RiskMandate

THE LAB · ENTRY 06 · FINDING

Do not attack anyone is the one rule everybody signs, and *it is the one with no barrier.*

Every other capability we have written up has an enumerable grant: a scope with a published list, and a delta you can count. This one does not. The grant is every address the process can route to, nobody granted it, and it is the residue of the machine having a network interface. So the measure breaks, the prohibition everybody would sign turns out to be enforced by nothing in the default configuration of every assistant we examined, and the reason to write it down anyway is evidential rather than technical.

Edition v1 · 12 September 2026

Live page riskmandate.ai/lab-network-reach.html

Published by RiskMandate · CC BY 4.0

This is a dated edition of a page that changes. The Lab holds our current thinking, and current thinking moves. This PDF does not: it is what we thought on the date stamped below, kept so the reasoning can be followed rather than only its conclusion. The live page may since have been corrected, extended or withdrawn — and if it has, the edition list on it will say so.

THE GRANT

Every routable address, and nobody granted it.

Two entries ago we could tabulate a grant. A code-host scope has a published description, a finite set of capabilities and documented exclusions; the delta between what the credential permits and what the holder was asked to do was a list, and the length of that list was the finding. Here there is no list.

- **The grant is every address the process can route to.** Every public service on the internet. Every host on whatever network the machine sits on. Every address in the private ranges reachable from there. The metadata endpoints a cloud instance exposes to its own workloads. **Nobody wrote that grant down because nobody granted it** — it is the residue of the machine having a network interface.
- **So the delta is an unbounded complement rather than a set of items,** and the measure this site has been using since Lab 04 — count the connectors, derive a number of bits — does not apply. Counting is the wrong operation on this axis, and saying so is more useful than producing a number that means nothing.

QUESTION	ON THE CODE-HOST SHAPE	ON THIS SHAPE
What does the credential permit	A published scope, enumerable	`not applicable — there is no credential`
What is the holder asked to do	A mandate, enumerable	A mandate, enumerable
What is the delta	A list, and its length is the finding	**Everything else, and its unboundedness is the finding**
What bounds it	A token scope or a branch rule	Only a boundary at the network or kernel layer
What can be measured	The number of items in the delta	**Whether a barrier exists, at which layer, and which egress paths it covers**

This is a result, not a defeat.

On the code-host shape the row says *here is a capability you did not know you had*. Here the row says there is no enumeration, and the only honest thing a policy can record is which of the four barriers stands between the mandate and the rest of the internet. That is a different kind of row, and the schema has to carry both.

● none ◉ an expectation ◐ a setting ○ a boundary — only the fourth bounds anything

CAPABILITY AND GUARDRAIL

A refusal layer belongs to one deployment.

Every major vendor's acceptable-use terms already prohibit exactly this, and those terms are backed by refusal training and by classifiers on the hosted services — more enforcement than a deployer's pasted line will ever have. None of that is evidence about the capability, and a behaviour policy must not treat it as though it were.

- **The same capability is reachable through the raw interface** without the hosted review layer, through a different provider serving a comparable family, through an open-weight model on the deployer's own hardware, through a fine-tune, and through a router that fails over to a second provider when the first is slow. Not one of those changes what the system can do. Every one of them changes whether anything refuses.
- **This is the label principle, applied exactly as it was defined.** A substance's label lists what the substance does. It does not stop listing an interaction because one hospital happens to employ a pharmacist who checks. The pharmacist is a control in one setting; the interaction is a property of the compound. Removing it from the label because of the pharmacist would make the label wrong everywhere else.

Controls on some deployments are a statement about those deployments. The capability is what the document describes.

Which means the two published incident reports have to be read the other way round from the way a security reader reaches for them — and read that way they are the strongest capability evidence in this corpus: first-party, dated, and published by a vendor about its own product.

EXECUTED INDEPENDENTLY

80–90%

Of tactical operations, in an orchestrated espionage campaign the vendor disrupted and reported. The tooling was the coding assistant plus connector servers.

RECONNAISSANCE

100s

Of discovered services and endpoints catalogued, with complete network topology mapped across multiple address ranges, across roughly thirty targeted entities.

A SECOND OPERATION

1,000s

Of remote access endpoints scanned, with vulnerable systems identified, domain controllers and database servers located, and real-time assistance during live intrusion. At least seventeen organisations affected in one month.

Both figures are the vendor's own, from its published disruption report and its August 2025 threat report, cited by date rather than characterised. The recorded remedy in both cases was detection and account closure after the fact — after roughly thirty entities had been targeted in the first, and at least seventeen organisations affected in the second.

That the guardrails were defeated is the smaller finding, and it dictates the phrasing. Both operations were defeated by reframing rather than by force: operators claimed to be employees of legitimate cybersecurity firms, presented the work as defensive testing, and decomposed multi-stage attacks into discrete technical tasks each of which looked ordinary in isolation. So an instruction saying *do not attack anyone* is satisfied, in the model's reading, by an operator who says *this is an authorised penetration test of our own estate*. **The generic line must therefore prohibit a set of destinations, which is not reframable, rather than a category of intent, which is.**

A PROPOSAL TO THE MODEL

A barrier has a position, and some have an expiry.

The four barriers were written from the agent's point of view. A vendor refusal layer exposes a second axis the vocabulary does not carry: *relative to whom*. From the deployer's seat it sits above them and they cannot switch it off, which looks like barrier four — and it formally passes

the enforcer test, because it is enforced by something the grant does not include. It is still not the same kind of object as a kernel firewall.

FIELD	KERNEL FIREWALL OR EGRESS PROXY	VENDOR
`above whom`	Above the agent and above the session	Above t and abo
`deterministic`	**Yes**	**No** — it a state
`what voids it`	Removing it, which is a deliberate act by an administrator	A model version, hosting **none o anybod a secur**
`can the party relying on it evidence that it acted`	**Yes**, there is a log it did not write	**No**

- **Perishability is the property that matters most.** A model-router configuration edit is a performance decision, usually made by somebody who has never seen the policy, frequently automatic — and it can remove the only thing standing between the mandate and the rest of the internet. A kernel firewall does not evaporate when the model changes.
- **And the party relying on it cannot evidence it.** The validation table below works because a proxy log and a resolver log are held outside the agent. There is no analogue for a refusal layer: the deployer cannot demonstrate that it acted, cannot demonstrate that it was present, and in a failover case cannot always establish afterwards which provider served a given turn.

> **So compute the minimum barrier twice, and publish the pessimistic number.**
>
> Once including perishable barriers and once excluding them. Otherwise the badge reads four, somebody edits a routing rule on a Tuesday afternoon with no ticket and no review, and the badge still reads four while the barrier is gone. **A barrier voided by a change nobody classifies as a security change must not be counted without its void condition printed beside it.** This joins our request list against the model site as a proposal, alongside the composition rules below.

THE FINDING THAT CHANGES THE RENDERING

One agent has more than one way out.

The most quotable sentence found anywhere in this research is in one vendor's permissions documentation, about its own product. It settles an argument this site keeps having, so it is reproduced verbatim with its date.

> “Note that using WebFetch alone doesn't prevent network access. If Bash is allowed, Claude can still use `curl`, `wget`, or other tools to reach any URL.”
>
> code.claude.com/docs/en/permissions, read 12 September 2026

> “Permission rules are enforced by Claude Code, not by the model. Instructions in your prompt or `CLAUDE.md` shape what Claude tries to do, but they don't change what Claude Code allows.”
>
> Same page, same date. That is a vendor describing the difference between barrier two and barrier four in its own words — and doing it more plainly than this site has managed. The same page adds, of writing the boundary into the project instructions file: “This shapes what Claude tries but doesn't enforce a boundary.”

The setting that *does* hold is a different one, and its scope is narrower than most deployers would assume. Four documented properties of it belong on any page that recommends it:

- **It covers shell subprocesses only.** In the vendor's words: “Claude Code enforces this for sandboxed commands only; in-process tools such as `WebFetch` still follow their permission rules.”
- **It does nothing when written into the repository.** Same sentence, next clause: “Setting it in a repository's `.claude/settings.json` or `.claude/settings.local.json` has no effect.” **This is the barrier-three to barrier-four promotion made mechanical** — the same words in one file are a setting the agent's own working tree could change, and in another file are a boundary above it. It is the clearest illustration of the enforcer test we have found in anybody's documentation.
- **It fails open by default.** “By default, if the sandbox cannot start because dependencies are missing or the platform is unsupported, Claude Code shows a warning and runs commands without sandboxing. To make this a hard failure instead, set `sandbox.failIfUnavailable` to `true`.”

- **It decides from the announced hostname.** “Because the proxy makes its allow decision from the client-supplied hostname without inspecting TLS, code running inside the sandbox can potentially use domain fronting or similar techniques to reach hosts outside the allowlist… Stronger TLS-aware network isolation is an active area of development.” Against an injected agent that will not think to lie about the hostname, this is adequate. Against a compromised one it is not, and a policy row claiming it should say which of the two it is claiming.

All four quoted from code.claude.com/docs/en/sandboxing, read 12 September 2026. A second vendor's defaults run the other way and are worth citing as the counter-example: its shell sandbox has network access off by default, its hosted service blocks internet access during the agent phase unless enabled, and its domain filter resolves deny before allow. A third states plainly that its automatic review of agent actions is not a security boundary and that the classifier can err in both directions. Publishing three positions side by side, sourced and dated, is more useful to a reader than any argument we could make.

> **A destination list is not a property of an agent. It is a property of an egress path, and an agent has several.**
>
> So the rendering has to be a matrix, with paths down the side, and a cell is green only when that path is covered by something at barrier four. A policy recording a single domain list for an agent is describing at most one of its exits.

EGRESS PATH	WHAT GOVERNS IT	COVERED BY	DEFAULT
`shell subprocesses`	The sandbox allowed-domains list, with strict mode, from user or managed settings	An OS-level boundary outside the process	○ 4
`in-process fetch`	Its own permission rules — a separate list that must be reconciled by hand	The client, not the OS	◐ 3
`connector servers`	Nothing in the sandbox — they run as separate host processes outside it	Only a network-layer boundary	● 1
`hooks`	Nothing in the sandbox — same reason	Only a network-layer boundary	● 1
`name resolution`	Not covered by any web-destination list	A resolver allow list, or the firewall's outbound resolution rule	● 1

The minimum across the paths is what the badge should show. **An agent with a perfectly written destination list and an unsandboxed connector server earns the glyph for the connector server,** because that is the path an attacker uses.

MEASURED ON THE MACHINE THAT BUILT THIS PAGE · 12 SEPTEMBER 2026

```
# the shell, from the environment that publishes riskmandate.ai
curl https://example.com/        200
curl https://ipinfo.io/ip        200
exec 3<>/dev/tcp/1.1.1.1/443     open   ← a raw socket, no proxy in the path
getent hosts example.com         resolves

# the in-process fetch tool, same session
fetch https://example.com/       200
```

No allow list stood in either path. This is one deployment — a managed remote environment with its own network policy — and that is exactly the point: the answer to *can this agent reach arbitrary hosts* lives in the deployment and not in the model. We publish our own measurement because a page arguing that the barrier is a property of the deployment should say what its own deployment does.

One thing we could not reproduce, and say so. The brief behind this entry reports observing the two egress paths *diverge* in a single session — the fetch tool reaching hosts the shell's tunnel refused. In this session they did not diverge: both reached every host tried. The documented finding stands on the vendor's own sentences above, which say the two paths are governed by different policy; whether they actually differ on a given day is configuration, and we did not see it here.

THE HOLE IN EVERY LIST

Name resolution is not on it.

A destination allow list that covers web traffic does not cover name resolution, and name resolution alone is sufficient to move data out. A vulnerability disclosed in May 2025 and fixed the following month used exactly that: the diagnostic commands for resolution and reachability were on the automatically approved list, and an injected instruction could embed the output of a credential search into a hostname and resolve it, delivering the secret to a server the attacker controlled. The remedy was to remove those commands from the automatically approved list.

Which makes the published reference container worth reading closely before it is copied. It is a genuine default-deny egress firewall at the kernel layer — `iptables -P OUTPUT DROP`, an allow set built from the code host's published address ranges and seven named service hosts, and a self-check that an arbitrary external host is unreachable while the code host is. We fetched the script and read it. Stated without adjectives, it permits four things a reader would not expect from a configuration whose purpose is to stop the agent reaching other people's machines.

permits

Name resolution to any address

`iptables -A OUTPUT -p udp --dport 53 -j ACCEPT` — added before the allow set is built. This is the exact channel the May 2025 vulnerability used.

permits

Secure shell to any address

`iptables -A OUTPUT -p tcp --dport 22 -j ACCEPT` — unconditional, to anywhere on the internet.

permits

The code host's full address ranges

Every repository and every snippet on it, which is therefore an outbound channel for anything the agent can read.

permits

The whole /24 around the host

`HOST_NETWORK=$(echo "$HOST_IP" | sed "s/\.[0-9]*$/.0\/24/")`, then accepted. On a typical developer machine that is the local network the developer is sitting on.

Read from the published script on 12 September 2026. It also rejects with --reject-with icmp-admin-prohibited and configures no logging target, so the validation table below loses a row unless one is added.

A document that says *do not touch any other company's systems*, running behind that firewall, can still reach every other machine on the office network.

Whether that is acceptable depends entirely on the deployment — which is consequence agnosticism doing exactly the job it was introduced to do. The same configuration is unremarkable on an isolated build host and is a very different object on a laptop plugged into a corporate network. **This is the strongest case yet for the deployment-shape field being mandatory rather than optional.**

THREE WAYS IT HAPPENS

Somebody else's machine gets hit without anyone intending it.

These three are documented, dated and different from each other in a way that matters: they need three different rows, and conflating them is the mistake to avoid.

01 The agent walks around its own restriction

In an account published in August 2026 by an independent researcher, the user prompt was benign — summarise this URL. The in-process fetch tool returned an HTTP 415, the assistant reasoned that it should try directly, and fell back to the shell. A redirect steered it to an archive, which it downloaded, extracted and executed, establishing an outbound callback. The reviewing classifier approved the execution and then denied the subsequent cleanup command.

Defeated by: an egress boundary. The vendor's position, as reported in that account, is that the automatic mode is a best-effort classifier and not a security guarantee, with operating system isolation and network controls being the actual boundary. That position is correct, and it is the whole argument of this entry.

02

The agent is turned into a reflector

In a January 2025 advisory, an endpoint operated by a model vendor accepted a list of URLs and issued one outbound request for each, from the vendor's own address ranges, with no deduplication and no observed cap. A single well-formed request was amplified into fifty requests aimed at a chosen third party. The endpoint was disabled after the report became public.

Not the deployer's exposure at all. This belongs on a provider page rather than a deployer one — the deployer's instruction file was never in the path. It is the clearest documented case of legitimately operated infrastructure being aimed at an uninvolved party by a single untrusted input.

03

The whole permission layer is switched off by something already trusted

In the supply-chain compromise of August 2025, a malicious package version shipped an install hook that invoked the developer's already-installed assistant command-line tools **with their confirmation flags disabled**, and handed them a prompt instructing a recursive search of the home directory and configuration paths for credential and wallet file patterns. Over a thousand valid tokens were leaked and thousands of files exfiltrated, with a second wave making several thousand repositories public.

Defeated by an egress boundary and by nothing else. Every prohibition in every instruction file on those machines was in force and none of them applied, because the flag that disabled confirmation was passed by a process the developer had already granted execution to. The agent layer was not merely bypassed — it was operated.

WHY WRITE IT ANYWAY

The answer is evidential, not technical.

If the instruction does not stop the agent, the honest question is why a deployer should write it. The answer is that in two of the three legal regimes examined, liability attaches to *causing an act to be done* — and the document is the record of what was authorised.

- **In one jurisdiction**, the impairment offence is committed by doing an unauthorised act in relation to a computer knowing it is unauthorised, with intent to impair operation, prevent or hinder access to data, or impair the reliability of data. Two sub-clauses do the work: recklessness as to whether the act will do any of those things is sufficient in place of intent, and a reference to *doing an act* includes *causing an act to be done*, with an act including a series of acts. The prosecuting authority's own guidance states that the related access offence is made out once a defendant has caused a computer — including his own — to perform a function with the relevant intent, and that the intent need not be directed at any particular program, data or computer.
- **In a second**, the corresponding provision covers knowingly causing the transmission of a program, information, code or command which intentionally causes damage without authorisation to a protected computer, with damage defined as any impairment to integrity or availability. The controlling interpretation of the access clauses is a gates-up-or-down inquiry, so fetching a public page is unlikely to qualify while getting past an authentication gate or causing impairment is.
- **In a third regime**, the access offence requires that a security measure be infringed, which likely places bare scanning outside it — but the same instrument carries a corporate liability article under which a legal person is liable where a *lack of supervision or control* made the offence possible for its benefit. That clause is the direct hook for a deployment with inadequate egress control.

> The tribunal called the submission “remarkable”, and said it should be obvious that the operator is responsible for all the information on its own site.
>
> An airline argued before a small claims tribunal in 2024 that it could not be held liable for information provided by its own conversational system. Quoted here from secondary coverage, because the primary record blocks automated retrieval — and it is a small claims tribunal on negligent misrepresentation rather than a computer misuse case. Cited with that caveat, because it is the only public judicial rejection of *the automated system acted on its own* that we could find, and the reasoning transfers.

What was not found is itself the finding. No court judgment, no regulator decision and no prosecutorial guidance in any jurisdiction examined addresses computer misuse liability specifically for an autonomous agent's network activity. The vendor disruption reports are the closest thing to a public record and they describe law-enforcement coordination rather than a charging decision. There is also no appellate authority either way on the legality of unauthorised scanning. **None of this is legal advice**: the statutory language is quoted, the gap in the case law is reported, and whether a given deployment is exposed is a question for the deployer's own counsel.

THE RISK OF WRITING IT WITHOUT ENFORCING IT

There is a version of this that cuts the other way, and this page would rather say it than be caught out by it. **A written prohibition is evidence that the operator foresaw the risk.** An operator who wrote *do not scan third-party hosts*, enforced nothing, and whose agent then scanned third-party hosts, has produced a document that establishes foresight on the record. On a standard where recklessness suffices, foresight without mitigation is not obviously better than no document at all.

The resolution is not to stop writing the line. It is to **never publish a prohibition without publishing what enforces it in the same row.** A row reading *do not reach hosts outside the list — enforced by nothing* is an honest disclosure of an unmitigated exposure, and reads as a decision taken with open eyes. A row with no enforcement column at all reads as an operator who believed the sentence was a control. That column is what converts the document from evidence against the operator into evidence of diligence — and it is the strongest argument yet found for this site's rule that every line carries its barrier.

VALIDATION

Self-report is a claim. The log is the evidence.

Can the agent report what it reached? Yes, and it is not worth much. Can it be validated? Yes — and every source that counts is held outside the agent.

- **Why self-report is not evidence, demonstrated first hand.** While researching the brief behind this entry, a research agent asked its own fetch tool to reproduce a vendor's documentation of that same fetch tool's behaviour. On one attempt the extraction returned a fluent, confident and entirely fabricated answer: a redirect limit that does not exist, a cache duration off by a factor of four, a parameter that does not exist, and a rejection behaviour that is the opposite of the documented one. The same agent had reported the section accurately before and reported it accurately again after. **One agent, one question, one session, three answers, one of them invented.** An agent's account of what it fetched is a claim about the past produced by the same machinery that produced the fetches.

- **And the telemetry has four gaps that a validation claim would fall through.** The tool-result event is not emitted when a call was rejected, so a naive reading undercounts attempts. There is no network event at any level — a shell call produces one record containing a command string, and the connections its child processes open are not recorded, so a command that reads a URL from a variable, a script or a compiled binary produces a log line with no destination in it. Values are truncated. And the stream is emitted by the agent's own process: it is self-report with better formatting, and a process that can write files can change where it is sent.

SOURCE	WHAT IT PROVES	CAN THE AGENT EDIT IT
`egress proxy connection log`	Every hostname the process asked to reach, allowed and refused, with timestamps	**No**, when the proxy runs outside the sandbox
`name resolution query log`	Every name looked up — including the channel the web list does not cover	**No**
`kernel firewall logging`	Every packet allowed or dropped, by address	**No** — and the reference container rejects without logging, so a target must be added
`cloud network flow records`	Address-level record of everything the workload sent	**No**
`agent telemetry, tool detail on`	What the agent asked its own tools to do	Yes, in principle
`the agent's narrative answer`	Nothing	Not applicable — it is generated

Two rows, never one.

The agent will report the destinations it *intended* to reach, and that report is a claim. The destinations it *actually* reached are in the following log, held by the following party. One vendor's hosted sessions state that all outbound traffic passes through a proxy providing a resolution-level audit trail of requested hostnames — which is exactly the artefact, and its presence for hosted sessions and absence for local ones is the single most useful difference between the two deployment shapes on this capability.

A PROPOSAL TO THE MODEL

A policy made of policies, composed the safe way round.

A larger policy composed of smaller ones is what makes the approach scale. The composition semantics are the whole question, because two well-known systems compose in opposite directions and only one of them is safe here.

> “When you use a policy to set the permissions boundary for a user, it limits the user's permissions but does not provide permissions on its own.” … “The effective permissions are the intersection of both policy types. An explicit deny in either of these policies overrides the allow.”

The precedent that works — permissions boundaries at docs.aws.amazon.com, read 12 September 2026. The first sentence is precisely the semantics a nested policy needs: the parent's list is a ceiling and the child can only narrow it.

> “Network policies do not conflict; they are additive.” … “the connections allowed in that direction from that pod is the union of what the applicable policies allow. Thus, order of evaluation does not affect the policy result.”

The anti-pattern, and instructive precisely because it looks like the right thing — at kubernetes.io, read 12 September 2026. Its own list of what it cannot do includes denying traffic — “there is no concept of a default deny which is overridden by an allow policy” — and name-based targeting. Union of allowances with no denial primitive means adding a policy can only *widen* the effective reach, which is the exact opposite of what a nested policy needs.

1 **Denial by default.** The absence of a rule is a refusal and never a permission. An empty policy means no reach.

2 **Intersection, never union.** The effective set is the intersection of every applicable layer — the organisation's managed layer, the project layer, the session layer, and any sub-agent's own layer.

3 **A child is a ceiling, not a grant.** A nested policy can only narrow. It cannot add a destination its parent did not have.

4 **Explicit denial is terminal at any layer.** No combining algorithm at any node may override it — which is the one place this departs from the formal standard for nested policy sets, where a per-node algorithm can let a child widen its parent.

5 **Ambiguity narrows or halts, never widens.** An unparseable or ambiguous entry is dropped, or the agent stops. It is never resolved by guessing. There is already a shipped precedent for this: one vendor documents that for an ambiguous address entry, “Claude Code never allows more than you wrote” and “may drop the entry entirely rather than widen the allowlist”, while denying every reading the entry parses as.

6 **Every layer records the barrier it binds at**, and a layer binding at barrier four carries the identifier of the log that proves it. Composition of instructions is composition of nothing unless at least one layer in the chain is enforced above the agent.

> **Ten nested policies at barrier two compose to a barrier two result.**
>
> Which is why rule six is the one that matters. The fractal structure buys maintainability and delegation, which is real value, and it does not manufacture enforcement. The barrier therefore has to be a required field on every node, and the rendering has to compute the minimum across the chain and print it at the top.

THE DRAFT

Generic and custom, with every line marked.

The generic version is for anybody to paste, and the point of it is the right-hand column rather than the left. Read that column first: on this capability, most of it says nothing, which is the harder and more honest version of the same table in Lab 05.

GENERIC LINE	WHAT IT IS	ENFORCED BY	BARRIER
Do not send traffic to any host you have not been asked to reach	A boundary on destinations, phrased so it cannot be reframed as authorised testing	**Nothing.** Model disposition only	◉ 2
Do not scan, probe or enumerate hosts, ports or paths	A boundary on a technique	**Nothing**	◉ 2
Do not follow a redirect to a host outside the list without asking	A procedure	Partly, where the in-process fetch tool returns cross-host redirects as text instead of following them	◐ 2→3
Do not use shell network utilities	A boundary on tooling	A deny rule on the command text — which the vendor documents as not matching "the same program by path or inside `sh -c`"	◉ 2
Report every destination you reached at the end of the session	A reporting duty	**Nothing.** The report is a claim	◉ 2

The custom version is the one worth something, and it has three destination sets with three different lifetimes: **operational** (registries, the model API, telemetry, the source host — fixed, and it should be short), **estate** (the deployer's own systems — generated from an inventory rather than typed), and **handed in** (whatever the user pastes in this morning — one turn, and it cannot be a list at all, so its line is a procedure).

CUSTOM LINE	WHICH SET	ENFORCED BY	BARRIER
Operational destinations, a short fixed list	What the agent needs to function	**The sandbox allowed-domains list**, set in user or managed settings, strict mode on, sandbox failure treated as a hard failure	○ 4
The same list for the in-process fetch tool	The second egress path	Fetch permission rules with a domain prefix — a **separate list** that must be reconciled by hand	◐ 3→4
Connector server destinations	The third egress path	**Nothing in the sandbox**, because connector servers run as separate host processes outside	● 1

CUSTOM LINE	WHICH SET	ENFORCED BY	BARRIER
		it. Only a network-layer boundary covers this path	
`Estate destinations, generated from an inventory`	The deployer's own systems	The same two lists, plus the network layer	○ 4
`Handed-in destinations — a procedure, not a list`	What the user pastes in today	An approval step at the moment of use, recorded	◐ 3
`Name resolution destinations`	The channel the web list does not cover	A resolver allow list, or the firewall's outbound resolution rule — which the reference container leaves open to any address	● 1
`Everything else`	The unbounded complement	A default-deny egress firewall or an allow-listing proxy outside the process	● 1

> **Fill in the enforcer column first.**
>
> Write the prohibition only for the rows where you can name something. For the rows where the enforcer is nothing, **still write the prohibition, and print the word *nothing* next to it** — because that row is the finding.

OPEN QUESTIONS

Six, and two of them change the vocabulary.

Real rather than rhetorical. The first two would alter a model this site has been building on for a month, which is why they are not settled here.

#	QUESTION	WHY IT IS HARD
1	**Two kinds of barrier now sit awkwardly at level four.** Hostname enforcement is real but defeatable by the process it constrains; a vendor refusal layer is real but probabilistic, perishable and unevidenceable. A fifth level, a qualifier, or the three companion fields?	Adding a level damages a vocabulary whose value is that it has four. The companion fields keep the four intact and make every rendering wider
2	**Should a perishable barrier count at all?**	Computing the minimum twice and publishing the pessimistic number is the proposal here. Refusing to record a barrier the deployer cannot evidence is cleaner and discards real information
3	**Who owns the reconciliation between an agent's several destination lists?**	It implies a derived object — the union of the paths and the intersection of the lists. Nobody generates that today, and it may be the smallest useful tool we could ship on this capability
4	**Should the generic document be published at all before the custom generator exists?**	It is the better acquisition object and the weaker artefact. Publishing it alone risks being the organisation that handed out a sentence and called it a control
5	**Can the delta be expressed as a measure rather than a count on unbounded capabilities?**	One candidate: the number of egress paths not covered at barrier four. Small, integer, comparable across deployments, and it does not pretend to count the internet
6	**Is there a defensible position on the local network?**	Every reference configuration examined permits the surrounding local network, usually without saying so in prose. A separate row for private address space is a design decision with a legal dimension, because the private range is where the authorisation question is least ambiguous

Four honest tensions, stated rather than resolved. The generic document is the most useful thing to hand out and the least defensible thing to stand behind — handing out a barrier-two artefact as a first contact is defensible only if the artefact says so on its face, which makes it a worse marketing object and a better document. Writing a prohibition can worsen the operator's position, and the resolution proposed above has not been tested against anybody's counsel. Recording a capability that a particular deployment reliably refuses can read as scaremongering, and that complaint is not unreasonable — it is why the barrier column and the void condition are a requirement of honesty rather than a nicety. And this site and the vendors will appear to contradict each other while making claims about different objects: we publish that the capability is present, sourced to a vendor's own reports; a vendor states, correctly, that its hosted service carries controls against it. Both are true and neither refutes the other. The mitigation is structural — capability and barrier are separate fields on the same row, never one sentence, and the barrier field names the deployment it describes.

What this entry deliberately is not. It is not a security assessment of any named product: every claim about a product is a quotation from that product's own published documentation with a date, and where a default is unfavourable it is stated without an adjective. It is not legal advice. It is not a claim that prompts are worthless — Lab 05 has the numbers, and the narrower claim here is that on this one capability the stated prohibition has already been defeated in production at a stronger layer than a deployer can reach. And it is not a proposal to build an egress proxy: four mechanisms that already work are named above, and our contribution is the document recording which one is in place.

Written 12 September 2026 from a dev brief of the same date, archived in full on the brief register. Load-bearing quotations fetched and checked on that date: the statement that WebFetch alone does not prevent network access, that permission rules are enforced by the client and not the model, that CLAUDE.md guidance does not enforce a boundary, and that a Bash deny rule does not match the same program by path or inside sh -c, at code.claude.com/docs/en/permissions; the strict-allowlist scope, its no-effect-in-a-repository clause, the fail-open default and the hostname/fronting limitation at code.claude.com/docs/en/sandboxing; the four permissions of the reference container firewall, read from its published script; the permissions-boundary semantics at docs.aws.amazon.com; and the additive-union model with its absent denial primitive at kubernetes.io. The incident reports, the disclosed vulnerabilities, the statutory provisions and the tribunal decision are cited as published by their authors on the dates given and are not re-derived here; the brief carries their URLs. The measurement of this machine is our own, taken in the session that produced this page. The four barriers and the enforcer test are from abp.sgit.ai.

Count the exits, not the internet.

You cannot enumerate this grant and you should stop trying. What you can count is how many of your agent's egress paths are covered by something outside the process — shell, in-process fetch, connector servers, hooks, name resolution. That number is small, it is an integer, and on most deployments today it is zero.