Attack paths for a Microsoft Foundry penetration test

Fifteen paths an attacker can take against a Foundry deployment, written for scoping. Each one says where the attacker starts, why the path can work, which control should stop each step, what the tester checks, and what a pass looks like.

How to use this page

Most Foundry risk sits in how the platform is configured and how agents are wired to tools and data, not in the model itself. A useful test therefore needs three things a normal web application test does not: accounts in several starting positions, an inventory of every place an agent reads text from, and canary data the tester can try to move without touching anything real.

To scope an engagement, pick the starting positions the client cares about, take the paths that begin from them, and adjust the effort for the number of agents and tools in scope. The summary table gives a priority for each path with the reasoning behind it. The threat IDs (E01, I04 and so on) match the threat library in the guide, so findings can be traced back to the control set.

The paths assume the reference design from the guide: Foundry and its stores behind private endpoints, local auth disabled, agents injected into a delegated subnet, APIM in front, and Prompt Shields switched on. Where a client has not built that, the early paths will find it quickly and the later ones become easier.

Rules of engagement that shape the scope

Microsoft no longer needs to be told about a test of Azure resources you own, but every test is bound by the Microsoft Cloud Unified Penetration Testing Rules of Engagement. Several of those rules change how specific paths have to be run.

Rule	Effect on this engagement	Paths affected
Third parties need explicit written authorisation from the resource owner	The statement of work names the subscriptions, Foundry resources, projects and agents in scope. Microsoft does not authorise on the client's behalf.	All

Rule	Effect on this engagement	Paths affected
Denial of service testing of any kind is prohibited, including tests that simulate it	Spend and rate limits are tested by lowering a limit in a test APIM product and sending a short, small burst. Nothing tries to exhaust deployment capacity.	AP08
No network intensive fuzzing or automated testing that generates excessive traffic	Automated jailbreak and injection corpora run at a modest, agreed rate.	AP03, AP04
Only credentials and secrets you own may be used	Leaked credential testing uses credentials the client plants and owns. Anything else the tester finds is reported, never used.	AP02
Attempts to break out of AI system boundaries and shared service containers are encouraged, with responsible disclosure and immediate stop on success	Sandbox and runtime boundary tests are allowed, but the moment the tester reaches Microsoft's platform they stop and report to MSRC. No post exploit action against Microsoft assets.	AP09, AP10
Testing tenants, data or storage you do not own is prohibited	Name guessing in AP01 stops at any name that resolves but is not in the client's inventory, because it may belong to another customer.	AP01
Microsoft may interrupt activity flagged by its abuse detection	Keep the authorisation letter to hand. Harmful content prompts in AP03 may trip abuse monitoring on the client's subscription; the client should know in advance.	AP03

Source: Microsoft Learn, Penetration testing, and the Microsoft Cloud Unified Penetration Testing Rules of Engagement, which is the authoritative text.

Starting positions

Each path starts from one of six positions. The client provides the access for each position the scope includes; most of the work in an AI test is from the inside, so an internet only test misses the paths that matter most.

Position	Who this represents	What the client provides
P1	Anyone on the internet, with no account	The list of public DNS names in scope; any test credentials to plant

Position	Who this represents	What the client provides
P2	An ordinary signed in user of an agent or app	Two user accounts in different groups, so cross user access can be tested
P3	Someone who can write text an agent will later read: a supplier uploading documents, an emailer to a shared mailbox, an editor of a SharePoint site or wiki	Write access to each grounding source in scope, and an agreed canary payload set
P4	A developer, or an attacker holding a developer's session	An account with Foundry User on a development project, as real developers get
P5	Someone with write access to the infrastructure repository or a pipeline	A sandbox copy of the repository and pipeline, or contributor access to the real one with a rollback plan
P6	A foothold on the network: a VM in another spoke, an on premises host, or a workload in AWS or GCP	A test host in each network location in scope, with an ordinary user identity

Summary of paths

Priority combines how likely the path is in practice with what it costs if it works. The reasoning column says why, so a client can move a path up or down for their own estate.

ID	Path	Start	Priority	Why that priority	Days
AP01	Exposure sweep from the internet	P1	High	Cheap to run and finds the most common gap: a connected store left public while Foundry is private	1
AP02	Planted credentials to model and data access	P1	High	Leaked secrets are the most common real world route to running someone else's models	1
AP03	Jailbreak and system prompt extraction	P2	Medium	Very likely to partly succeed, but impact depends on what the prompt holds and what tools exist	2
AP04	Indirect prompt injection into tool use	P3	Critical	The attacker never signs in, the victim is an innocent user, and the damage is whatever the agent's tools can do	3

ID	Path	Start	Priority	Why that priority	Days
AP05	Data leak through rendered output	P3	High	Server side controls never see the leak; only the client can stop it	1
AP06	Cross user access through retrieval and threads	P2	Critical	Retrieval trimming is application logic and is often missing on one tool or one index	2
AP07	Confused deputy through the agent identity	P2	High	Agents are commonly given the union of every user's rights	1.5
AP08	Write actions without approval, loops and spend limits	P2, P3	High	Approval is configured per tool and usually missed on the tool that matters	1.5
AP09	Tool supply chain and request forgery from the agent subnet	P3, P4	High	The agent runtime sits inside the client network, so any URL fetching tool becomes an internal request	2
AP10	Egress bypass through allowed destinations	P3, P4	High	Wildcard cloud FQDNs in firewall rules let data reach an attacker's own Azure resources	1
AP11	Developer privilege escalation and guardrail tampering	P4	High	Convenience role grants are common, and one change can disable every guardrail at once	2
AP12	Unapproved training and model deployment	P4	Medium	Default roles allow fine tuning; impact is regulatory rather than immediate	1
AP13	Pipeline identity abuse	P5	High	Pipelines hold the highest standing rights and federation subjects are often too broad	1.5
AP14	Network foothold to private endpoints	P6	Medium	Private endpoints are reachable from more places than their owners think, but Entra ID still applies	1.5
AP15	Detection and response validation	All	High	A control that blocks silently looks the same as one that never ran	2

The attack paths

AP01 Exposure sweep from the internet

AP02 Planted credentials

AP03 Jailbreak and prompt extraction

AP04 Indirect prompt injection

AP05 Leak through rendered output

AP06 Cross user access

AP07 Confused deputy

AP08 Unapproved writes and loops

AP09 Tool supply chain and request forgery

AP10 Egress bypass

AP11 Developer escalation

AP12 Unapproved training

AP13 Pipeline identity abuse

AP14 Network foothold

AP15 Detection and response

AP01 Exposure sweep from the internet

Start	Target	Priority
P1, no account	**Foundry and every connected resource**	**High**
Estimate		
1 day		

Goal. Find any Foundry, AI Search, Storage, Cosmos DB, Key Vault, Container Registry or APIM endpoint that answers from the internet, or that accepts a key.

Why this can work

Foundry's connected resources are separate Azure resources with their own network settings. Teams lock down the Foundry account because it is the thing called "AI", then leave the AI Search index or the storage account public because someone needed portal access during the build. The data the agent reads lives in those stores, so the private Foundry endpoint protects very little on its own.

Names are also predictable. Entra ID authentication and private link both need a custom subdomain on the Foundry account, and organisations name resources by a convention, so an attacker who knows one name can guess the rest. Public DNS for a resource with a private endpoint still returns a `privatelink` CNAME, which confirms the resource exists without saying whether public access is off. The only way to know is to send a request.

Chain

Step	Attacker action	Control expected to stop it	Tester records
1	Build candidate names from the naming convention and resolve them under `services.ai.azure.com`, `openai.azure.com`, `cognitiveservices.azure.com`, `search.windows.net`, `blob.core.windows.net`, `documents.azure.com`, `vault.azure.net`, `azurecr.io`, `azure-api.net`	None. Existence is expected to be discoverable	Every name that resolves, its CNAME chain, and whether it is on the client inventory
2	Send an unauthenticated request to each endpoint	Public network access disabled, returning a network deny rather than an authentication error	Status code and message per endpoint; a 401 means the endpoint is

Step	Attacker action	Control expected to stop it	Tester records
			reachable and only authentication stands in the way
3	Send a request with a planted key or SAS token from AP02	Local auth disabled on Foundry, AI Search and Cosmos DB; shared key access off on Storage	Whether any key is accepted
4	List anonymous blob containers and check for a static website endpoint	Blob public access disabled at the account	Any readable container
5	Check APIM for an external gateway, a public developer portal, or a Front Door route	APIM in internal mode; any internet route behind WAF and approved	Each internet reachable API and what it fronts

Pass looks like

Every in scope endpoint refuses the request at the network layer, no key is accepted anywhere, and no container is readable anonymously. Anything that returns 401 rather than a network deny is a finding even if authentication held, because the design intent was no internet path at all.

Rules of engagement. Only probe names on the client's inventory. If a guessed name resolves but is not on the inventory, stop: it may belong to another customer, and testing it is prohibited.

Maps to

`E10` `E14` `I11` `ATT&CK T1526` `ATT&CK T1580` `ATT&CK T1190`

AP02 Planted credentials to model and data access

Start	Target	Priority
P1, holding a leaked secret	**Model deployments, stores, service principals**	**High**
Estimate		
1 day		

Goal. Show whether a secret that escapes into a repository, notebook, ticket or pipeline log gives anyone access to models or data, and whether anyone notices.

Why this can work

Running someone else's models with a stolen key is the most common real world abuse of AI services, because it needs no exploit at all. A Foundry account with local auth disabled has no keys to steal, but the estate still holds other secret material: storage SAS tokens in data science notebooks, AI Search admin keys if local auth was left on, client secrets for service principals that hold Foundry User, and connection secrets held by older hub based workspaces.

A client secret is the dangerous one. It works from anywhere on the internet unless Conditional Access for workload identities restricts where it can sign in from, which needs Workload ID licensing and is often not configured. The private endpoint then becomes the last control: the attacker can get a valid token but has no network path to use it.

Chain

Step	Attacker action	Control expected to stop it	Tester records
1	Search repositories, wikis, pipeline logs and notebooks the tester was given read access to	Secret scanning with push protection	Every secret found, where, how old, and whether scanning flagged it
2	Call the model endpoint with a planted key	Local auth disabled	Response
3	Request a token from the internet with a planted service principal secret	Conditional Access for workload identities limiting sign in location; short secret lifetime	Whether a token is issued, and the sign in log entry

Step	Attacker action	Control expected to stop it	Tester records
4	Use any token obtained against the Foundry and store endpoints from the internet	Public network access disabled	Response per endpoint
5	Use a planted SAS token against storage	Shared key access off, which also invalidates account SAS	Response

Pass looks like

No planted secret gives data or model access from the internet, and each planted secret raised an alert (secret scanning, Entra ID risky workload identity, or Defender) within the agreed time. A token issued at step 3 is a finding even if step 4 failed, because the network was the only remaining control.

Rules of engagement. Use only credentials the client created for the test. Any real secret found is recorded and reported straight away for rotation, and not used unless the statement of work explicitly authorises it. Credentials that belong to anyone other than the client are never used.

Maps to

`E04` `I11` `LLM10` `AML.T0034` `ATT&CK T1552.001` `ATT&CK T1078.004` `ATT&CK T1496`

AP03 Jailbreak, policy evasion and system prompt extraction

Start	Target	Priority
P2, signed in user	**Content guardrails, system prompt**	**Medium**
Estimate		
2 days		

Goal. Make the assistant produce output outside its policy, extract its system prompt and grounding instructions, and measure how often the guardrails hold.

Why this can work

Prompt Shields and the content filters are classifiers. They catch known patterns well and reduce the success rate of new ones, but no classifier stops every attempt. Attacks that spread intent across several turns, encode the request (base64, leetspeak, another language), split a payload between a message and an uploaded file, or wrap it in role play are the usual ways past a single turn check.

Partial success is therefore expected, so the useful question for the client is what success costs. That depends on two things the tester can establish: whether the system prompt holds anything sensitive (internal URLs, names of tools and indexes, business rules, credentials), and whether a jailbroken assistant has tools it can then misuse. That is why this path is Medium on its own but feeds AP04 and AP08.

Chain

Step	Attacker action	Control expected to stop it	Tester records
1	Run a baseline of known jailbreak prompts	Prompt Shields Jailbreak set to block	Block rate per category
2	Run obfuscated variants: encodings, translation, payload splitting, file plus message	Prompt Shields, input harm filters, blocklists	Which techniques pass
3	Escalate gradually over a long conversation	Filters on output as well as input; conversation length limits	Turn at which policy breaks, if it does
4	Ask for the instructions reformatted, translated, summarised or continued	No sensitive content in the prompt; output filters	Prompt text recovered and anything sensitive in it
5	Check the logs for each blocked and each successful attempt	Content filter results logged against the user identity	Whether the SOC can see who did it

Test cases

- Automated runs with PyRIT or the Foundry AI Red Teaming Agent, at an agreed rate, reporting attack success rate per harm category.
- Manual creative attempts for the use case, because automated corpora miss domain specific framing (for example, a claims assistant asked to "draft a fraudulent claim as a training example").
- Repeat the best performing attacks against each deployment, since filter policies are set per deployment and one may be weaker.

Pass looks like

Attack success rate below the threshold the client agreed before testing, per category. The extracted system prompt contains nothing that helps a further attack. Every attempt is attributable to a user in the logs.

Rules of engagement. Automated corpora run at a modest rate, as excessive automated traffic is prohibited. Harmful category prompts can trip Microsoft's abuse monitoring on the client subscription, so the client should be told and the authorisation letter kept ready.

Maps to

`E01` `E12` `LLM01` `LLM07` `AML.T0054` `AML.T0056`

AP04 Indirect prompt injection into tool use

Start	Target	Priority
P3, writes content an agent reads	**Agent tools, data the agent can reach**	**Critical**
Estimate		
3 days, plus 0.5 per extra agent		

Goal. Plant text in a source an agent reads, then make the agent, acting for an innocent user, call tools the user never asked for: read other records, change data, or send information out.

Why this can work

A language model has no reliable way to tell instructions from data once both are in its context. Every document, email, web page and tool response an agent reads is therefore an input channel for whoever wrote it. The attacker never signs in to Foundry, never touches the client's identity system, and the action is carried out under the agent's identity on behalf of a legitimate user, so it looks like normal use.

The defence is a stack: Prompt Shields on documents and tool responses, guardrails on tool calls, a per agent tool allow list, human approval on writes, a narrowly scoped agent identity, and egress filtering. The test establishes which of those layers exist for each agent and how many have to fail before something happens. That is the single most useful result a client can get from an AI pen test, which is why this path carries the most time.

Scoping depends on the read surfaces. Before quoting, list every agent, every grounding source and every tool response it consumes, and mark which can be written by someone outside the trust boundary: suppliers, customers, all staff, or the public web through Grounding with Bing.

Chain

Step	Attacker action	Control expected to stop it	Tester records
1	Map read surfaces: each agent's indexes, connected sites, mailboxes, web grounding and tools, and who can write to each	Grounding only from curated sources	Writable surfaces per agent
2	Plant canary payloads using hiding techniques: white on white text, HTML comments, alt text, document properties, Unicode tag characters, text inside images	Ingestion scanning; Prompt Shields Indirect Attack on documents	Which techniques survive ingestion into the index

Step	Attacker action	Control expected to stop it	Tester records
3	As a second, innocent user, ask the agent a normal question that retrieves the planted document	Tool response guardrail with Indirect Attack set to block	Whether the payload reached the model, read from the trace
4	Payload instructs a tool call outside the user's request: look up another customer, update a record, send a message	Tool allow list; human approval on writes; guardrail on tool calls	Which calls were attempted and which executed
5	Payload instructs the agent to place canary data in a tool parameter or URL bound for a tester owned endpoint	Agent identity scope; egress firewall	Whether the tester endpoint received anything
6	Leave the payload in place and ask from other accounts over the following days	Detection and removal process	Time until detected and removed; number of users affected meanwhile

Pass looks like

- No tool call outside the user's intent executes without a person approving it.
- No canary data reaches the tester endpoint.
- The payload is flagged in the logs, and the SOC is alerted to a tool call following an Indirect Attack annotation.
- The client can find and remove a planted document from every index it reached.

Any single layer failing is a finding; the report states how many layers remained.

Rules of engagement. Payloads are benign and canary based: the instruction is to move a seeded canary record, never real data. Destination endpoints are owned by the tester and agreed in advance. All payloads are removed at the end and their removal confirmed in each index.

Maps to

`E02` `E09` `I03` `LLM01` `LLM06` `LLM08` `AML.T0051.001` `AML.T0053` `ATT&CK T1213`

AP05 Data leak through rendered output

Start	Target	Priority
P3, or P2 with a direct prompt	**Chat client, link previews**	**High**
Estimate		
1 day		

Goal. Make the model write a link or image whose URL carries conversation data, so the user's browser or chat platform sends it to the attacker.

Why this can work

This leak happens after every server side control has finished. Foundry returns text; the client turns markdown into an image tag; the browser fetches the image; the query string carries the data. Nothing in Foundry, APIM or the firewall sees the browser's request, because it leaves from the user's device.

Chat platforms add a second channel. When an agent posts into Teams or Slack, link unfurling makes the platform's own servers fetch any URL in the message to build a preview, with no click from the user. The client control is therefore the only control, and it has to cover every surface the agent posts to.

Chain

Step	Attacker action	Control expected to stop it	Tester records
1	Through an AP04 surface, instruct the model to end its answer with an image whose URL includes the user's question or a canary value	Tool response guardrail	Whether the instruction reached the model
2	Check whether the model wrote the link	Output filters, which are weak against this	Model output
3	Open the answer in each client: web app, Teams, Slack, email digest	Client renders as text or proxies images; restrictive image policy in the client's own CSP; unfurling off for agent messages	Requests received at the tester listener, with source IP showing browser or platform

Pass looks like

The tester listener receives nothing from any client, for images, links or previews.

Maps to

E03 E15 LLM02 LLM05 AML.T0057 ATT&CK T1567

AP06 Cross user access through retrieval and threads

Start	Target	Priority
P2, two users in different groups	**AI Search indexes, threads, files**	**Critical where data is personal or confidential**
Estimate **2 days**		

Goal. As user A, read documents only user B may see, or read B's conversations and uploaded files.

Why this can work

AI Search does not know who the end user is. Security trimming works only if the application or agent adds a filter on an access control field, built from the signed in user's groups, to every query. If one tool omits the filter, if the filter is built from parameters the model chooses, or if a second index was added without access control fields, the search identity returns everything it can see, which is usually everything.

Threads and files have the same weakness at the API layer. If the client application accepts a conversation or file ID from the browser and passes it on without checking who owns it, any user can read any thread by changing an ID. The platform stores the data correctly; the application decides who sees it.

Chain

Step	Attacker action	Control expected to stop it	Tester records
1	Client seeds canary documents with distinct access lists for A and B	Not applicable	Seed set
2	As A, ask questions only B's documents answer	Security trimming filter on every retrieval tool	Any B only content returned
3	As A, ask the agent to search "without filters", widen the scope, or use another tool or index	Filter applied in code from the token, not chosen by the model	Whether model wording changes the result
4	Replay A's API calls with B's conversation, thread, run and file IDs	Ownership check per request; session isolation per user	Status codes and any content returned

Step	Attacker action	Control expected to stop it	Tester records
5	Upload a file as B, then reference it from A's session	File access scoped to the uploading user	Result

Pass looks like

No B only content reaches A by any wording, tool or ID manipulation.

Maps to

E12 E13 I06 LLM02 LLM08 ATT&CK T1213 ATT&CK T1530

AP07 Confused deputy through the agent identity

Start	Target	Priority
P2, a low privilege user	**Systems the agent can call**	**High**
Estimate		
1.5 days		

Goal. Get the agent to do something in a downstream system that the user could not do directly.

Why this can work

AP06 is about what an index returns. This path is about the systems behind the agent's tools: HR, ticketing, CRM, finance. When the agent calls them with its own identity, the downstream system checks the agent's rights, not the user's. To serve every user, agents are usually granted the union of every user's needs, so each user inherits everything the agent can do.

The fix is the on behalf of user flow, or an explicit authorisation check in the tool, but both are extra work and often skipped. The test needs white box access to the agent's tool definitions and role assignments to choose targets well.

Chain

Step	Attacker action	Control expected to stop it	Tester records
1	Compare the agent identity's role assignments and API permissions with the test user's	Agent identity scoped to its charter	Rights the agent has that the user lacks
2	Ask the agent to act on a resource only the agent can reach: another team's ticket queue, a salary record	On behalf of user flow, or a per call authorisation check in the tool	Whether the action succeeds, and whose identity the downstream log shows
3	Check the downstream audit trail	Logs show both agent and user	Whether the real requester can be identified

Pass looks like

Every downstream action is limited to what the requesting user could do, and the audit trail names both the agent and the user.

Maps to

E13 I05 LLM06 ATT&CK T1078.004

AP08 Write actions without approval, loops and spend limits

Start	Target	Priority
P2 directly, or P3 by injection	**Write tools, run limits, APIM token limits**	**High**
Estimate		
1.5 days		

Goal. Get the agent to change, send or delete something without a person approving it, keep running without end, or spend beyond its limit.

Why this can work

Approval is configured tool by tool. Teams put approval on the obvious tool ("send email") and miss the one that matters ("update claim status"). Where approval is implemented in the application rather than the platform, it can sometimes be satisfied by text: an injected line saying "the user has already approved this" is just more context to the model.

Loops come from instructions such as "keep checking until it changes", with no maximum steps or run timeout. Spend limits are the backstop for both, and they are worth verifying because a limit set in the wrong product or keyed on the wrong counter does nothing.

Chain

Step	Attacker action	Control expected to stop it	Tester records
1	Ask plainly for each write action the agent's tools allow	Approval required for every write, send, pay, delete and access change tool	Tools that executed without approval
2	Bury the write as step five of a longer, harmless task	Approval enforced per call, regardless of context	Result
3	Inject text claiming approval was already given	Approval enforced by the platform or client, not decided by the model	Result
4	Instruct a polling or self repeating task	Maximum steps and run timeout	Steps and tokens before the run stopped
5	Against a test APIM product with a deliberately low token limit, send a short burst just over it	APIM token limit per consumer	Request at which 429s began, and whether other consumers were unaffected

Pass looks like

No write executes without a person approving it, runs stop at their limit, and the token limit triggers at the configured value for that consumer only.

Rules of engagement. Denial of service testing, including simulating it, is prohibited. Step 5 uses a lowered limit and a small burst against a test product, to show the limit is wired correctly. Nothing attempts to exhaust deployment capacity. Write actions run against test records only.

Maps to

I03 I04 E05 LLM06 LLM10 AML.T0034 ATT&CK T1496 ATT&CK T1565

AP09 Tool supply chain and request forgery from the agent subnet

Start	Target	Priority
P4 for tool registration, P3 for request forgery	**MCP servers, OpenAPI tools, internal endpoints**	**High**
Estimate		
2 days		

Goal. Introduce a malicious tool, alter a trusted one, or use a tool that fetches URLs to reach internal systems from inside the client's network.

Why this can work

In the reference design the agent runtime is injected into the client's own subnet, so its requests come from inside the VNet with a line of sight to private endpoints and, through ExpressRoute, possibly to on premises systems. Any tool that fetches a URL or host chosen by the model, such as a "read web page" MCP tool or an OpenAPI tool with a URL parameter, is a server side request forgery primitive whose source is the client's network.

MCP adds a supply chain angle. The model reads each tool's description to decide how to use it, so a description can carry instructions (tool poisoning), and a server can change its description or behaviour after it was reviewed. Allow listing by exact server URL and pinning the tool schema are the controls; both need verifying, not assuming.

Chain

Step	Attacker action	Control expected to stop it	Tester records
1	List every tool that takes a URL, host, path or query as input	Tools take identifiers, not URLs	Candidate tools
2	Through such a tool, request private endpoint FQDNs, other spoke addresses, on premises hosts	Input validation in the tool; NSGs on the agent subnet; firewall rules from the agent subnet	Internal targets reached and their responses
3	In the development project, register a tester owned MCP server; after review, change its description to include instructions	Exact URL allow list; schema pinned and re-reviewed on change	Whether the changed description was accepted and acted on

Step	Attacker action	Control expected to stop it	Tester records
4	Check whether a tool registered in development can be used by a production agent	Project separation; production agents deployed only by the pipeline	Result

Pass looks like

No tool reaches an internal address outside its purpose, a changed tool schema is rejected or flagged, and nothing registered in development reaches production.

Rules of engagement. The agent runtime is Microsoft managed. If a request from the runtime reaches a Microsoft internal service, such as an instance metadata endpoint of the platform, stop at proof of concept and report to MSRC. Enumeration, secret dumping or pivoting on Microsoft assets is prohibited.

Maps to

`E08` `I03` `LLM03` `LLM06` `AML.T0053` `AML.T0010` `ATT&CK T1195` `ATT&CK T1190`

AP10 Egress bypass through allowed destinations

Start	Target	Priority
P3 by injection, or P4	**Firewall rules, DNS, code interpreter**	**High**
Estimate		
1 day, plus optional boundary testing		

Goal. Move canary data out of the agent environment through destinations the firewall already allows.

Why this can work

Agent subnets need to reach Entra ID and Azure services, and firewall rules for that are often written with wildcards such as `*.blob.core.windows.net` or broad service tags. Those match every storage account in Azure, including one the attacker created in their own subscription. "Private egress only to Azure" can still mean "egress to the attacker's Azure".

DNS is the other common gap. If the agent subnet can resolve arbitrary external names, data can leave encoded in subdomain lookups to a domain whose name server the attacker controls, even when no HTTP egress is allowed. Code interpreter runs in a Microsoft managed sandbox; whether it has outbound access in the client's configuration is worth checking rather than assuming.

Chain

Step	Attacker action	Control expected to stop it	Tester records
1	Review firewall rules and service tags applied to the agent subnet	Exact FQDNs, no wildcards over shared cloud domains	Wildcards and broad tags
2	From the agent context, write a canary to a tester owned storage account on an allowed wildcard	Exact FQDN rules; private endpoints for the client's own accounts only	Whether the write succeeded
3	Trigger lookups of tester controlled subdomains carrying a canary	DNS filtering on the resolver or firewall DNS proxy	Queries received at the tester name server

Step	Attacker action	Control expected to stop it	Tester records
4	From code interpreter, attempt DNS and HTTP to a tester endpoint	No outbound access from the sandbox	Result

Pass looks like

No canary reaches any tester owned destination by HTTP, storage write or DNS.

Rules of engagement. Attempting to break out of the code interpreter sandbox is permitted, but on success the tester stops immediately and reports to MSRC; no further action on Microsoft infrastructure. Agree in advance whether boundary testing is in scope, as it adds time without testing the client's own configuration.

Maps to

`E03` `I12` `I13` `LLM06` `ATT&CK T1567.002` `ATT&CK T1048` `ATT&CK T1071.004`

AP11 Developer privilege escalation and guardrail tampering

Start	Target	Priority
P4, Foundry User on a development project	**Control plane, other projects, connection secrets**	**High**
Estimate **2 days**		

Goal. From an ordinary developer account, reach the control plane: create deployments, weaken content filters, read connection secrets, reach other projects, or switch on public access or keys.

Why this can work

Foundry's roles separate control plane from data plane well on paper, but organisations hand out Contributor or Foundry Owner to unblock a team and never take it back. One account with those rights can detach a filter policy from a deployment, which disables content safety for every app on it in a single change.

Connections are the subtler route. Connections created at the Foundry resource level can be shared across projects, and some connection types still hold secrets that an identity with the right action can list. Azure Policy is meant to be the backstop for public access and keys, so the test also confirms the policy is in Deny, at the right scope, without an exemption covering this resource.

Chain

Step	Attacker action	Control expected to stop it	Tester records
1	Enumerate own role assignments and every resource visible through Resource Graph	Least privilege, project scoped assignments	Effective rights
2	Create a model deployment	Deployment write not in Foundry User	Result
3	Change or detach a filter policy	Filter policy writes restricted to the platform team	Result
4	List connections and try to retrieve their secrets	Entra ID connections; secret listing not granted	Any secret returned
5	Read agents, threads or files in another project	Project scoped role assignments	Result

Step	Attacker action	Control expected to stop it	Tester records
6	Enable local auth or public network access	Policy Deny, no exemption	Result and the policy that denied it
7	Create a role assignment	No role assignment rights	Result

Pass looks like

Every step is denied, each denial appears in the Activity Log, and steps 3 and 6 raise alerts.

Maps to

`I01` `I05` `I07` `I11` `I12` `ATT&CK T1098` `ATT&CK T1078.004` `ATT&CK T1580`

AP12 Unapproved training and model deployment

Start	Target	Priority
P4	**Fine tuning, model policies, data collection**	**Medium**
Estimate		
1 day		

Goal. Fine tune a model on uploaded data, deploy it or another unapproved model, or switch on features that keep user data for later reuse.

Why this can work

Default data plane roles include fine tuning and dataset upload. Personal data trained into weights cannot be selectively removed, so this is a data protection failure even if no attacker is involved, which is why the priority is Medium: the harm is regulatory and slow rather than an immediate breach.

The approved models policy matches asset IDs by prefix. An entry without a trailing slash approves every model whose name starts the same way, and a publisher entry approves every future model from that publisher. Testing what the policy actually allows is the only way to know.

Chain

Step	Attacker action	Control expected to stop it	Tester records
1	Upload a small training file and start a fine tuning job	Builder role without fine tuning actions; training only in the approved project	Result
2	Deploy a fine tuned model outside the training scope	Custom deny on fine tuned deployments	Result
3	Deploy a model not on the list, and one whose name extends an approved name	Approved models policy with exact asset IDs	Which deployments the policy allowed
4	Deploy a preview model, and use the Developer deployment type	Eligibility policy; Developer type denied outside training	Result
5	Enable stored completions or agent memory	Off unless approved	Result

Pass looks like

Every step is denied by role or policy, with the policy named in the denial, and each attempt is visible to the SOC.

Cost. Fine tuning jobs and deployments cost money. Agree a cost cap and the smallest job size, and delete every artefact afterwards.

Maps to

`I01` `I02` `LLM03` `LLM04` `AML.T0010` `AML.T0020`

AP13 Pipeline identity abuse

Start	Target	Priority
P5, repository write access	**Federated credentials, self hosted runner, infrastructure**	**High**
Estimate		
1.5 days		

Goal. Use the pipeline's identity to change infrastructure without review, for example to switch off a guardrail or open network access.

Why this can work

The pipeline usually holds the highest standing rights in the estate, because it deploys everything. Workload identity federation removes the secret but moves the risk into the federated credential's subject: a subject that matches any branch, any pull request or any environment lets code that never passed review obtain a token.

The self hosted runner inside the spoke is both a privileged identity holder and a network foothold. If it is not ephemeral, tokens and files from earlier runs can remain on disk. If the repository is public and the runner accepts jobs from forks, an outsider can run code inside the client's network.

Chain

Step	Attacker action	Control expected to stop it	Tester records
1	Review federated credential subjects and the roles of the pipeline identity	Subjects bound to a protected branch and environment; least privilege	Subjects and rights
2	Push a workflow to an unprotected branch that requests a token	Subject does not match the branch	Whether a token was issued
3	If the repository is public, open a pull request from a fork targeting the self hosted runner	Fork jobs never run on self hosted runners	Result
4	With any token obtained, attempt a guardrail weakening change	Environment approval; Azure Policy Deny as the backstop	Result
5	Inspect the runner for leftover tokens and files	Ephemeral runners	Findings

Pass looks like

Only reviewed code on a protected branch and approved environment gets a token, and policy denies guardrail weakening even when it does.

Rules of engagement. Prefer a sandbox copy of the repository and subscription. If the real pipeline is used, agree a rollback plan and a change window.

Maps to

`I01` `I07` `I12` `E11` `ATT&CK T1199` `ATT&CK T1528` `ATT&CK T1195.002`

AP14 Network foothold to private endpoints

Start	Target	Priority
P6, a host in another network location	**Private endpoints, DNS**	**Medium**
Estimate		
1.5 days		

Goal. From a compromised host elsewhere, reach Foundry and its stores, and establish what an attacker with network reach can do.

Why this can work

A private endpoint is reachable from anything routed to its subnet: peered spokes, on premises over ExpressRoute, and other clouds over the interconnect. NSGs on a private endpoint subnet only take effect when network policies are enabled on it, which is off by default on older subnets. Private often means "reachable from the whole corporate network".

Entra ID still applies, which is why the priority is Medium: reach alone is not access. The finding that matters is reach combined with any valid token, for example a compromised user who holds Foundry User for an unrelated project.

Chain

Step	Attacker action	Control expected to stop it	Tester records
1	From each foothold, resolve and connect to each private endpoint	Firewall and NSGs limit sources to approved subnets	Reachability matrix
2	Call each endpoint with the foothold user's token	RBAC	Result
3	Check whether the foothold can be made to resolve a public address for a private name	Public access disabled, so a wrong resolution fails closed	Result

Pass looks like

Only approved sources reach the private endpoints, and no foothold identity gets data.

Maps to

`E14` `I12` `ATT&CK T1046` `ATT&CK T1557`

AP15 Detection and response validation

Start	Target	Priority
Runs alongside every other path	**Logs, alerts, SOC process, kill switch**	**High**
Estimate		
2 days		

Goal. For every other test, establish whether it was recorded, whether an alert fired, and how long the response took. Then rehearse the agent kill switch.

Why this matters

A control that blocks silently looks exactly like a control that never ran, and a detection rule that has never fired may not work. Running detection checks alongside the attack paths, as a purple team exercise, turns the test into evidence: each blocked attempt shows up as a log line, each successful one as an alert, and any gap between them is a finding.

Logging itself is a target. An attacker who deletes a diagnostic setting blinds every rule that depends on it, and the only rule that catches that is one that looks for absence.

Chain

Step	Action	Control under test	Tester records
1	Agree a detection matrix: for each path, which log and which alert should show it	Not applicable	Matrix
2	Record the time of each test action	Logs and alerts	Time to log, time to alert, time to triage
3	On a test agent, run the kill switch steps in order	Kill switch runbook	Time for each step to take effect
4	With approval, delete the diagnostic setting on a test resource	Delete alert and absence rule	Whether and when each fired

Pass looks like

Every test action appears in the logs with an identity, the agreed alerts fire within the agreed time, the kill switch stops the test agent within 15 minutes, and the absence rule fires when logs stop.

Maps to

I04 | I08 | ATT&CK T1562.008

Out of scope, and why

Excluded	Reason
Microsoft's model hosting, Agent Service infrastructure and the Prompt Shields models as a product	Microsoft's assets, not the client's. Boundary tests stop at proof of concept and go to MSRC.
Load, stress or capacity exhaustion of model deployments	Denial of service testing in any form is prohibited. Limits are verified at low volume instead (AP08).
Any tenant, subscription or resource the client does not own	Prohibited, and a legal risk for the tester.
Use of real leaked credentials	Only credentials the client owns and plants for the test may be used.
Poisoning a production model through training data	Irreversible for the client. AP12 tests whether training can start without approval, which is the control that matters.
Social engineering of Microsoft staff	Prohibited. Social engineering of client staff needs its own scope if wanted.

What to ask the client before quoting

Effort depends far more on the number of agents, tools and grounding sources than on the size of the Azure estate. These answers set the quote.

1. An agent inventory: each agent, its model deployment, its tools, its grounding sources, and who can write to each source.
2. The architecture as built: network diagram, firewall rules for the agent subnet, private endpoint list, APIM configuration.
3. Role assignments for the AI subscriptions, and the federated credential subjects of the deployment pipeline.
4. Which starting positions are in scope, and the test accounts for each (two end users in different groups as a minimum).
5. A canary data set: seeded records and documents the tester may try to move, with distinct access lists.
6. Tester owned endpoints for listeners, storage and DNS, agreed in writing.
7. Acceptable attack success rates for AI tests, per harm category, agreed before testing.

8. Whether the SOC is told (purple team) or not (detection blind test), and the emergency contacts on both sides.
9. A cost cap for tokens, fine tuning and any deployments the tests create.
10. The test window, any change freeze, and agreement on reporting any Microsoft platform issue to MSRC.

Effort and reporting

For one Foundry resource, one production project and up to three agents, the paths above total 24 tester days, and 31 with setup, reporting and a retest. These are planning estimates for an experienced cloud and AI tester, not a quote; several paths can run in parallel.

Item	Days	Scales with
Scoping and setup: accounts, canaries, listeners, detection matrix	2	Number of starting positions
Platform paths: AP01, AP02, AP11, AP12, AP13, AP14	8	Number of Foundry resources and subscriptions
Agent paths: AP03 to AP10	14	Number of agents and writable read surfaces; add about 0.5 day per extra agent for AP04
Detection and response: AP15	2	Whether the SOC is involved live
Reporting	3	Number of findings
Retest after fixes	2	Number of findings
Total	**31**	

What the report should contain

- Each finding mapped to its attack path, threat ID, STRIDE category and the control that failed, so remediation lands on the right owner.
- For each agent, how many defensive layers remained when a path partly succeeded. One layer left is a different risk from four.
- Attack success rates for AI tests, per category and per deployment, against the agreed threshold.
- The detection matrix as observed: logged, alerted, triaged, with times.
- Kill switch timings from AP15.
- Any issue reported to MSRC, listed without detail until Microsoft confirms a fix.

Sources: Microsoft Learn, Penetration testing; Microsoft Cloud Unified Penetration Testing Rules of Engagement; Set up private networking for Foundry Agent Service;

Built-in policies for model deployment. OWASP codes refer to the OWASP Top 10 for LLM Applications 2025; AML codes to MITRE ATLAS; T codes to MITRE ATT&CK.

Part of the Foundry Field Guide by Saleem Yousaf. Independent and not endorsed by Microsoft. No cookies, analytics or tracking.