For the Internet-freedom community, which routinely promotes VPNs and related tools to users under surveillance, this gap translates directly into human risk.

The remainder of this paper reflects on lessons drawn from long-term disclosure, mitigation attempts, and re-evaluation. We examine how architectural vulnerabilities interact with existing reporting workflows, trace recurring patterns of fragmentation and responsibility shifting, and assess whether modern VPN deployments meaningfully mitigate the original attack primitives.

Contributions. This paper makes three primary contributions. First, we present an experience report documenting seven years of disclosure and vendor interaction surrounding an architectural VPN vulnerability. Second, we demonstrate through re-testing on fully updated Android systems in 2025 that core attack primitives remain exploitable across OpenVPN, WireGuard, and proprietary VPN protocols despite multiple CVEs and patch cycles. Third, we distill lessons about the limits of patch-based disclosure and propose a complementary framework for tracking long-lived, cross-vendor architectural vulnerabilities.

2 Background: The Nature of Architectural Attacks

Before turning to the disclosure timeline, we briefly summarize the technical background of the architectural VPN vulnerabilities that motivated this study.

Architectural VPN vulnerabilities, such as blind in/on-path traffic inference attacks, differ fundamentally from traditional security flaws. Unlike memory-safety, cryptographic, or configuration bugs [1, 9, 11, 21, 24, 32], these attacks exploit stable and predictable design properties of VPNs: namely, traffic metadata exposure (*e.g.*, packet size, timing, and direction), deterministic NAT behavior, and tunnel routing assumptions.

The adversary model in these attacks assumes a passive observer with the ability to inject spoofed traffic. Positioned between the client and VPN server, such an adversary may be a local network operator (*e.g.*, a malicious access point), a surveillance node at an ISP or exchange, or a state-aligned infrastructure provider [50]. These adversaries do not require packet decryption; they operate solely on metadata and system-level behavior.

Free and Open Communications on the Internet 2026(1), 48–57

Under this model, adversaries can reliably carry out the following types of attacks:

- **DNS Hijacking:** Redirecting or spoofing DNS queries without access to their plaintext contents.
- **TCP Injection and Reset:** Brute-forcing ephemeral port and sequence space to infer and terminate encrypted TCP sessions.
- **Client Activity Surveillance:** Using fine-grained metadata analysis to infer user activity patterns despite encrypted tunnels [51].

At a high level, the attack operates by sending carefully crafted spoofed packets into the VPN tunnel and observing the system's reaction. If a probe matches an active flow, due to predictable NAT assignment or routing logic, the endpoint often produces a distinct encrypted response (*e.g.*, reset, retransmission, or measurable timing/size differences). By observing these responses externally, an adversary can infer whether a connection to a given destination exists and identify internal VPN-assigned IPs; furthermore, by iteratively probing and observing sequence/ack behavior, the adversary can discover the correct sequence and acknowledgment window and subsequently inject, hijack, or terminate an active TCP session.

These techniques have been demonstrated effective across a range of VPN implementations (*e.g.*, OpenVPN, WireGuard, IPSec) and operating systems, including those configured with strong cryptographic defaults [47]. Crucially, the attack does not rely on timing jitter [41], amplification effects [34], or reflection behavior such as IPID-based inference [2, 14, 19, 25]; rather, it succeeds by exploiting stable, predictable infrastructure design.

Because these vulnerabilities arise from architectural constraints rather than implementation bugs, and therefore exhibit systemic behavior across platforms and vendors as discussed in Section 4, they are not easily patched. Comprehensive mitigations would require cross-layer redesigns with substantial usability and deployment trade-offs. This places them outside the scope of most traditional vulnerability-disclosure processes, which are calibrated to track discrete software flaws rather than long-lived, systemic conditions.

The persistence of these behaviors across protocols and platforms became a central theme of our seven-year disclosure experience, detailed in Section 3, and the lessons derived from it, discussed in Section 4.

2.1 Vendor Response Dynamics

The long-term reporting history for this class of vulnerabilities reflects a wide range of vendor responses, shaped in part by the difficulty of categorizing and resolving architectural issues. Across documented disclosure efforts, vendor behavior ranged from responsive and transparent to opaque or inconclusive [13, 47]. *These contrasts are not evaluations of individual vendors, but observations of how existing disclosure workflows fragment responsibility for cross-layer flaws.* While limited client-side mitigations emerged for specific systems and configurations, the underlying architectural condition—particularly the server-side variant—remained unacknowledged and unpatched across all tested platforms.

Responses to the client-side vulnerability varied substantially. Google issued multiple Android patches and addressed the issue under CVE-2019-9461, CVE-2019-14899, and later CVE-2024-49734 following re-disclosure [27–29]. Apple initially claimed a fix in iOS 13.6 [6], and later released additional networking changes in iOS 17.2.1 [7]. Because Apple's networking stack is not externally inspectable, independent verification of architectural changes is inherently limited. In contrast, WireGuard acknowledged the architectural nature of the issue and proposed configuration-level mitigations that reduced exposure in some environments [18], while OpenBSD issued an early patch adjusting default interface filtering rules [17].

Other vendors declined to engage meaningfully. NordVPN did not acknowledge the vulnerability across multiple disclosure attempts, and neither its WireGuard-based NordLynx protocol nor its later proprietary protocol, NordWhisper, resisted the same inference attacks [35]. OpenVPN responded early in the original disclosure process but ultimately took no mitigating action [37]. Detailed validation results for these findings are presented in Section 5.

Taken together, these interactions indicate that the persistence of the vulnerability reflects a disclosure ecosystem unable to assign or sustain ownership for architectural conditions that span products and platforms. The client-side variant has seen only partial, platform-specific mitigation, while the server-side attack remains unmitigated and untracked across all examined systems. A detailed timeline of vendor interactions and disclosure outcomes is provided in Section 3.4.

2.2 Structural Gaps in the Vulnerability Ecosystem

From these vendor interactions, several broader lessons emerge about the structure of vulnerability disclosure itself. The failures documented in vendor triage patterns are not merely anecdotal or organizational, but point to deeper limitations in how architectural vulnerabilities are handled within existing disclosure systems.

These varied responses underscore a core challenge: architectural vulnerabilities do not map cleanly onto conventional triage processes. Traditional vulnerability workflows, centered around CVEs, vendor advisories, and patch rollouts, are optimized for discrete implementation errors with bounded scope and clear product ownership. Architectural flaws, by contrast, often span multiple abstraction layers, emerge from systemic interactions, and persist across vendor boundaries. They evade patch closure logic not because they are technically obscure, but because no single stakeholder owns the full exploit surface or bears responsibility for the conditions that make the vulnerability possible.

In several cases documented in prior work [47], vendors treated the issue as "out of scope" or "non-actionable," even when provided with detailed proofs-of-concept. Some implemented partial mitigations narrowly scoped to specific platforms or configurations, while others deferred entirely, citing external causes such as OS-level routing logic. This is not a failure of individual vendors but a failure of the vulnerability ecosystem itself, one that is structurally incapable of accommodating flaws not tied to a single patch, product, or protocol.

The proposed Internet Freedom vulnerability registry responds to this systemic limitation. Rather than relying on one-off disclosures or vendor-specific timelines, it aims to create a persistent, multi-stakeholder tracking model for long-lived vulnerabilities, including those that are architectural in nature. It provides a venue for documenting cross-vendor patterns, partial mitigations,

re-emergent behaviors, and affected threat models over time. Crucially, it centers risk around user impact, especially for high-risk populations operating in censored or adversarial environments, rather than code changes or release cycles. By treating design-level flaws as persistent features of the security landscape rather than transient bugs, the Internet Freedom registry offers a framework for long-term accountability, coordinated mitigation, and transparency that conventional systems struggle to provide.

3 Seven Years of Disclosure: An Experience Report

The following chronology summarizes how the structural gaps discussed above manifested in practice. Figure 1 provides a consolidated view of the disclosure history covered in this section, including initial reports, CVE assignments, vendor responses, and subsequent re-validation.

This section traces the disclosure and response process over nearly seven years, involving repeated reporting, varied vendor engagement, a peer-reviewed publication, and ongoing experimentation. Rather than relitigating technical details, the goal is to illustrate how existing disclosure mechanisms repeatedly failed to resolve or contain an architectural vulnerability over time.

3.1 Initial Disclosure and Assigned CVEs (2018–2019)

The vulnerability class was first disclosed to major VPN providers and operating system vendors in late 2019. Public and vendor attention focused almost entirely on the client-side variant, in which a local network adversary could infer active connections or disrupt encrypted TCP streams. Two CVEs were assigned at the time: CVE-2019-9461 (affecting Android) and CVE-2019-14899 (affecting Linux and other Unix-like systems).

Importantly, these CVEs captured only client-side manifestations of the attack. The more general server-side variant, where blind in/on-path adversaries exploit correct-but-predictable NAT and routing behavior, was explicitly deemed ineligible for CVE assignment. In correspondence with MITRE [30], we were informed that CVE identifiers apply to specific software implementation mistakes, not to cross-vendor architectural behaviors or attack methodologies, particularly where no single codebase or patchable defect can be identified. Under this interpretation, MITRE stated that CVE-2019-14899 itself should not have been assigned, despite being scoped to the client-side attack. As a result, the server-side attack surface was left untracked despite being disclosed concurrently and remaining exploitable in practice.

At the time, the issue was publicly disclosed through blogs, press reports, and conference presentations [8, 12, 23, 26, 42–45, 47]. Some vendors issued partial mitigations focused on interface filtering or packet validation (*e.g.*, OpenBSD [17], WireGuard [18], DD-WRT [15], Oracle [38], Fortinet [20]), while others deferred, minimized, or dismissed the issue entirely (*e.g.*, OpenVPN [37], ProtonVPN [40], Mullvad [33], Private Internet Access [39]). These early exchanges set the pattern for what followed: limited, inconsistent mitigations and no durable ownership of the underlying architectural condition.

3.2 Peer-Reviewed Publication and Vendor Re-engagement (2021)

In prior work, we presented both client- and server-side variants of the attack in a peer-reviewed paper at USENIX Security 2021 [47], emphasizing that the vulnerabilities were not implementation-specific but instead arose from cross-layer design decisions common across VPN architectures.

Following the publication, we again contacted vendors with additional technical material and a refined proof-of-concept. Responses were mixed. Some vendors claimed the issue had been resolved, despite continued evidence of exploitability. Others reopened triage discussions but failed to implement effective mitigations. Given the multi-year timescale of this vulnerability, institutional memory is not maintained across disclosure rounds; reports re-enter vendor intake pipelines rather than being carried forward through a consistent point of contact or engineering team, effectively restarting the process.

3.3 Re-reporting and New CVEs (Post-2021)

Between 2020 and 2024, we repeatedly re-tested across devices and operating systems, marking a shift from the initial 2019–2021 disclosure cycle to sustained longitudinal evaluation in which the same architectural flaws resurfaced. Notably, no new CVEs were assigned between CVE-2019-14899 and CVE-2024-49734 despite continued reporting during this period. CVE-2024-49734 was eventually assigned following confirmation of a working attack on a fully updated Pixel 8 Pro, addressing a report originally submitted in December 2021; this multi-year delay is reflected in the January 2025 Android Security Bulletin [5] and corresponding Android networking code history [3]. Despite intervening patch attempts, the same attack pattern remained effective.

Following this disclosure, we conducted further testing and confirmed that the vulnerability remains exploitable on a Pixel 10 Pro XL running Android 16 with October 2025 security updates. The same inference primitives—internal IP discovery, connection-state inference, and sequence/acknowledgment window scanning—remain viable despite CVE-2024-49734 having been formally closed. These findings were re-submitted to the Android Security Team, which acknowledged continued exploitability and confirmed that a new CVE assignment is pending.

This pattern reflects a broader systemic issue across disclosure cycles. When vulnerabilities fall outside conventional patch domains, such as the server-side variant or related attacks like port shadowing [31] and tunnel cracking [49], they may receive CVE identifiers without meaningful remediation. When considered in-scope, patches tend to be narrowly scoped, with new CVEs assigned each time the same architectural behavior resurfaces. The resulting record suggests technical progress without materially reducing exploitability, illustrating how architectural vulnerabilities evolve administratively rather than technically.

3.4 Vendor Coordination and Divergent Responses

Vendor coordination around this class of vulnerabilities proved inconsistent and short-lived. Each actor operated within its own

disclosure and patch workflow, producing fragmented outcomes rather than collective resolution.

Google participated in multiple patch cycles following the original report and a subsequent re-disclosure, including live testing, and issued CVE-2024-49734 in response. Despite mitigation attempts associated with CVE-2019-9461, CVE-2019-14899, and CVE-2024-49734, the vulnerability remains reproducible on fully updated Android 16 systems. Continued exploitability corresponds to an additional CVE currently pending based on our continued re-testing [5, 27–29].

Apple stated that the issue was addressed in iOS 13.6, noting that "a routing issue was addressed with improved restrictions" [6]. Using the same unmodified probing methodology described in prior work [47], the original attack sequence continued to succeed on subsequent iOS releases. Starting with iOS 17.2.1, however, the attack no longer succeeded [7]. Due to the closed nature of Apple's networking stack, deeper validation of whether the underlying architectural condition was resolved remains infeasible.

OpenVPN was among the earliest vendors contacted in 2019. The company ultimately declined to accept responsibility for the vulnerability, stating publicly that it had found no flaws in the OpenVPN software itself and characterizing the issue as a consequence of operating system behavior rather than the VPN implementation [37].

WireGuard responded promptly and acknowledged the architectural nature of the issue. The maintainer proposed configuration-level mitigations that reduced exposure to the client-side attack in some environments but did not fully eliminate the vulnerability and, by design, could not address the server-side variant. The project remained communicative throughout the process and publicly documented its response.

NordVPN was contacted at least four times between 2019 and 2024 regarding both client- and server-side variants and declined to engage. Although NordLynx is based on WireGuard, NordVPN did not adopt its mitigations. Their later protocol, NordWhisper, promoted as a circumvention tool, also remains vulnerable to the same inference primitives [35].

Across all tested systems, the server-side variant, where blind in/on-path adversaries can inject spoofed packets that are indistinguishable from legitimate encrypted traffic, remains unacknowledged and unmitigated. *Because the vulnerability originates in core networking behavior, meaningful remediation ultimately depends on operating-system vendors such as Google and Apple. When those platforms leave the underlying tunnel semantics unchanged, downstream projects, including WireGuard, OpenVPN, and commercial VPN providers, cannot fully resolve the issue regardless of configuration-level mitigations.* No vendor has proposed a viable solution or included this variant in formal remediation guidance. The persistence of this untracked attack surface underscores how architectural flaws fall outside the scope of conventional patch ownership.

3.5 Narrative Gaps and the Absence of Authority

Disclosure without a clear authority leaves a vacuum that is quickly filled by vendors and secondary commentary, often before peer-reviewed publication can establish an accurate technical record. In the case of CVE-2019-14899 [27], vendor statements and media coverage displaced the original disclosure, shaping public perception early and persistently. ProtonVPN characterized the issue as "a narrow guessing attack" unsuitable for mass surveillance [40]; OpenVPN disclaimed responsibility by attributing the behavior to operating-system design [37]; and Mullvad framed the vulnerability as largely irrelevant under default configurations [33]. These framings were amplified by blogs and security podcasts [22], becoming the dominant narrative despite diverging from the underlying technical evidence. This episode illustrates a broader pattern: when architectural vulnerabilities lack an authoritative, persistent reference point, their credibility and perceived severity are determined less by technical analysis than by narrative control, underscoring the need for stronger civil-society participation to provide independent validation and continuity.

4 Lessons Learned from Long-Term Disclosure

In this section, we distill recurring patterns and insights drawn from the seven years of disclosure activity surrounding this vulnerability class. Each lesson reflects a specific obstacle observed repeatedly across vendors, systems, and time. Together, they describe a cycle in which architectural flaws are discovered, acknowledged, patched in narrow contexts, and then forgotten, only to reappear under new names or CVE identifiers. The goal is not to assign blame, but to understand how current vulnerability processes systematically erase persistence and prevent collective learning. These lessons ground the framework proposed in Section 6, translating repeated failure into concrete design requirements for how long-lived vulnerabilities should be tracked and communicated.

Why This Vulnerability Is Systemic. We use the term *systemic* to emphasize that this vulnerability does not arise from a single faulty implementation, configuration mistake, or protocol bug. Instead, it emerges from widely shared design assumptions about how VPN tunnels interact with operating-system networking stacks: exposure of traffic metadata (*e.g.*, packet size, timing, and direction), predictable NAT behavior, and tunnel routing and response semantics under adversarial probing. These properties are present across protocols, platforms, and vendors, and therefore cannot be eliminated by patching any single codebase. As a result, the vulnerability persists even as specific implementations change, making it a recurring feature of the ecosystem rather than an isolated defect.

4.1 Lesson 1: Architectural Flaws Defy CVE Closure

Architectural vulnerabilities are not easily routed through conventional triage pipelines. They are too general to be tied to a single product, too persistent to be closed with a patch, and too cross-cutting to belong to any single team. Vulnerability tracking systems like CVE do not accommodate long-tail re-emergence or design-level risks. Once a vulnerability is cataloged and assigned, its "resolved" status becomes the default, regardless of whether it remains exploitable in practice.

4.2 Lesson 2: Vendor Turnover Erases Institutional Memory

Over multi-year timelines, disclosure efforts must often be restarted from scratch. Engineering and security teams change, institutional context disappears, and previously acknowledged issues resurface as new reports. Each re-reporting round repeats triage, verification, and internal coordination, but rarely builds cumulative knowledge.

4.3 Lesson 3: Media Narratives Shape Disclosure Outcomes

As discussed in Section 3.5, media coverage and secondary commentary reshaped the vulnerability's framing in ways that undermined both peer review and coordinated response. Some reviewers echoed misconceptions from non-technical sources, while vendors cited public narratives as justification for limited action. The credibility of design-level vulnerabilities can hinge less on evidence than on how they are represented in public discourse.

4.4 Lesson 4: Partial Mitigations Decay Over Time

Even when mitigations are implemented, they often fail to persist across releases or vendor forks. Patches scoped to specific builds, configurations, or client variants tend to regress in subsequent updates. The result is an illusion of progress that masks continued exploitability.

4.5 Lesson 5: Internet-Freedom Users Need Persistent Visibility Tools

For users in censored or adversarial environments, transient disclosure cycles provide little protection. Long-lived, architectural vulnerabilities require ongoing visibility, tracking mechanisms that capture persistence and partial remediation rather than binary "fixed" or "unfixed" states.

5 Illustrative Evidence: Persistence of Architectural Vulnerabilities

To ground the preceding lessons empirically, we re-tested the VPN inference attacks first disclosed in 2019 and formalized in prior work [47]. We evaluated whether client-side attack primitives for inference, hijacking, or disruption remain exploitable on modern mobile devices and VPN implementations, using NordVPN where applicable due to its integration of multiple protocols in a widely deployed consumer application.

All tests were conducted after the assignment of CVE-2024-49734 on fully patched platforms. Annotated PCAP traces and minimal reproduction scripts, provided to support verification while limiting unnecessary exposure, are available in a public artifact repository [48].

5.1 Experimental Setup

The experiments were conducted using a Pixel 10 Pro XL running Android 16 with all October 2025 security updates applied. The device was connected to a WiFi access point controlled by the attacker, simulating an in/on-path adversary with the ability to observe and inject packets on the local network.

Our goal was to determine whether the core client-side attack primitives first described in prior work [47] remain exploitable on fully patched systems. Specifically, for each tested tool we evaluate whether a blind in/on-path adversary can (1) discover the VPN-assigned internal IP address of the device, (2) infer the existence of an active encrypted TCP connection to an external destination, and (3) disrupt that connection *via* TCP sequence/acknowledgment manipulation. A tool is considered vulnerable if any of these primitives succeeds.

The experiments replicated the attack methodology described in prior work [47], including:

- Scanning common private subnets (*e.g.*, 10.0.0.0/8) to infer VPN-assigned internal IPs
- Sending spoofed SYN/ACK probes from external IPs to candidate VPN IPs and ports
- Inferring active connections based on response behavior (*e.g.*, silent drops vs. encrypted RSTs)
- Performing sequence number window scans to trigger session resets

Across all tests, the attacker position, network topology, and probing strategy were held constant; only the VPN or circumvention tool under test was varied. No traffic decryption was attempted; all results were derived from metadata analysis and observable side-channel behavior at the tunnel boundary.

5.2 Tool-Specific Results

We evaluated six widely used VPN and circumvention tools under identical adversarial conditions, using default configurations and routing all traffic through an attacker-controlled gateway. Where applicable, tools were tested *via* their integration within the NordVPN Android client in addition to their upstream reference implementations, reflecting common real-world deployment.

Across all tested tools, we observed successful internal IP inference, connection-state detection, and TCP reset injection, indicating that the attack primitives remain viable in practice.

OpenVPN. The attack was fully successful against both the reference OpenVPN client and NordVPN's OpenVPN-based implementation. In NordVPN's Android client, we inferred the VPN-assigned internal IP, identified active ephemeral ports, and successfully triggered TCP connection resets using forged RST packets.

WireGuard. The attack remained viable against the reference WireGuard implementation as well as NordVPN's WireGuard-based integration, NordLynx. Spoofed packets within the TCP sequence window consistently triggered encrypted resets.

NordWhisper. Despite being marketed for use in restricted environments [35], NordWhisper failed to resist connection inference or disruption. We confirmed successful attacks across multiple transport layers, indicating that obfuscation did not extend to tunnel endpoint behavior or metadata exposure.

Orbot: Tor on Android. The attack was successful against Orbot, which uses VPN-style tunneling to encapsulate Tor traffic. We inferred active connections to the Tor network and triggered session resets via spoofed probes. These results raise concerns about the reliability of Orbot under adversarial network conditions.

Lantern. We observed consistent success in triggering resets and fingerprinting active sessions. Port prediction was straightforward, and no evidence of metadata obfuscation or injection resistance was present.

Psiphon. Despite being positioned as a censorship circumvention tool, Psiphon failed to block spoofed resets or conceal session metadata. Obfuscation at higher protocol layers did not prevent TCP-layer disruption.

Rationale and Limitations. Our central claim is not that any single implementation is uniquely flawed, but that the underlying attack primitives persist at the operating system and tunnel-behavior level. Differences in implementation may affect exposure in specific cases, but the presence of widely deployed configurations in which inference and disruption remain possible is sufficient to demonstrate that this vulnerability class persists in practice.

5.3 Persistence of Attack Primitives

Across all tools, the core attack primitives, internal IP discovery, ephemeral port inference, and TCP sequence window probing, remained effective. These behaviors were consistent regardless of VPN protocol or application-layer obfuscation, confirming that the root causes lie not in application bugs but in fundamental design assumptions about tunnel behavior.

Mitigations such as strict interface scoping, policy-based routing, or stateful response filtering were inconsistently applied, often disabled by default, or absent altogether. In multiple cases, previously discussed mitigations did not appear to be present in current builds, raising the possibility that they were either incomplete, ineffective, or not retained across versions.

5.4 Risks to At-Risk Users

These findings are especially concerning for users in high-risk environments. Tools like Orbot, Lantern, and Psiphon are regularly recommended in digital security trainings and circumvention guides, where they are assumed to protect users against surveillance and traffic correlation. However, our results show that adversaries do not need to decrypt traffic to extract sensitive information—they only require local network position and the ability to spoof.

Blind inference attacks allow an adversary to determine whether a device is accessing a censored service, and in some cases, forcibly terminate that session. In authoritarian contexts, such capabilities can be used to punish users for forbidden behavior, track political activity, or deny access to vital information. These attacks do not require malware, privilege escalation, or sophisticated tooling (only packet injection and a basic understanding of tunnel routing behavior).

5.5 Accountability Gaps in Internet Freedom Security Models

Beyond individual user risk, these results raise broader questions for the Internet Freedom community itself. Many of the tools we tested are routinely promoted by civil society groups, open-source coalitions, and international aid organizations as trusted defenses against censorship and surveillance. However, these endorsements are rarely accompanied by systematic evaluations of long-term exploitability or metadata exposure risk.

The tools in question may offer application-layer obfuscation or encryption, but they still rely on tunnel architectures that remain vulnerable to packet injection and inference attacks. As a result, the security posture they project often exceeds what is technically delivered.

This disconnect points to a critical gap: there is no sustained mechanism for tracking the security status of architectural vulnerabilities across circumvention tools, nor is there infrastructure for incorporating attack persistence into tool vetting or threat modeling. The Internet Freedom community's reliance on patch cycles and protocol audits, rather than adversary-aligned testing, leaves users vulnerable to attack classes that are never explicitly acknowledged.

These findings validate the central claim of this paper: architectural vulnerabilities must be treated as systemic and long-lived. For the Internet Freedom community in particular, there is an urgent need for a persistent, community-driven reporting framework. A framework that foregrounds human risk and prioritizes architectural transparency over version-level fixes.

These empirical findings reinforce the broader lessons drawn from our disclosure experience and motivate the framework described in Section 6.

5.6 Ethical Considerations

The attack class examined in this paper has been publicly documented since 2019, including CVE assignments and prior peer-reviewed publication; our experiments re-test this known vulnerability on current platforms without introducing new exploit techniques or undisclosed attack surfaces. All testing was conducted in environments under the authors' control, did not affect uninvolved users or systems, and vendors were notified of re-testing results where applicable; the pending CVE in Section 3 reflects continued exploitability of a previously disclosed issue rather than a new discovery.

6 A Framework for the Internet Freedom Community

Building on the seven years of disclosure activity and subsequent re-testing described above, this section proposes a framework to address the recurring structural failures observed in practice.

The failures outlined in previous sections, persistent vulnerabilities, inadequate triage mechanisms, misleading CVE coverage, and the continued risk posed to high-risk users, demonstrate the need for a better standard for describing and reporting systemic, architectural vulnerabilities. Our experimental findings in Section 5 confirm that the attack primitives first disclosed in 2019 remain exploitable in 2025, despite multiple CVEs and vendor patches. These results underscore the central challenge: there is no infrastructure for tracking persistent, cross-vendor design-level risk.

This section proposes such a framework, tailored to the needs of the Internet Freedom community.

The framework outlined here is not presented as a definitive solution, but as a practical starting point shaped by experience. After years of encountering the same coordination failures, incomplete

fixes, and vanishing records, it is clear that traditional disclosure alone cannot keep pace with how architectural vulnerabilities affect real users.

Importantly, this is not a call to create an entirely new ecosystem, but a model for augmenting existing disclosure infrastructure with elements that prioritize user risk, contextual threat modeling, and long-term architectural persistence. It is not another layer of bureaucracy for security professionals and tool developers, but a practical format to capture and track vulnerabilities that currently escape our reporting systems.

At its core, this proposal is a commitment to recognizing, tracking, and addressing long-lived architectural vulnerabilities through sustained collaboration with the users most affected by them, and to treating transparency and persistence as shared responsibilities rather than afterthoughts.

6.1 Motivation and Scope

Unlike traditional CVEs, which are tied to specific software bugs, this framework is designed to track the lifecycle of architectural and behavioral vulnerabilities, particularly those relevant to censorship circumvention, anonymity preservation, and adversarial network conditions. These issues often emerge not as implementation defects, but as consequences of how systems are composed or deployed across platforms, protocols, and layers.

An Internet Freedom-specific reporting format would serve as a persistent, cross-vendor ledger of structural risks, recording attack methodologies, affected configurations, known partial mitigations, and temporal verification status. Instead of tracking patch closure, it would track exploitability, user relevance, and the structural origins of the risk. Most importantly, it would treat threat surface characteristics through the lens of who is most at risk and not just what code was changed.

These observations informed the following design principles, derived directly from the obstacles encountered during disclosure.

6.2 Core Design Principles

We propose five core principles for such a framework:

1. Persistent Cross-Vendor Tracking. All vulnerabilities would be tracked across protocols, platforms, and products under a shared identifier (*e.g.*, Internet Freedom-ARCH-2025-001). Entries would stay visible and current as long as the risk remains reproducible, regardless of vendor-specific claims of resolution or shifting product versions.

2. Human-Centered Impact Ratings. Vulnerabilities would be scored not solely by exploitability or CVSS-like metrics, but by harm potential to vulnerable users. An attack that allows a state-level adversary to infer connections to a censored news site is more urgent to a journalist than a memory safety bug that requires root access. This framework centers severity around those realities. In practice, human-centered impact need not take the form of a fine-grained numerical score. Instead, entries could use a small number of coarse impact tiers (*e.g.*, Low, Moderate, High, Critical) derived from clearly defined questions: Does the attack enable identification of tool usage? Does it allow inference of access to specific services? Does it enable disruption, blocking, or punishment of user behavior? Does it expose users to physical, legal, or political harm? By anchoring severity to concrete consequences rather than abstract exploitability metrics, ratings can remain consistent across diverse vulnerability classes while better reflecting real-world risk to high-threat populations.

3. Verification and Retesting Hooks. Each entry would include a history of re-verification efforts, third-party confirmations, and updated proofs-of-concept. Timestamped observations would capture re-emergence or persistent viability. Entries could optionally link to PCAPs, test scripts, and minimal configs to facilitate independent reproduction. This supports scientific rigor while discouraging the closure bias that currently follows CVE assignments.

4. Structured Media and Advocacy Interface. Rather than allow narratives to be shaped by incomplete or inaccurate coverage, each entry would include a layperson-oriented impact summary, shareable quotes and caveats, and a vendor status matrix. This empowers security trainers, advocates, and journalists to relay both the existence and context of a vulnerability clearly.

5. Transparency and Public Access. The registry would be open and searchable, with simple public interfaces and optional APIs for integration into risk dashboards and secure configuration guides. A user running a tool like Orbot or Lantern should be able to quickly check whether its current configuration is vulnerable to known metadata-based attacks.

6.3 Use Case Alignment and Generalizability

Although motivated by the VPN-based attack revisited in this paper, this framework generalizes to a wide class of Internet Freedom-relevant vulnerabilities. These include:

- Timing analysis of encrypted messaging and transport (*e.g.*, Signal, Tor bridges, MPTCP)
- Protocol-level fingerprinting and traffic correlation of circumvention tools (*e.g.*, Lantern, Psiphon, Orbot)
- Behavioral side-channels in mobile OS traffic shaping, sensor co-activation, or captive portal logic
- Metadata leaks from DNS-over-HTTPS fallbacks or inconsistent tunneling
- Protocol downgrade attacks that reintroduce known weaknesses under edge case routing
- Persistent network side-channels such as IPID-based inference and sequence-window probing

These categories all exhibit the same traits: they operate across traditional product boundaries, defy patch-based remediation, and are rediscovered again and again due to the lack of institutional memory or shared tracking.

6.4 Relationship to Existing Infrastructure

This framework does not seek to replace the Common Vulnerabilities and Exposures (CVE) system, but it does challenge its suitability for tracking the types of vulnerabilities that matter most to the Internet Freedom community. CVE is fundamentally optimized for vendor-acknowledged implementation flaws: memory corruption, input validation, misconfiguration. It is not built to track cross-layer behavioral risks, like how VPN tunnels handle spoofed packets

across NAT boundaries, or how Tor behaves on Android in the presence of captive portal logic.

Even when architectural flaws are acknowledged, the CVE system rarely provides a usable path for continued visibility or user-oriented threat modeling. CVE entries are frequently abstract, vendor-scoped, and tied to specific software versions. For users and advocates trying to determine whether a given tool or configuration remains vulnerable, CVE offers little actionable context.

Recent events have only deepened these concerns. In April 2025, funding for the CVE program was nearly withdrawn by the U.S. government, prompting a short-term extension but exposing the system's structural fragility [36]. In response, the European Union launched the EU Vulnerability Database (EUVD) as a parallel effort to ensure more resilient and transparent vulnerability tracking [10].

These developments do not suggest CVE is obsolete, but they do make clear that it is incomplete. Our proposed framework builds alongside these systems, not on top of them. It aims to fill the gaps CVE cannot: tracking long-lived architectural flaws, enabling adversary-aligned verification, and giving users, not vendors, a way to assess their own risk.

6.5 Call for Community Participation

No framework like this can succeed in isolation. It must be owned, iterated on, and refined by those who understand the problem best: researchers, educators, advocacy organizations, community VPN projects, and security trainers working directly with high-risk users.

We invite collaboration from organizations with deep expertise in Internet Freedom tool evaluation and deployment. These groups already maintain infrastructure for testing censorship, monitoring throttling, and building privacy-preserving tools. This framework is a natural extension of that mission: a living ledger of structural risks that persist across vendors, protocols, and platforms.

The goal is not to impose a standard, but to create a vocabulary for harms that are otherwise ignored—a shared language for architectural failure that resists dismissal as a mere quirk of routing. As discussed in Section 3.5, the absence of trusted, persistent framing has allowed such vulnerabilities to be distorted, minimized, or quietly dismissed, undermining both vendor response and academic review. This registry offers an alternative: a stable, community-grounded frame for architectural risk that cannot be overwritten by vendor statements or media spin.

Ultimately, this proposal is the product of lived experience with a vulnerability that refused to disappear; it translates that persistence into a model for sustained visibility and accountability. We recognize that this framework will evolve as the community engages with it; its value lies less in completeness than in resisting the institutional amnesia that allows architectural flaws to persist.

7 Recommendations and Call to Action

The failures described throughout this paper are not isolated events but recurring patterns that will continue unless the community commits to collective accountability. The following recommendations translate these observations into actionable steps for the groups most directly positioned to intervene. Each builds on the same principle: persistent vulnerabilities require persistent attention.

For Researchers

Systemic, architectural vulnerabilities demand long-term thinking and tactical persistence. We urge researchers in Internet Freedom contexts to resist the closure bias embedded in CVE-driven workflows. Track vulnerabilities across versions, beyond vendor patches and publication timelines, and document patterns that re-emerge even after supposed resolution. Adopting a shared reporting framework, such as the standard proposed in Section 6, will provide visibility and enable the collective tracking of architectural risk across projects and institutions. This work lays the groundwork for shared visibility, even when vendor engagement is absent or inconsistent.

For the Internet Freedom Community

Security trainers, tool developers, and advocacy organizations must treat systemic behavior as a threat surface. The tools we recommend—VPNs, proxies, and tunnels—may expose users to metadata inference and protocol-level surveillance even when patches are up to date. Threat models must reflect this. We call on the Internet Freedom community to adopt user-centered reporting standards that prioritize harm potential, elevate architectural flaws, and treat persistent design weaknesses as active threats rather than background noise. Doing so will enable more accurate recommendations and strengthen user trust in the tools we endorse.

For Vendors

If your product is used in adversarial environments, your security model must reflect that reality. Threats like spoofed traffic, side channels, and tunnel-based fingerprinting are not edge cases, they are core risks for journalists, dissidents, and rights defenders. We urge vendors to be transparent about architectural limitations, to support long-term tracking of design-level risks, and to collaborate openly with civil society actors working on real-world harm reduction.

8 Conclusion

After seven years of disclosure, testing, and re-engagement, the continued viability of VPN-based inference attacks reveals a structural failure rather than a sequence of isolated lapses. Architectural risks cross layers, evade clear ownership, and remain exploitable across updates, leaving critical threats—such as server-side injection and tunnel-based fingerprinting—untracked and poorly understood.

This failure has direct consequences for users who rely on VPNs and circumvention tools in adversarial environments, including journalists, dissidents, and civil-society actors. For these communities, architectural vulnerabilities are not abstract deficiencies in process but ongoing threats to safety and anonymity.

To address this gap, this paper proposes a complementary framework for tracking persistent, behavioral, and architectural vulnerabilities that resist the patch-and-close model. Rather than replacing CVE, the framework augments it by emphasizing longitudinal tracking, cross-vendor analysis, and user-centered risk reporting. The Internet Freedom community is uniquely positioned to lead this effort by building the shared institutional memory that current systems lack, ensuring that long-lived threats remain visible, accountable, and actionable.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. CNS-2452885. We thank the anonymous reviewers and our shepherd for their constructive feedback, which improved the clarity and quality of this work. We also thank Beau Kujath, Mohammad Taha Khan, and Narseo Vallina-Rodriguez for their contributions to the original work on blind in/on-path VPN attacks, and the Open Technology Fund for support during earlier phases of this research, including William's time as an Information Controls Fellow.

References

[1] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, and et al. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 5–17. ACM, 2015.

[2] Geoffrey Alexander, Antonio M. Espinoza, and Jedidiah R. Crandall. Detecting TCP/IP connections via IPID hash collisions. In *Proceedings on Privacy Enhancing Technologies*, volume 2019 (4), pages 311–328. Sciendo, 2019.

[3] Android Open Source Project. Handle v4-mapped v6 address in struct parsing, 2025. Commit e72c61380c52a4450970556e5936c5ec03fd66fb.

[4] Android Security Team. Pixel update bulletin — September, 2019.

[5] Android Security Team. Android security bulletin — January, 2025.

[6] Apple Inc. About the security content of iOS 13.6 and iPadOS 13.6, 2020. https://support.apple.com/en-us/103112.

[7] Apple Inc. About the security content of iOS 17.2.1 and iPadOS 17.2.1, 2025. https://support.apple.com/en-us/120877.

[8] BleepingComputer. New linux vulnerability lets attackers hijack vpn connections. https://www.bleepingcomputer.com/news/security/new-linux-vulnerability-lets-attackers-hijack-vpn-connections/, 2019.

[9] Check Point Software Technologies. sk182336 - preventative hotfix for CVE-2024-24919. https://support.checkpoint.com/results/sk/sk182336, 2024.

[10] Thomas Claburn. Splintering bug tracking: Europe launches its own CVE-style vulnerability database. https://www.theregister.com/2025/04/18/splintering_cve_bug_tracking/, 2025.

[11] Codenomicon and Google. CVE-2014-0160: Heartbleed vulnerability in openssl. https://www.heartbleed.com/, 2014.

[12] Jedidiah R. Crandall, Beau Kujath, and William J. Tolley. Blind in/on-path attack disclosure faq. https://breakpointingbad.com/2020/08/12/VPN-FAQ.html, 2020.

[13] Jedidiah R. Crandall, Beau Kujath, and William J. Tolley. Vintage protocol nonsense: Annoying the tcp stack to uncover tunneled vpn connections. https://breakpointingbad.com/2020/05/25/Vintage-Protocol-Nonsense.html, 2020.

[14] Joshua J. Daymude, Antonio M. Espinoza, Holly Bergen, Benjamin Mixon-Baca, Jeffrey Knockel, and Jedidiah R. Crandall. A taxonomy and comparative analysis of ipv4 identifier selection correctness, security, and performance. *ACM Comput. Surv.*, 58(6), December 2025.

[15] DD-WRT Forum. Cve-2019-14899: Suggested vpn vulnerability mitigation. https://forum.dd-wrt.com/phpBB2/viewtopic.php?t=322428&start=60, 2019.

[16] Theo de Raadt. Re: Vpn leak vulnerability (cve-2019-14899). https://marc.info/?l=openbsd-tech&m=157580561114203, 2019.

[17] Theo de Raadt. Re: VPN leak vulnerability (CVE-2019-14899). https://marc.info/?l=openbsd-tech&m=157580561114203, 2019. OpenBSD tech@ mailing list post, December 8, 2019.

[18] Jason A. Donenfeld. Regarding "inferring and hijacking vpn-tunneled tcp connections". https://lists.zx2c4.com/pipermail/wireguard/2019-December/004679.html, 2019.

[19] Roya Ensafi, Jeffrey Knockel, Geoffrey Alexander, and Jedidiah R. Crandall. Detecting intentional packet drops on the internet via tcp/ip side channels. In *Passive and Active Measurement (PAM)*, pages 109–118. Springer, 2014.

[20] Fortinet PSIRT. Psirt note: Cve-2019-14899 – inferring and hijacking vpn-tunneled tcp connections. https://community.fortinet.com/t5/FortiClient/PSIRT-Note-CVE-2019-14899-Inferring-and-hijacking-VPN-tunneled/ta-p/192058, 2019.

[21] Fortinet PSIRT. Heap buffer overflow in SSLVPN pre-authentication (CVE-2023-27997). https://fortiguard.fortinet.com/psirt/FG-IR-23-097, 2023.

[22] Steve Gibson and Leo Laporte. VPN-geddon denied (security now! #744). https://www.grc.com/sn/sn-744.htm. Podcast episode recorded December 10, 2019.

[23] HackerNoon. Military grade encryption won't save you, or your business. https://hackernoon.com/military-grade-encryption-wont-save-you-9a3d32zs, 2019.

[24] Ivanti. April security advisory: Ivanti connect secure, policy secure & zta gateways (CVE-2025-22457). https://forums.ivanti.com/s/article/April-Security-Advisory-Ivanti-Connect-Secure-Policy-Secure-ZTA-Gateways-CVE-2025-22457, 2025.

[25] Jeffrey Knockel and Jedidiah R. Crandall. Counting packets sent between arbitrary internet hosts. In *4th USENIX Workshop on Free and Open Communications on the Internet (FOCI 14)*, San Diego, CA, August 2014. USENIX Association.

[26] LWN.net. Vpn hijacking on linux (and beyond) systems. https://lwn.net/Articles/806546/, 2019.

[27] MITRE. CVE-2019-14899: Inferring and hijacking VPN-tunneled tcp connections. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-14899, 2019.

[28] MITRE. CVE-2019-9461: Android VPN routing information disclosure. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-9461, 2019.

[29] MITRE. CVE-2024-49734: Connectivityservice side-channel information disclosure. https://nvd.nist.gov/vuln/detail/CVE-2024-49734, 2024.

[30] MITRE CVE Team. Email correspondence regarding cve eligibility for in/on-path vpn attacks. Archived correspondence. https://git.breakpointingbad.com/Breakpointing-Bad-Public/FOCI-2026-Artifact/src/branch/main/correspondence, 2019. Available in the FOCI 2026 artifact repository.

[31] Ben Mixon-Baca, Ron Deibert, and Miles Kenyon. Vulnerabilities in VPNs: The port shadows attack and the case for greater transparency. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2024(3):373–391, 2024.

[32] Lizzie Moratti and Dani Cronce. TunnelVision: How attackers can decloak routing-based VPNs for a total VPN leak. https://www.leviathansecurity.com/blog/tunnelvision, 2024.

[33] Mullvad VPN. A closer look at VPN vulnerability CVE-2019-14899. https://mullvad.net/en/blog/closer-look-vpn-vulnerability-cve-2019-14899, 2019.

[34] Marcin Nawrocki, Mattijs Jonker, Thomas C. Schmidt, and Matthias Wählisch. The far side of DNS amplification: Tracing the DDoS attack ecosystem from the internet core. In *Proceedings of the 2021 ACM SIGCOMM Conference*, pages 704–717, New York, NY, USA, 2021. Association for Computing Machinery.

[35] NordVPN. Introducing the nordwhisper protocol — a step towards a more open internet. https://nordvpn.com/blog/nordwhisper-protocol/, 2025.

[36] Kate O'Flaherty. CVE program faces uncertain future after funding cut—what it means and what to do next. *Forbes*, 2025.

[37] OpenVPN Inc. Response to CVE-2019-14899. https://archive.today/0ub9s, 2019. Archived version of now-deleted official statement originally available at https://openvpn.net/security-advisory/no-flaws-found-in-openvpn-software/.

[38] Oracle Corporation. Oracle critical patch update advisory – october 2020. https://www.oracle.com/security-alerts/cpuoct2020verbose.html, 2020.

[39] Private Internet Access. Private internet access updates linux desktop client to prevent against cve-2019-14899. https://www.privateinternetaccess.com/blog/private-internet-access-updates-linux-desktop-client-to-prevent-against-cve-2019-14899/, 2019.

[40] ProtonVPN. Statement on CVE-2019-14899. https://protonvpn.com/blog/statement-on-cve-2019-14899/, 2019.

[41] Haya Shulman. Pretty bad privacy: Pitfalls of DNS encryption. In *13th Workshop on Privacy in the Electronic Society (WPES)*, pages 191–200. ACM, 2014.

[42] Slashdot. New linux vulnerability lets attackers hijack vpn connections. https://linux.slashdot.org/story/19/12/05/2022205/new-linux-vulnerability-lets-attackers-hijack-vpn-connections, 2019.

[43] TechRadar. Vpn connections could be hacked due to linux security flaw. https://www.techradar.com/news/vpn-connections-could-be-hacked-due-to-linux-security-flaw, 2019.

[44] The Hacker News. New linux bug lets attackers hijack encrypted vpn connections. https://thehackernews.com/2019/12/linux-vpn-hacking.html, 2019.

[45] The Register. Tricky vpn-busting bug lurks in ios, android, linux distros, macos, freebsd, openbsd, say university eggheads. https://www.theregister.com/2019/12/06/vpnbusting_bug_spotted/, 2019.

[46] William J. Tolley, Beau Kujath, and Jedidiah R. Crandall. [CVE-2019-14899] inferring and hijacking VPN-tunneled tcp connections. https://seclists.org/oss-sec/2019/q4/122, 2019. oss-security mailing list, December 4, 2019.

[47] William J. Tolley, Beau Kujath, and Jedidiah R. Crandall. Blind in/on-path attacks and applications to VPNs. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2021.

[48] William J. Tolley, Everett Morse, Gabriel Hogan, Jeffrey Knockel, and Jedidiah R. Crandall. Foci 2026 artifact: Vpn architectural vulnerability evaluation. https://git.breakpointingbad.com/Breakpointing-Bad-Public/FOCI-2026-Artifact, 2026. Artifact repository.

[49] Mathy Vanhoef and Lennert Wouters. Tunnelcrack: Exploiting tunnel misconfigurations to leak VPN client traffic. In *Proceedings of the 32nd USENIX Security Symposium*, 2023.

[50] Diwen Xue, Benjamin Mixon-Baca, S. S. Valdik, Anna Ablove, Beau Kujath, Jedidiah R. Crandall, and Roya Ensafi. TSPU: Russia's decentralized censorship system. In *Proceedings of the 22nd ACM Internet Measurement Conference (IMC '22)*. Association for Computing Machinery, 2022.

[51] Diwen Xue, Reethika Ramesh, Arham Jain, Michalis Kallitsis, J. Alex Halderman, Jedidiah R. Crandall, and Roya Ensafi. OpenVPN is open to VPN fingerprinting. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Assoc, 2022.

A Timeline of VPN Vulnerability Disclosure

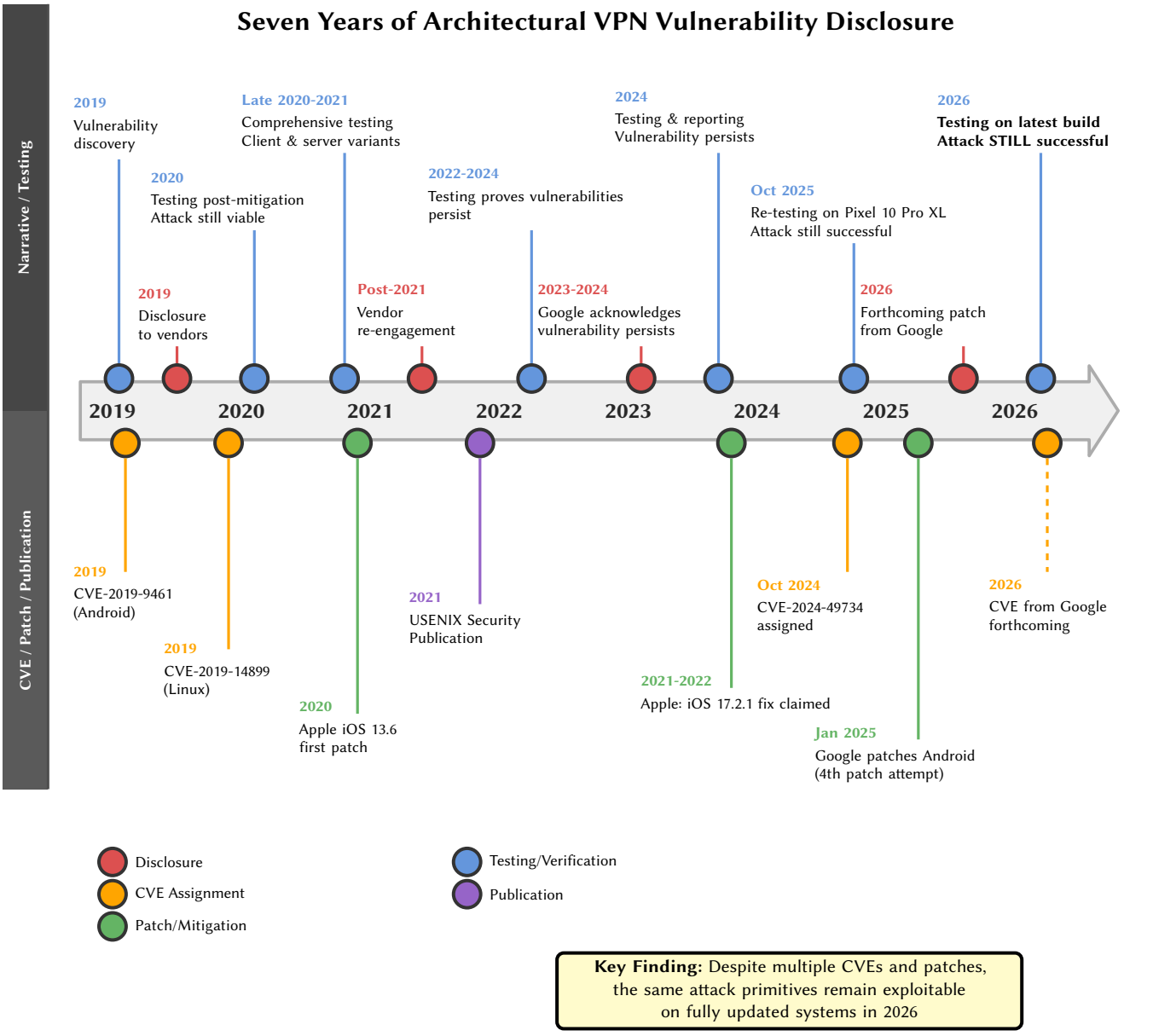

Figure 1: Seven-year timeline of the architectural VPN vulnerability, showing the divergence between formal disclosure/mitigation events and persistent exploitability across operating system versions.