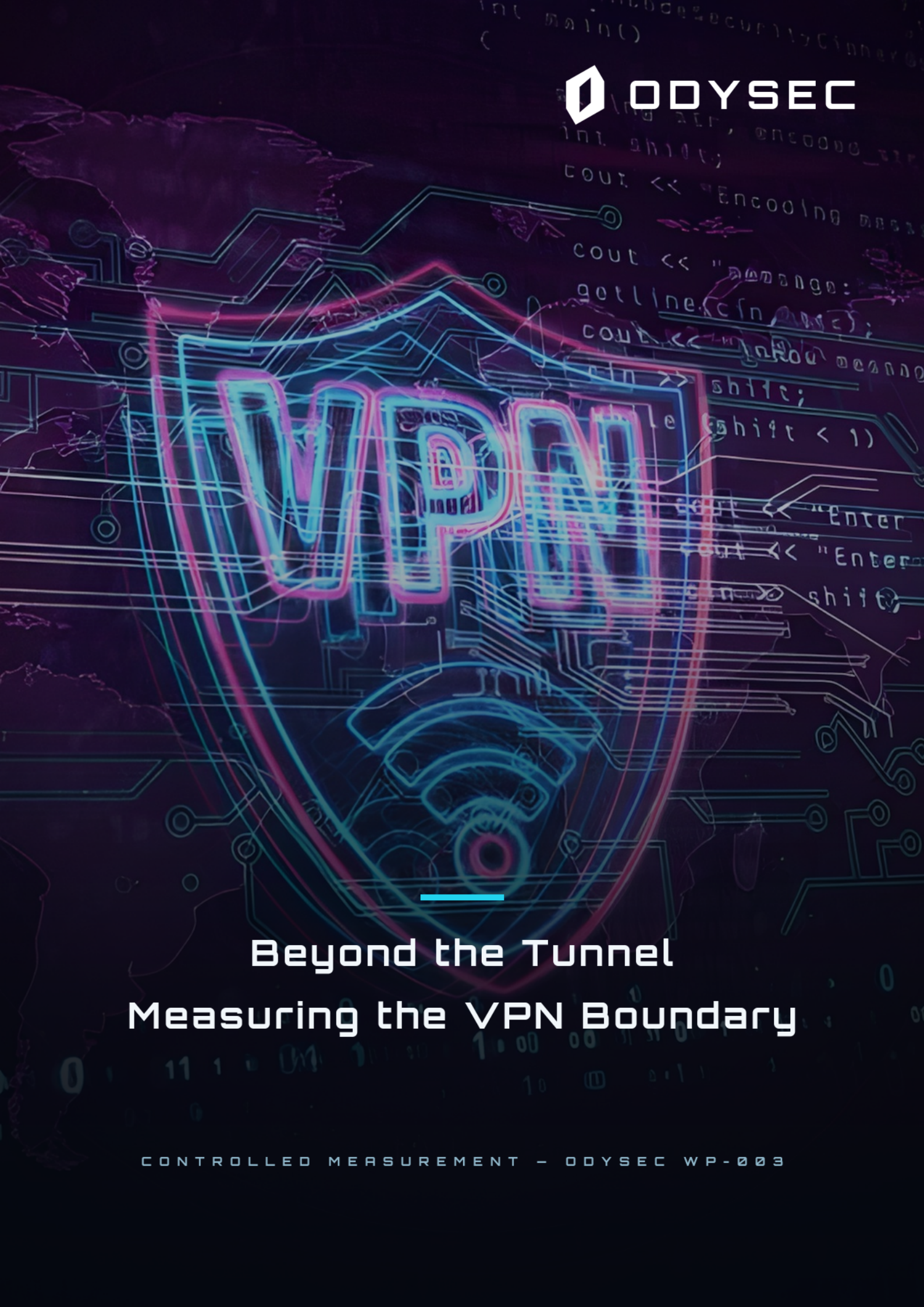

Beyond the Tunnel
Measuring the VPN Boundary

CONTROLLED MEASUREMENT – ODYSEC WP-003

Beyond the Tunnel

Measuring the Protection Boundary of a VPN

Odysec Research Report WP-002 | **Version** 1.0 | **Published** 2026-08-31 **Domain**: Cyber × Physical

The Traditional Chinese edition is the version of record. This is a translation.

Table of Contents

About this report

Every number in this report comes from a fully public controlled environment. Its topology file and all scripts are published alongside this report, and anyone can rebuild it and re-run the measurements.

The subject under test is a manually configured WireGuard tunnel (`wg-quick`), not any commercial VPN application. Commercial products implement their own kill switches, DNS handling and reconnection logic; this report knows nothing about them. Every number here applies only to the environment described in Section 3. That boundary is restated in Section 8; please do not cross it when citing this work.

What we measure are **failure modes of tunnel and routing logic**. This is methodological validation, not a product review.

Summary

That VPNs leak is not news. An `AllowedIPs` without `::/0` does not carry IPv6; a configuration that does not specify a resolver keeps using the local one. Both are documented behaviour. What is missing is not *whether* but **the shape of the boundary**: at which moments, for which traffic, how much, for how long.

Using a four-node controlled environment, we ran eight scenarios across seventeen configurations, 30 times each — 510 runs in total — and reached four findings:

1. **What determines whether traffic leaks is not whether the connection drops, but whether the tunnel interface survives.** With the physical link taken down for two seconds and restored, all 30 runs leaked **nothing** — the tunnel interface and its route persist through the outage, so packets are dropped inside the tunnel rather than diverted to the physical interface. When the interface itself disappears (a VPN process dying, being reclaimed by the operating system, or not rebuilt after resume), **every single packet goes out with the real address, and it never stops on its own.** The danger is not a flaky signal; it is a VPN that has quietly stopped running — and at that moment the network works perfectly, with nothing on screen to indicate otherwise.
2. **Failures fall into four classes with entirely different remedies.** The state class (establishment gap, link flap, interface loss) is solved by a kill switch **at no latency cost** — establishment time is statistically indistinguishable with and without one; it merely converts exposure into dropped packets. For the missing-configuration class (IPv6, DNS) a kill switch is no help at all. The third class is the hardest to notice: with a captive-portal exception allowing the local subnet, **the kill switch works throughout and traffic still gets out.** The fourth is not a "leak" at all; see the next point.
3. **Whether traffic leaks is determined by configuration, not by chance.** For IPv6, DNS and the portal exception, the dispersion of the leak fraction was **zero** — all 30 runs produced identical

results. IPv6 leaks 100% of the time under a v4-only tunnel; DNS leaks 100% of the time when the resolver is not taken over. Not once was there an exception. (Timing measurements do show dispersion; see the interquartile ranges in Section 4.)

4. **Get everything right and the local network still knows who you are.** With a full tunnel, resolver takeover and a strict kill switch all in effect throughout, the untrusted local network **still obtained, every single time**, this device's MAC address, NIC vendor, hostname and local address (identical across all 30 runs). That traffic did not "escape the tunnel"; it **never entered it** — ARP and DHCP do not even travel the ordinary IP output path, so a kill switch cannot reach them in principle. A tunnel protects what it has been configured to carry, not your presence on the network.

What to do: if you rely on a VPN to conceal your location, verify three things — that `AllowedIPs` covers IPv6 (or that IPv6 is disabled), that the resolver really has been taken over by the tunnel, and that any local-subnet exception was removed after you logged in to the public network. The three are independent; getting two right does not compensate for the third. **And if you also need this network to be unable to recognise this device, what that takes is a device-layer remedy, not any VPN setting.**

---

1. The problem

1.1 Who this matters to

For most people a VPN leak is a privacy problem. For the readers of our previous two reports — people trying to stay ahead of someone following them — it is a question of physical safety.

The methodology in WP-001 *The Eyes Beside You* begins with "Principle 0: switch to a clean device". But what happens once that clean device connects to a network? If its real address or its DNS queries escape the tunnel into a network the other person can reach, every precaution before that moment is undone. **A single leak is enough to reveal where you are.**

This is the canonical path by which digital risk becomes physical risk, and it is why this report follows the previous two.

1.2 What is missing is the shape of the boundary

VPN leakage has been discussed in technical circles for years, but public writing largely stops at "this can leak, be careful"; Chinese-language material is dominated by product reviews, with almost no systematic failure testing.

That level of detail helps a reader very little. "Can leak" answers none of the following:

- Between pressing connect and protection taking effect, how long is the gap, and what goes out during it?
- Walking into a lift and losing signal, then regaining it — does that leak?

- If the VPN process is reclaimed by the operating system, will I know? Does traffic stop, or does it continue in the clear?
- What does a kill switch cost? What does it stop, and what does it not?

All of these are measurable. Measuring them is the work of this report.

1.3 Questions this report answers

1. Under each failure scenario, do the client's real address, DNS queries and traffic escape the tunnel?
2. If so, how much, and for how long?
3. In which scenarios is a kill switch effective, in which is it not, and at what cost?
4. Can the user detect any of this?

---

2. Threat model

2.1 The attacker

We assume an attacker who **controls the local network the victim is on**: the home router, hotel or airport Wi-Fi, or a hotspot the other person provides. This is an extremely common position in stalking situations — a cohabitant, a former partner, or whoever administers the network equipment at the residence holds it naturally.

The attacker needs neither to compromise a device nor to break encryption. They only need to **see**: which packets go out with the real address, and which domains are looked up.

2.2 Capability assumptions

The attacker can:

- Observe all traffic on the local segment and its source addresses
- Supply DHCP configuration, including a local DNS resolver
- Observe and record every query passing through that resolver
- Run services on the local segment

The attacker cannot:

- Decrypt the contents of the WireGuard tunnel
- Execute code on the device under test
- Take control of the tunnel endpoint

In other words this report is entirely about **bypass**: the encryption is not broken, some things simply never enter it.

2.3 The victim's situation

Continuing the assumptions of WP-001: the victim already suspects they are being monitored and is taking precautions, and has **limited technical knowledge** — they followed general advice, installed a VPN, saw the interface say "connected", and concluded their location was protected.

The gap between that belief and the actual state is what this report examines.

---

3. Method

3.1 Environment and topology

The experiment runs in an EVE-NG network emulation environment. All four nodes are Debian 13, driven entirely through serial consoles by script, with no manual intervention.

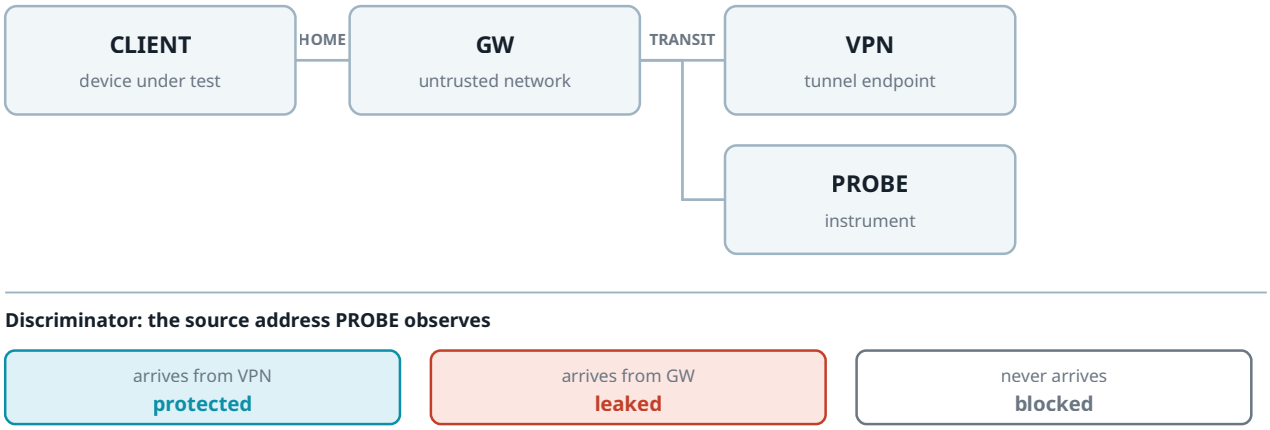

Figure 1: Experimental topology and the discriminator

Node	Role
CLIENT	Device under test. Takes its configuration from the untrusted network by DHCP, then establishes a WireGuard tunnel
GW	Home router / untrusted local network. DHCP, local DNS resolver, outbound NAT
VPN	Tunnel endpoint. Tunnelled traffic leaves here bearing its own address
PROBE	The instrument. Authoritative DNS plus HTTP/UDP receiver, recording the source address of every request

All addresses come from the documentation ranges of RFC 5737 and RFC 3849, so readers rebuilding the environment will not collide with real networks.

3.2 The discriminator

The whole measurement rests on one thing: **the source address PROBE observes.**

Observed source	Verdict
The VPN's address	Sent through the tunnel; protected
The GW's address	Leaked — sent directly, bypassing the tunnel
Never arrives	Blocked (kill switch engaged) or lost

This only works if **there is no bypass path between the two segments**. We verified before measuring that with NAT not yet configured on GW, CLIENT cannot reach the TRANSIT segment by any means. Without that check, a source address would not map uniquely to a path and the whole measurement would be meaningless.

3.3 Beacon design

Timing windows are measured with UDP beacons at 20 ms intervals. UDP rather than HTTP is deliberate: UDP is connectionless, has no retransmission and no handshake, so whether a datagram goes out faithfully reflects the routing and filtering state at that instant. **TCP retransmission would smear the timeline and the window boundaries would be unmeasurable.**

The beacons and the perturbation (for example "issue the connect command") run in **the same process on the same clock**. Split across two processes, we would have to account for process start-up jitter — measured in tens of milliseconds, the same order of magnitude as the quantity being measured. All arithmetic is done on the client's own monotonic clock; PROBE only attributes sources, so no cross-node clock synchronisation is needed at all.

Each beacon sequence number has exactly three possible outcomes: arrives from the VPN address, arrives from the GW address, or never arrives. These three cover every case of interest.

3.4 Scenarios and controls

Eight scenarios, 30 repetitions per cell — 510 runs. s1 through s6 have two variants each; s7 adds a supplementary variant to make attribution possible (see 4.4).

Scenario	Perturbation	Baseline	Control
s1 establishment gap	issue connect command	no kill switch	kill switch
s2 link flap	link down 2 s, then up	no kill switch	kill switch
s3 interface loss	delete the tunnel interface	no kill switch	kill switch
s4 IPv6	steady state	`AllowedIPs` v4 only	includes `::/0`
s5 DNS	steady state	resolver not taken over	resolver taken over
s6 portal exception	tunnel not yet up	local-subnet exception	strict kill switch
s7 out of scope	steady state	no VPN in use	full tunnel + takeover + strict kill switch (supplementary: same configuration, no kill switch)
s8 stub resolver	steady state	resolver not specified	resolver specified

Each variant changes exactly one thing. A variant that changes two would leave any difference unattributable. For s4 the server side is dual-stack in both arms; the only variable is the client's `AllowedIPs`.

The choice of perturbation for s3 deserves comment. Our first design removed the tunnel peer to simulate failure, but that measures nothing: the interface and its route remain, so packets are simply discarded inside the tunnel and **do not leak**. What is actually dangerous is the interface disappearing together with its route, after which traffic falls back to the original default route. **Choosing the wrong perturbation produces a false negative.**

3.5 Validating the instrument

During this work we hit three **silent failures** — the instrument was broken, yet behaved exactly like a genuine negative result. Left unnoticed, the resulting "no leak" would have been written up as a finding.

1. **The client could not obtain IPv6.** The node image enabled IPv6 forwarding by default for its router role, and Linux refuses router advertisements when forwarding is on. The client therefore had no IPv6 address at all — an IPv6 leak test would have reported "no leak", when in fact there was no v6 to leak.
2. **The router could not send router advertisements.** `ip addr flush` removes the `fe80::` link-local address along with everything else and does not restore it, yet advertisements must be sourced

from link-local. **The failure was entirely silent**: the DNS service logged every advertisement as sent, so the log looked healthy, while a packet capture showed only the client's solicitations and no reply whatsoever.

3. **A setting was applied but had no effect.** The resolver setting in the tunnel configuration takes effect through the system's resolver-management mechanism, which this experiment had disabled in order to make "which resolver did the client ask" directly observable. The setting appeared to apply, the resolver did not change, and the control arm was identical to the baseline.

From which a general rule, recommended to anyone attempting similar work:

> **Before each scenario, verify that the instrument itself is in a measurable state, rather than going straight to the result.**

All three checks are now negative assertions inside the provisioning scripts — if link-local does not come back, provisioning aborts rather than letting the experiment run on a broken instrument.

---

4. Results

Figures are medians of 30 runs, with the interquartile range in parentheses. Medians rather than means, because these latency distributions are right-skewed and a mean would be dragged by a few outliers.

4.1 State failures

Scenario	No kill switch	With kill switch
s1 connect command → protected	53.3 ms (52.8–71.9), **2 packets** still leak afterwards (worst 4)	53.4 ms (53.2–53.5), **0** leaked
s2 link down 2 s, then restored	protected again 47.8 ms after recovery, **0** leaked	43.2 ms, **0** leaked
s3 tunnel interface disappears	**every packet leaks**, and it never stops	last packet arrives 8.6 ms *before* the event; silence thereafter, **0** leaked

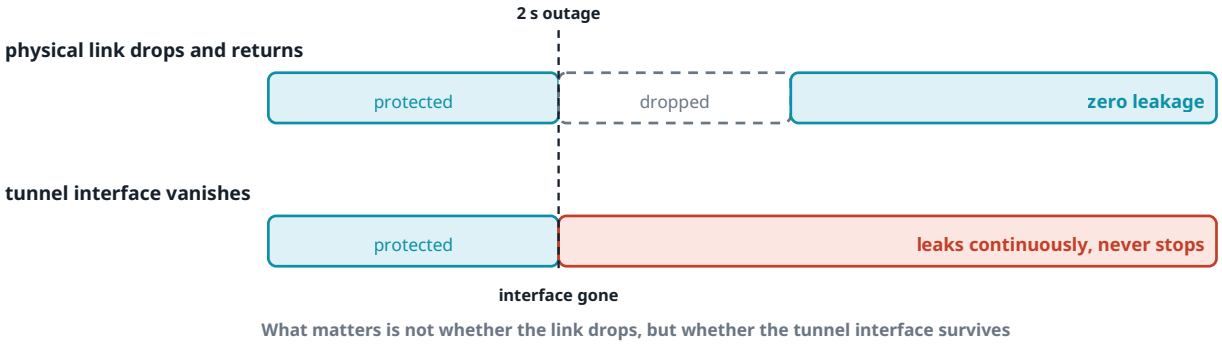

Figure 2: Two failures on the same timeline

Each merits comment.

s1: protection is not immediate after the user presses connect. During the roughly 53 ms of establishment in this environment, 2 to 4 packets still go out with the real address. Subjectively the user is "connected"; in fact they are not yet.

s2 is the most counter-intuitive result in this report. Taking the physical link down and bringing it back leaked **nothing across all 30 runs**. The tunnel interface and its route survive the outage: packets enter the tunnel and are discarded because the peer is unreachable (about 102 sequence gaps are visible), **they are not diverted to the physical interface**. Switching networks, walking into a lift and losing signal all belong to this class.

s3 is the opposite. When the interface itself disappears, routing falls back to the physical interface and from then on every packet goes out with the real address — and **it never stops on its own**; it was still leaking when the measurement window closed. The corresponding real-world events are a VPN process dying, being reclaimed by the operating system, or not being rebuilt after resume.

4.2 Missing configuration

Scenario	Baseline (common default)	Control (correct setting)
s4 IPv6 traffic	v4-only tunnel → **100% leaked** (all 30 runs)	includes `::/0` → 0.7%
s5 DNS queries	resolver not taken over → **100% leaked** (all 30 runs)	taken over → **0%**

Dispersion in both baselines is zero. **This is not a matter of probability; it is what the configuration determines.**

The residual 0.7% in the s4 control is the two packets sent before the tunnel came up — the same tail measured in s1, not something specific to IPv6.

s8: the same question with a stub resolver in place. To make "which resolver did the client ask" directly observable, s5 deliberately disabled the system's stub resolver; on a real desktop it is enabled by default. s8 re-measures with it enabled, 30 runs per cell and 10 distinct names per run (300 queries per cell):

Measure	Resolver not specified	Resolver specified
Names sent only through the tunnel	0	**10 / 10**
Names sent only to the local resolver	**10 / 10**	0
Names sent to both	0	0

No partial leakage, and no duplicate sending. In this combination the tunnel's resolver takeover is fully effective: the mechanism is a routing domain covering all domains, set on the tunnel interface, after which the stub resolver no longer distributes queries to other interfaces. This is worth stating, because the stub resolver's split-DNS mechanism is precisely what could otherwise send some queries to the local resolver — we did not observe that here.

4.3 The exception itself

The captive-portal exception exists so that users can log in to a hotel or airport portal; at that moment the tunnel is not yet up, so nearly every kill switch implementation allows the local subnet.

Variant	Packets reaching the untrusted network	Fully blocked
Strict kill switch	**0**	30/30 runs
With portal exception	**226** (identical across 30 runs)	0/30 runs

The observation point sits on GW — the local subnet is precisely what the attacker controls, so any packet arriving there is already in their hands. In the real world the lure is DNS: the local resolver remains reachable under the exception, and the attacker need only answer with a local-subnet address.

The kill switch works throughout and traffic still gets out. This is the least detectable of the eight: the user sees that the kill switch is enabled, and it genuinely is.

4.4 Out of scope

The first three classes of failure share a shape: traffic that should have gone into the tunnel did not. The fourth is different. **There is a whole class of traffic that never entered the tunnel at all**, because the tunnel was never configured to carry it — and some of it does not even travel the ordinary IP output path.

The observation point sits on GW, that is, on the untrusted local network's side of the link. Three configurations, 30 runs each:

Observed	No VPN in use	Everything configured correctly	Supplementary: same, no kill switch
Frames sent by the client (median)	23	6	22
Items of identifying information	10	**4**	9

The count of identifying items has **zero dispersion** in all three cells — all 30 runs produced identical results.

The four items that still get out under the "everything configured correctly" arm, and why a kill switch does not stop them:

Information still obtainable	Why it is not stopped
MAC address	Every Ethernet frame carries the real MAC. Under a strict kill switch the only frames still leaving are the tunnel's own — and those carry it too
NIC vendor (OUI)	Derived directly from the first three bytes of the MAC
Hostname	DHCP carries the hostname in the clear, and that request goes out over a raw socket, bypassing the IP output chain
Local address	ARP belongs to a different filtering family and likewise never reaches the IP output chain

The supplementary variant lets each cause be attributed individually:

- **The tunnel itself removed only one item** (NTP). It is ordinary routed IP traffic, carried along by the full-tunnel setting. The other nine the tunnel never touched at all.
- **Five more were removed by the kill switch, not the tunnel**: mDNS, LLMNR and the IPv6 router solicitations, neighbour solicitations and multicast listener reports (MLD) all travel the IP output path and so hit the blocking rules.
- **Four are stopped by neither**, for the reasons in the table above.

What this class means: with the kill switch in effect throughout, the tunnel present throughout, and the resolver genuinely taken over, the untrusted local network still knows which device this is. It is not a "leak" but the boundary of what the tunnel carries — and for exactly that reason, none of the remedies for the first three classes has any effect on it.

> **What this section cannot support**: we tested one kill switch implementation only (a rule set acting on the IP output chain). **It does not follow that no kill switch can stop ARP and DHCP** — a rule set acting at the network device layer's egress hook was not tested. The list of identifying information is also limited by the software installed on this client: our DHCP client does not send

> the vendor class or client identifier by default, so what we measured here is "not sent", not "no such channel"; other operating systems do send them.

4.5 Overall

Traffic / event	Result under default configuration
IPv4, steady state	Protected
IPv4, physical link flap	Protected
IPv4, tunnel interface gone	**100% leaked, and it does not stop**
IPv4, first tens of ms after connect	2–4 packets leaked
All IPv6	**100% leaked**
All DNS queries	**100% leaked**
Under the portal exception	**100% escapes** (even with the kill switch working)
Identity on the local segment	**four items get out even when everything is right** (MAC, OUI, hostname, local address)

---

5. Discussion

5.1 Four classes, four remedies

Class	Scenarios	Remedy	Kill switch?
State	establishment gap, flap, interface loss	kill switch	✔ effective, at no cost
Missing configuration	IPv6, DNS	correct settings	✘ no effect
The exception itself	portal allows local subnet	remove after login	✘ the exception defeats it
Out of scope	ARP, the DHCP hostname, the MAC at the Ethernet layer	device-layer remedies instead: randomise the MAC, send no hostname	✘ cannot reach them

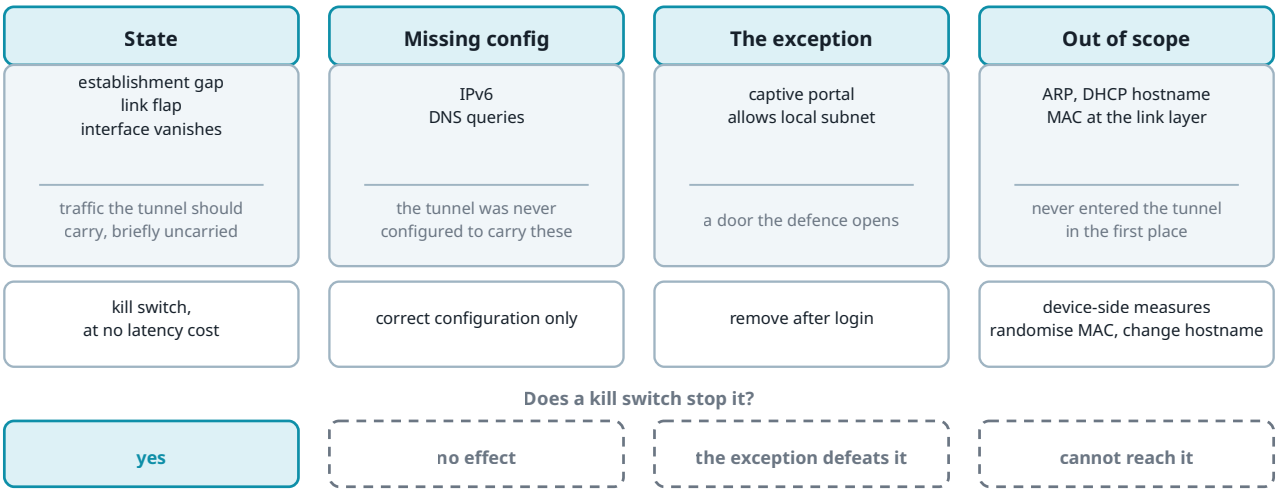

Figure 3: Four classes of failure and their remedies. A kill switch covers only the first.

This distinction is the main practical output of the report. It turns the vague question "is a VPN safe" into four concrete questions with four different answers. **Getting three of them right does not compensate for the fourth.**

The first three classes differ in whether the tunnel carried properly what it was meant to carry; the fourth differs in that **that information was never within the tunnel's remit to begin with**. It is the class most easily misunderstood by anyone who treats a VPN as a cloak of invisibility: they believe they have bought not being recognised, when what they have bought is their traffic's contents and destinations not being seen.

5.2 A kill switch costs less than people assume

A common objection is that a kill switch slows connections down. The measurements say otherwise: establishment time is **statistically indistinguishable** with and without one (s1: 53.3 vs 53.4 ms; s2: 47.8 vs 43.2 ms).

It does not make the connection slower; it converts exposure into dropped packets. In s1 the packets that would have leaked are stopped locally instead; in s3 it is the only mechanism that prevents leakage at all.

5.3 Why the "dropping out is dangerous" intuition is wrong

Most people picture VPN leakage as "the moment the connection drops, everything goes out in the clear". The s2 result shows otherwise: as long as the tunnel interface exists, packets during an outage are discarded rather than diverted.

What is genuinely dangerous is the interface disappearing — and that is **invisible** to the user. The network works perfectly, pages load, nothing indicates anything is wrong. The only clues are things the user has no reason to look at: the VPN application's icon state, or a tunnel indicator that has quietly vanished from the notification area.

That gap is itself the risk: **users judge whether they are protected by whether the internet still works, and in the most dangerous scenario that test returns the wrong answer.**

---

6. A checklist for users

The items below are independent. Please confirm each.

One — is IPv6 inside the tunnel? Confirm that the tunnel's allowed range covers both IPv4 and IPv6. If your VPN does not support IPv6, the safer course is to disable IPv6 at the operating system level rather than let it flow outside the tunnel.

Two — has the resolver been taken over? After connecting, confirm that the DNS server the system actually uses has changed to the one the tunnel provides, and is not the one the local network handed out. **Being written in a configuration file is not the same as being in effect** — during this research we encountered exactly that: the setting appeared to apply, the resolver did not change, and no error was raised. Judge by the system's actual state.

Three — has the kill switch exception been removed? If you allowed the local subnet in order to log in to a public Wi-Fi portal, **remove that exception once you are logged in**. While it exists, the kill switch is not protecting you from the local network.

Four — the local network can still recognise this device Once all three of the above are right, the untrusted local network will still obtain your MAC address, NIC vendor, hostname and local address. If your threat model includes not wanting this network to recognise this device, what you need is a device-layer remedy — randomise the MAC address, change the hostname to something that identifies nothing — **not any VPN setting**.

One further limitation is worth internalising: **you cannot tell whether you are protected by whether the internet works.** In the most dangerous scenario it works perfectly. If the situation bears on physical safety, verify your external address before each significant action rather than relying on what the application's status display says.

---

7. Recommendations for implementers

1. **Interface loss should be treated as a failure event and raise an alert.** The current behaviour is a silent fallback that the user has no way to notice. This is the most harmful and most easily remedied item in this study.
2. **Portal exceptions should expire automatically.** By time limit, or by being withdrawn once the tunnel is established. An exception that persists indefinitely disables the kill switch precisely where it is needed.

3. **Resolver takeover should be verifiable and should fail loudly.** A setting that fails to apply without producing any message leaves the user operating under a false sense of safety.
4. **Packets during establishment should be blocked by default.** The measurements show that this adds no establishment latency.
5. **The scope of a kill switch should be disclosed honestly.** A rule set acting on the IP output chain does not reach ARP or DHCP. If a product's interface presents its kill switch as "blocks all traffic", that description does not match the implementation. To cover those two as well, the rules must go down to the network device layer's egress hook.
6. **A user's hostname should not be handed to an untrusted network.** The DHCP hostname option is sent in the clear, and most users have no idea their device is broadcasting an identifying name. Products sold on privacy should offer an option to randomise the MAC address and hostname on connection, and explain its limits.

---

8. Limitations and further work

8.1 External validity (the most important item)

The subject under test is a manually configured `wg-quick` tunnel, and is not equivalent to any commercial VPN application.

Commercial products commonly implement their own kill switches, resolver takeover and reconnection logic, and their behaviour may differ entirely from this experiment — possibly better, possibly worse. This report tested no commercial product and **must not** be cited as an evaluation of one.

What this report does support is that these failure modes **are possible** at the protocol and routing level, and that the method for measuring them is workable and reproducible. Establishing how a particular product behaves requires testing that product separately — different work, requiring real devices rather than an emulated environment.

8.2 Measurement resolution

The beacon interval is 20 ms, and every millisecond figure carries that quantisation error. The 53.3 ms in s1 in truth lies somewhere between about 33 and 53 ms. A shorter interval would improve resolution at the cost of perturbing the system more.

8.3 Environmental differences

The experiment runs in an emulated network whose link latency is far below that of a real one. Establishment in the real world will take longer, so the time windows here should be read as **lower bounds** rather than typical values.

The figure "53 packets leaked over the whole run" from s1 is not presented as a conclusion, because it is an artefact of an experimental parameter — a one-second pre-connection baseline. In the real world

that period lasts as long as the user takes to connect, which may be seconds or may be the entire time since boot.

8.4 Further work

- The same measurements against commercial VPN applications on real devices
- Whether a kill switch acting at the network device layer's egress hook can cover ARP and DHCP (not tested in Section 4.4)
- The identifying fields sent by other operating systems' DHCP clients (vendor class, client identifier)
- Sleep and resume behaviour on mobile platforms
- Packaging the environment so that non-technical users can test their own setup

8.5 Feedback we would welcome

If you use this environment to test other configurations or products and reach different results, we would like to know. If you believe the method is flawed, we would equally like to know — the environment is fully public precisely so that such scrutiny is possible.

Appendix A: Reproducing this work

The topology file and all scripts are published with this report and can be downloaded from:

https://odysec.org/research/wp-002-lab-vlk-274.tar.gz

The archive unpacks to a `vlk-274/` directory.

The environment uses only the freely available Debian 13 image and **deliberately avoids any image that cannot be lawfully redistributed**, so readers can rebuild it completely. All four provisioning scripts are idempotent and repeatable; the environment was rebuilt from scratch three times during this study.

Step-by-step instructions, tool usage and log formats are in that directory's documentation.

Appendix B: Full data

Per-run results are published as JSON Lines under `vlk-274/data/` inside the archive above, 510 records in total. The aggregation script `aggregate.py` reproduces every statistic in this report and emits CSV.

Appendix C: Terminology

中文	English	日本語
隧道	tunnel	トンネル
隧道介面	tunnel interface	トンネルインターフェース
洩漏	leak	リーク
中斷開關	kill switch	キルスイッチ
允許範圍	allowed IPs	許可 IP 範囲
解析器	resolver	リゾルバ
解析器接管	resolver takeover	リゾルバの引き継ぎ
路由器公告	router advertisement	ルーター広告
信標	beacon	ビーコン
建立空窗	establishment gap	確立までの空白
入口網頁	captive portal	キャプティブポータル
對照組	control group	対照群
離散度	dispersion	ばらつき
外部效度	external validity	外的妥当性
範圍之外	out of scope	範囲外
可識別資訊	identifying information	識別可能な情報
網卡位址	MAC address	MAC アドレス
網卡廠商代碼	OUI	OUI
主機名稱	hostname	ホスト名
送出鏈	output chain	出力チェーン
原始通訊端	raw socket	生ソケット

Appendix D: Sources

All data in this report comes from a controlled environment built by the team; there are no external data sources. Protocol behaviour is described per the WireGuard documentation and the `wg-quick` manual page; address ranges per RFC 5737 and RFC 3849.

This report belongs to the same series as WP-001 *The Eyes Beside You*, with which it shares a threat model.

Citation: Odysec (2026). *Beyond the Tunnel: Measuring the Protection Boundary of a VPN*. Odysec Research Report WP-002, version 1.0.

Disclaimer: this report is technical research for educational purposes and does not constitute legal advice. For the legal assessment of any specific case, consult a qualified lawyer. If you are in immediate physical danger in Taiwan, call 110; the protection services and counselling line is 113.