SF² Core Framework

Software Factory Security Framework

Julie Davila

Contents

1. Foundation 3
1.1 Foundation: Software Factory Definition 3
1.2 Engaging the Atelier Critique 6
2. Universal Security Conditions 8
2.1 Universal Security Conditions 8
2.2 Supply Chain 12
2.3 Third-Party 14
2.4 Process 16
2.5 Runtime 17
2.6 Adaptive Capacity 18
3. Strategic Positioning 20
3.1 Two-Axis Positioning Model 20
3.2 Four Strategic Positions 28
3.3 Strategic Movement Paths 34
4. Investment Portfolio 44
4.1 Investment Portfolio Framework 44
4.2 BAU vs Scaling Investments 52
4.3 Platform Effects 62
4.4 Investment Evaluation Criteria 70

1. Foundation

1.1 Foundation: Software Factory Definition

1.1.1 The Universal Challenge

Whether you're leading security for a three-person startup or a multinational corporation, you face the same fundamental question: **How do you maintain security accountability for code-based systems that deliver value to end users?**

This responsibility spans your entire value delivery chain, from internal development through third-party dependencies to production operations.

1.1.2 What is a Software Factory?

Definition

A **software factory** is whoever bears operational responsibility for deploying, maintaining, and evolving code-based systems that deliver value to end users, including systematic risk stewardship across all components in their value delivery chain, whether directly controlled or third-party.

This definition encompasses:

- **Scale Agnostic**: From single-developer startups to enterprise organizations
- **Technology Agnostic**: Any tech stack, deployment model, or infrastructure approach
- **Responsibility Focused**: Emphasis on operational accountability rather than just code ownership
- **Third-Party Inclusive**: Recognition that modern software depends extensively on external components

1.1.3 Key Characteristics

Operational Responsibility

Software factories are accountable for how code reaches end users and impacts their experience, regardless of whether every component is built in-house.

Value Delivery Chain

The entire pipeline from code creation through production deployment and ongoing operations.

Risk Stewardship

Ongoing responsibility for understanding, assessing, and responding to security risks across the complete software stack.

Systematic Processes

Repeatable, improvable approaches to software creation and deployment rather than ad-hoc development.

1.1.4 Why This Definition Matters

Understanding your organization as a software factory helps clarify:

1. **Scope of Responsibility**: You're accountable for security across the entire value delivery chain, beyond the code you write
2. **Third-Party Dependencies**: External components are part of your security responsibility

3. **Operational Focus**: Security accountability extends through production operations
4. **Universal Applicability**: The same framework applies regardless of organization size or technology choices

1.1.5 How This Framework Complements Existing Security Standards

SF² addresses strategic questions that existing frameworks don't answer:

- **Resource Allocation Strategy**: How do you sustainably invest in security as your organization scales?
- **Contextual Implementation**: How do you adapt security approaches to your specific organizational reality?
- **Business Integration**: How do you align security investments with business outcomes and competitive advantage?

Framework Relationships

Framework	Primary Focus	SF² Relationship	When to Use Together
NIST SSDF	Secure development lifecycle practices	SF² addresses sustainable resourcing of SSDF practices at scale	Use SSDF for development security practices, SF² for sustainable implementation strategy
OWASP SAMM	Security practice maturity progression	SF² contextualizes SAMM implementation based on organizational readiness	Implementation speed and scope vary by organizational complexity and readiness level
BSIMM	Security activity measurement and benchmarking	SF² determines investment priorities for BSIMM activities based on organizational positioning	Use SF² assessment to guide BSIMM implementation scope and sequencing
OWASP ASVS	Security verification requirements	SF² helps sequence ASVS implementation within scaling investment strategy	Use SF² to determine risk-based ASVS subset vs. comprehensive implementation

The four frameworks in the table are today's instance of the baseline, not the baseline itself. SF² overlays whatever the prevailing practice baseline is, and right now that is a set of human-pace maturity models: SSDF, SAMM, BSIMM, ASVS. That baseline is already broadening. As more software gets built and shipped by machines, a new layer is emerging alongside the maturity models: attestation and capability standards. Two examples point the way. Verifiable provenance for what was built is the direction SLSA (https://slsa.dev/) points, and scoped identity for what an agent may do is what SPIFFE (https://spiffe.io) provides. The roster of the baseline will grow; SF²'s job does not. It sequences the investment, decides what to fund first for your context, and aims it at the thing both baselines are ultimately trying to protect: a capability boundary that bounds what any part of the system can do. That boundary is the floor. When the baseline grows a new wing, SF² sequences it the way it sequenced the old. The durable commitment is to that floor, set out in Boundary Enforcement, not to the four names that express it today.

1.1.6 Examples of Software Factories

To illustrate the universal nature of this definition:

Startup Software Factory

- **Team**: 5 developers
- **Product**: SaaS application
- **Value Chain**: Cloud infrastructure (AWS), development tools (GitHub), monitoring (Datadog), dozens of open-source dependencies

- **Security Responsibility**: Entire stack, despite building ~5% of code themselves

Enterprise Software Factory

- **Team**: 500+ developers across multiple teams
- **Products**: Multiple applications and services
- **Value Chain**: Multi-cloud infrastructure, extensive third-party integrations, internal platform services, hundreds of dependencies
- **Security Responsibility**: Complex ecosystem with multiple ownership boundaries but unified accountability to end users

Platform Software Factory

- **Team**: 50 platform engineers
- **Product**: Internal developer platform
- **Value Chain**: Kubernetes infrastructure, CI/CD pipelines, security tooling, compliance automation
- **Security Responsibility**: Enabling other teams while maintaining platform security posture

> **Common Thread**
>
> In each case, security responsibility extends far beyond code directly written by the organization. The software factory definition emphasizes this operational accountability across the complete value delivery chain.

This raises a gap the rest of the framework has to close. You are accountable for far more than you can personally read, and as generation outpaces comprehension the gap widens: the volume of code, dependencies, and agent actions you answer for grows faster than anyone's capacity to inspect it. Accountability does not shrink to match. You discharge it by bounding what the system was ever able to do, not by comprehending everything that happened. Comprehending everything is no longer possible at scale. That is what boundary enforcement and the three-layer model provide: a way to answer for a system you cannot fully read. The same logic governs what you delegate rather than build: as Third-Party stewardship puts it, a contract can move a capped slice of the cost, never the responsibility.

1.1.7 Next Steps

Now that you understand what constitutes a software factory, the next section explores the five universal security responsibilities that every software factory must address.

1.2 Engaging the Atelier Critique

Foundation · Atelier engagement

This chapter extends: Software Factory Definition. **Scope:** the steelmanned counter-framing (Sankar's artist colony) and why SF² + Coadaptive Security absorbs both factory and atelier views as the synthesis.

Not everyone who builds software believes they work in a factory, and some of the most influential builders reject the metaphor outright. Shyam Sankar, Palantir's CTO, calls software production an artist colony: position and portfolio decoupled, the role itself an arbitrary construct, the artist and the work the only durable units. He is describing something true. A framework called Software Factory Security owes that view a real answer rather than a dismissal. The answer is that the factory framing and the atelier framing are both correct at different layers of the same system, and SF² plus Coadaptive Security is what holds them together.

1.2.1 The atelier critique, steelmanned

Sankar's claim deserves its strongest form. In *Position and Portfolio* (https://www.shyamsankar.com/p/position-and-portfolio) he argues that titles and org charts are scaffolding the work does not need: "There is only the artist and the work." Roles are fluid, ownership stays with the person who made the thing, and opportunity gets seized without waiting for a box on a chart to authorize it. He put it more plainly to the *Financial Times* in 2021: "We at Palantir, we're an artist colony, extraordinarily and exquisitely flat."

This is not generational posturing, and it names something real. A generation of engineers reads software as craft, where the person and the artifact are inseparable and the value lives in judgment rather than in process. Anyone who has watched one strong engineer outproduce a carefully managed team of ten has felt the pull of the argument. The atelier framing is a real cultural force, and a framework that pretends otherwise gets ignored by exactly the people it most needs to reach.

Primary citation:

- Sankar, Shyam. *Position and Portfolio.* shyamsankar.com, 12 November 2023. https://www.shyamsankar.com/p/position-and-portfolio (https://www.shyamsankar.com/p/position-and-portfolio)
- Sankar, Shyam. *Financial Times* interview, 12 August 2021: *"We at Palantir, we're an artist colony, extraordinarily and exquisitely flat."*

1.2.2 Why "Software Factory" is the right term anyway

The factory framing survives the critique because it answers a different question. The atelier describes how creative work feels and how talent arranges itself. The factory describes who is accountable when ten thousand deployments a day have to ship safely. Those are not competing claims about the same thing. The term names operational responsibility for delivery at scale, and that responsibility does not dissolve because the people doing the work think of themselves as artists. Someone still owns the blast radius when the artist's brilliant, unreviewed change reaches production.

The term is also not a coinage invented to make this argument, and not a jab at Palantir. It is industrial-lineage vocabulary with a fifty-year paper trail.

Term provenance

The history predates the current debate by decades:

- **Bob Bemer, 1968** (https://www.computerhistory.org/collections/catalog/102724781): the GE paper often credited as the earliest "software factory" proposal.
- **Cusumano, Michael, 1991**: *Japan's Software Factories: A Challenge to U.S. Management.* (https://global.oup.com/academic/product/japans-software-factories-9780195062168) Oxford University Press.
- **Greenfield, Jack and Keith Short, 2004**: *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools.* (https://archive.org/details/softwarefactorie0000gree) Wiley.
- **U.S. Department of Defense**: 20+ accredited Software Factories operating today (Platform One (https://software.af.mil/), Kessel Run, Kobayashi Maru, Black Pearl, others).

The Defense Department runs more than twenty accredited software factories right now. The word carries operational and regulatory weight in the one ecosystem where getting delivery wrong gets people killed. It is the right word for what SF² governs.

1.2.3 SF² + Coadaptive as the synthesis

Both framings are load-bearing, and the synthesis is naming the layer where each one holds.

The factory framing is right at the layer of operational accountability. Foundation and the Universal Security Conditions live here: who owns the supply chain, who answers for what runs in production, how delivery stays safe as it scales. The atelier framing is right at the layer of the creative act and how roles get arranged inside a team. SF² never claimed that layer and does not want it. How you organize your artists is your business.

Coadaptive Security extends the picture to a third layer neither metaphor anticipated: the unit of operation in the AI era. That unit is most often a person working with agents, and it resembles a paired-intelligence cell more than a factory worker or a lone craftsman. Chapter 03, The Unit of Defense, takes that up. The synthesis does not ask anyone to pick a team. It names the layer at which each framing earns its keep.

1.2.4 The AI-era production-model question

Sankar's framing surfaces a harder question than the one it answers, and the synthesis has to sit with it honestly. If the unit of production is shifting from a team of humans to a human working with agents, neither metaphor maps. A factory is a human assembly line. An atelier is a lone craftsman with tools. A paired-intelligence cell is neither. The tools in this case reason, act, and occasionally get things wrong on their own initiative.

Foundation does not resolve that question. It names it and hands it forward to Coadaptive Security Chapter 03, where the unit of operation and the property that defends it get worked out.

1.2.5 See also

- Software Factory Definition: the base definition this chapter engages
- Coadaptive Security · 03 The Unit of Defense: where the AI-era production-model question is taken up
- Coadaptive Security · Overview: the AI-era layer that absorbs the production-model shift

2. Universal Security Conditions

2.1 Universal Security Conditions

2.1.1 Conditions, not controls

A control is something you check off. It passes or it fails, it lives as a line in a spreadsheet, and half the energy around it goes into arguing the spirit of the control rather than the state of the system. A condition is a different kind of object. It is something you cultivate and keep alive, and it has no passing grade. The security posture of a software factory is a set of conditions you tend, not a register of controls that cleared an audit. Like anything living, they start to degrade the moment you stop tending them.

Your strategic position tells you *how* to tend them. A Craft shop and a Lean enterprise cultivate the same conditions with very different tools, budgets, and timelines. The conditions themselves do not move with size, stack, or industry. They are what every software factory has to keep alive regardless of where it sits on the map, which is why this section comes before the positioning work and not after it.

There are four conditions you can hand to a team, and one you cannot.

Four conditions you can name, staff, and put on an org chart, plus Adaptive Capacity running underneath all four. You assess the fifth; you cannot hand it to a team.

2.1.2 The four you can staff

Condition	The question it asks	The lever
Supply Chain (#1)	Do I know what is *inside* what I shipped?	Comprehension: SBOM, provenance, signing, pinning
Third-Party	Do I know what I *handed off*, and who holds the bag when it fails?	Containment: blast-radius limits, shared-responsibility clarity, failover
Process	Does the way we build *produce* security, or bolt it on afterward?	Feedback: security arriving while the work happens, not in a review at the end
Runtime	Can we *sense* what production is doing right now and respond before a user does?	Observability: detection, response, recovery

These four map to functions you can name, staff, and put on an org chart. That is deliberate. A condition you cannot assign to anyone is a condition nobody tends.

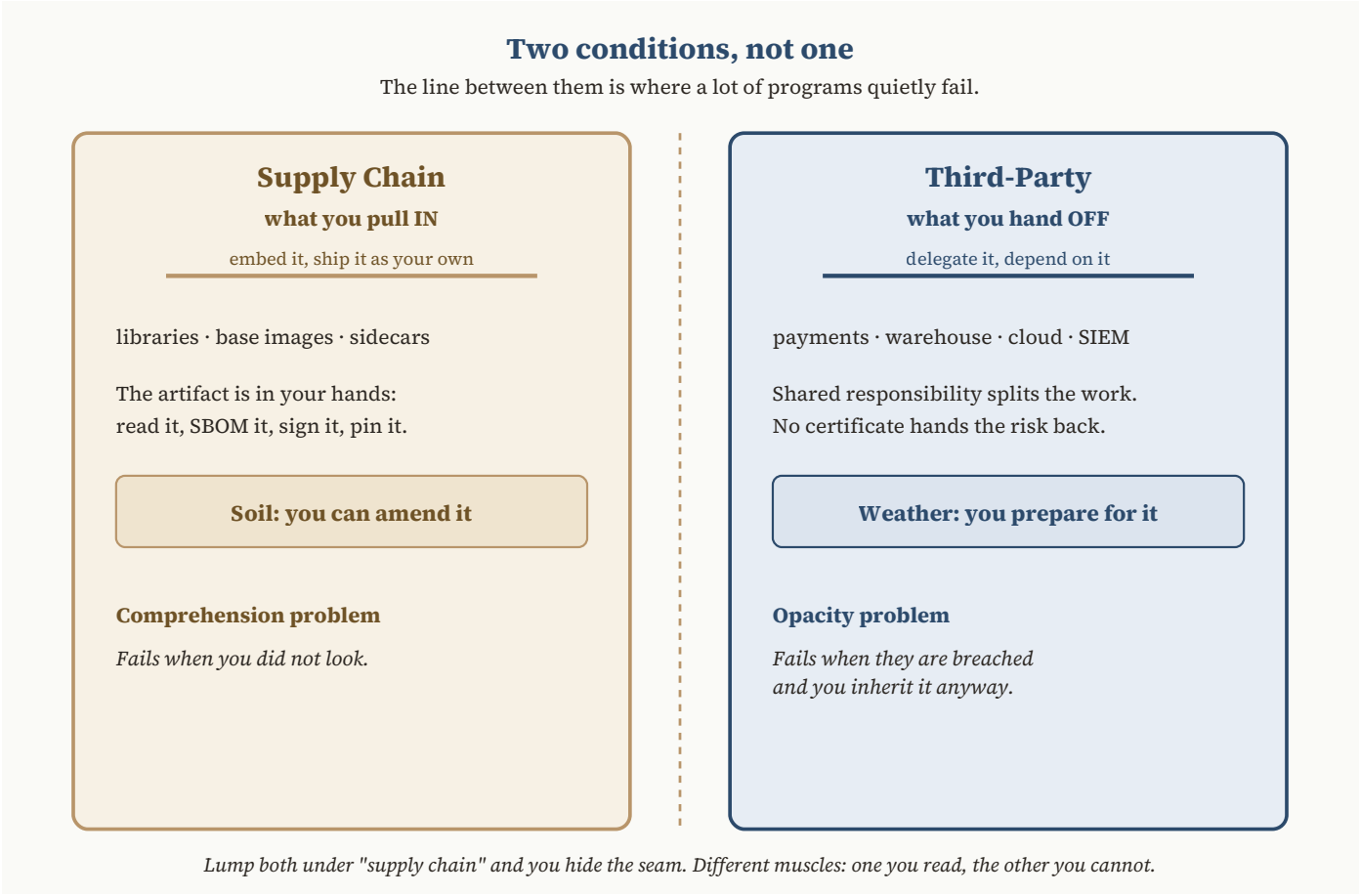
Supply Chain and Third-Party are not the same condition

They look like one thing. They are not, and the line between them is where a lot of programs quietly fail.

Supply Chain is what you pull *in*: third-party libraries, base images, sidecars, the code you embed and then ship as if it were your own. Once you embed it, the liability is yours. You can also do something about it, because the artifact is in your hands. You can read it, generate an SBOM, sign it, pin it, rebuild it. Supply Chain is soil: you can amend it. The way it fails is that you did not look.

Third-Party is what you hand *off*: the payments processor, the data warehouse, the identity-verification provider, the cloud, the SIEM. Some of these are infrastructure; others are core to how your product delivers value at all. You delegate the function, and a shared-responsibility model splits the work. A contract can shift the legal liability to them; it cannot shift the responsibility, and no certificate hands it back. Compliance is a market-access key, not a security proof: a vendor earns FedRAMP to unlock the federal market, not to become safe to depend on, and its attestation retires none of your risk. You still have to do your part and trust they are doing theirs. You cannot directly inspect or observe what they run. Third-Party is weather: you cannot change it, you can only prepare for it. The way it fails is that they got breached and you inherited it anyway.

Lumping both under "supply chain" because the word has stretched to cover everything external is the move that hides the seam. The two conditions take different muscles. One is a comprehension problem: you embedded something and never read it. The other is an opacity problem: you delegated something you cannot directly inspect, and no contract or certificate changes that. Keep them apart on the page so they stay apart in the work.



Lump both under "supply chain" and you hide the seam. Different muscles: one you read, the other you cannot.

Supply Chain is what you pull in and can amend, a comprehension problem. Third-Party is what you hand off and can only prepare for, an opacity problem. Lump them together and you hide the seam.

Supply Chain is still the loudest

Of the four, Supply Chain has been the one to watch for years, roughly since adversaries moved discovery to automation at internet scale and started finding vulnerable dependencies faster than defenders could inventory them. This is about tempo, not a ranking of which matters more. The supply-chain condition degrades faster and gets exploited sooner than the rest, so it earns first call on attention and budget. Treat it as the default #1 and argue yourself *down* from there if your context warrants.

2.1.3 The one you cannot staff: Adaptive Capacity

The fifth condition does not get a team, and that is the point.

Adaptive Capacity is whether the system as a whole can absorb a surprise it was not designed for and keep working. It is the old Continuous Learning idea, finally named for what it actually is. The other four conditions each map to a function you can put on an org chart. This one does not. It is closer to the resilience of an ecosystem than to anything you could install or assign. The capacity of a living system to take a shock, a drought or a new predator, and reorganize without collapsing is spread across the whole web rather than held in any single species or place. It is never finished, because the system and the things stressing it keep changing against each other. You assess whether the system *has* this capacity. You cannot hand it to a team.

It runs across the other four rather than beside them. A healthy Adaptive Capacity shows up as blameless post-incident review that changes something, as feedback loops that shorten, as the organization sensing a shift in the threat landscape and adjusting before it gets hit rather than after. When it is missing, the other four can each look fine on a maturity chart while the system stays brittle, because nothing is teaching it to bend.

This is the condition that carries the framework's resilience thinking, and it is the seam where this base framework meets the Coadaptive Security layer. The idea comes from ecology. A system survives shocks when it can reorganize as new ones arrive, and that capacity lives in the whole web rather than in any single part. Security works the same way. Certify a system once and trust it to hold, and it falls behind, because the things trying to break it keep changing while it stands still. Adaptive Capacity measures whether the whole system is still adapting faster than its adversaries. That is the question worth asking.

2.1.4 How the conditions hold each other up

The conditions are coupled, not a checklist, and the coupling is where leverage hides:

- A weak **Supply Chain** condition rarely announces itself in Supply Chain. It surfaces at **Runtime**, as the incident you trace back to a dependency you never inventoried.
- **Process** is where the other conditions either get cultivated or get skipped. A build pipeline that produces provenance is tending Supply Chain for free.
- **Third-Party** failures are bounded by **Runtime** containment. The vendor breach you survive is the one whose blast radius you limited in advance.
- **Adaptive Capacity** is how all four improve at all. Without it, you are just repainting the same four walls on a fixed cadence.

Invest where a single move strengthens more than one condition at once. Those are the moves worth sequencing first.

2.1.5 Implementation varies, the conditions don't

The conditions are universal. How you cultivate them is not.

A three-person Craft shop and a five-thousand-person Lean enterprise both have to tend Supply Chain, but one does it with a single well-chosen managed scanner and the other with a platform team and a paved road. The Strategic Positioning section is how you decide which version of "tending" your organization can actually sustain. Read the conditions here as the *what*. Read positioning as the *how*.

> **Naming note (v0.5 v0.6)**
>
> In v0.5 this section was "Universal Risk Stewardship Responsibilities," organized as five areas you steward. v0.6 reframes them as conditions you cultivate, splits the old Supply Chain area along the embedded/delegated line, and recasts Continuous Learning as the cross-cutting Adaptive Capacity condition. The migration crosswalk maps every old name to its v0.6 home.

2.1.6 Next Steps

Start with the condition that degrades fastest:

2.2 Supply Chain

Supply Chain is the condition that asks one question: do you know what is inside what you shipped? Every modern build pulls in code you did not write, the third-party libraries, base images, sidecars, and a transitive dependency four levels down that you have never once opened. Once you embed it, you ship it as if it were your own, and the liability is yours from that moment on.

The question now reaches past code. A modern build also ships model weights you did not train, datasets you did not assemble, and the pieces an agent pulls in to do its work: the skills it loads, the tools it calls, the MCP servers it connects to. Each one you embed and then ship as your own, the same as any library, and the liability arrives with it. These resist inventory harder than code does. An agent's tools tend to self-install, self-update, and connect at runtime, often outside whatever software intake process you thought you had. And a model weight arrives as an opaque blob you cannot read or rebuild from source, whose origins you take on trust. The surface widened faster than the practice did. The discipline does not change: know what is inside, and refuse to ship what you cannot account for.

Some of what you embed does not sit still. An agent's tools and MCP servers reach out at runtime to dependencies of their own, pulled in past whatever intake you thought you had, so the thing you shipped is operating sub-dependencies you never reviewed. That is the operator beneath the operator seen from the supply side: a delegated dependency, embedded.

This is the condition you can do the most about, because the artifact is in your hands. You can read it, generate an SBOM, sign it, pin it, rebuild it from source. Supply Chain is soil. You can amend it. The way it fails is almost always the same: you did not look.

2.2.1 Why it is the loudest of the four

Supply Chain has been the one to watch for years, roughly since adversaries moved discovery to automation at internet scale and began finding vulnerable dependencies faster than defenders could inventory them. The asymmetry is the whole problem. An attacker runs continuous, internet-wide scans for a known-vulnerable package. A defender who inventories dependencies once a quarter is answering last quarter's question.

This is about tempo, not a ranking of which matters more. The supply-chain condition degrades faster and gets exploited sooner than the rest, so it earns first call on attention and budget. Treat it as the default #1 and argue yourself *down* from there if your context warrants.

The asymmetry that earns it the slot is that adversaries scan faster than defenders inventory. If build-generated inventory ever becomes universal and continuous, that gap narrows and the call would have to be re-argued. Abandonment is the part that would survive the change, because knowing which orphaned package you depend on does not make it maintained.

2.2.2 What cultivating it looks like

Tending Supply Chain is the practice of comprehension: closing the gap between what you shipped and what you actually understand about it. That gap is widening on its own, as more of the code entering your artifacts is generated faster than anyone reads it. The Coadaptive Security layer takes up that pressure directly. Here it is enough to say the lever is comprehension, and the work is refusing to ship what you have not understood.

- **Inventory, continuously.** A complete, current picture of direct and transitive dependencies, generated by the build rather than maintained by hand. The SBOM is the artifact; the comprehension is the point.
- **Establish provenance and sign.** Know where each artifact came from and that it arrived unaltered. Verified provenance turns "we think this is the library we meant" into something you can check.
- **Pin and rebuild.** Pin versions so a dependency cannot change under you silently. Rebuild from source where the supply matters enough to warrant it.
- **Catch the supply-specific attacks.** Dependency confusion, typosquatting, a compromised maintainer pushing a poisoned release, and now a poisoned MCP server or a tool whose description carries instructions the model will follow (https://cheatsheetseries.owasp.org/cheatsheets/MCP_Security_Cheat_Sheet.html). They are attacks on your trust rather than bugs in your code, which is why your own tests never catch them.
- **Weigh survivability, not only security.** A dependency's future is part of its risk. Take it on partly on whether the project is alive and whether you could carry it if the upstream stopped.

None of this is a control you install once. A dependency you vetted last year is a dependency that has shipped forty releases since. Comprehension is a standing practice or it is nothing.

2.2.3 When no one is left to tend it

The practices so far assume the thing you embedded keeps shipping. You inventory it, sign it, pin it, and wait for the next release to patch. But a dependency can fail in a way none of that touches: it can simply stop. The maintainer walks away, the project is archived, the company behind it folds or sells and the buyer turns it off. The code still sits in your tree doing its job, until the day a vulnerability lands in it and no fix is ever coming.

This is a different failure from a breach. A poisoned release is an attack on your trust; an abandoned dependency is the absence of anyone to trust. None of the breach-facing controls reach it. An SBOM tells you that you depend on the orphaned package, not how to keep it alive. A signature proves the last release was genuine, not that there will be another. Against a vendor that no longer exists, an indemnity is a claim on an empty estate.

core-js (https://www.theregister.com/2023/02/15/corejs_russia_open_source/) sits in a large share of the web, downloaded billions of times, and is maintained by one person who has said the money to keep it going has collapsed and that he is ready to walk. Nothing was breached. The risk is that a load-bearing dependency rests on a single human, and the bus factor upstream is one.

With open source you always have the right to fork. What you rarely have is the capacity to maintain what you forked. Forking core-js means owning a polyfill library you did not write and cannot staff. So the real question is your own bus factor on a dependency whose upstream bus factor is one, and you answer it before the abandonment, not after. Vendor the source so you can patch it yourself when no one upstream will. Weigh how alive a project is before you take it on: the release cadence, the number of hands, whether a foundation stands behind it or one tired person does. Funding the maintainer can buy time, but it is goodwill, not a control.

That is the embedded version. A delegated vendor can die the same way, on a deadline or in a bankruptcy, and because that is Third-Party the continuity plan for it is built there.

Containment does not save you from abandonment. This is a continuity problem, not a blast-radius one, carried by source escrow, the contractual right to fork or self-host, and your own capacity to take the code over, planned before you need it rather than improvised the week the upstream goes quiet.

2.2.4 How tending differs by position

The condition is the same everywhere; the way you can sustain it is not. A Craft shop tends Supply Chain with a single well-chosen managed scanner and a short list of vetted dependencies. A Lean enterprise tends it with a platform team, a dependency proxy, and a paved road that makes the secure choice the default choice. The Craft shop that reaches for the Lean enterprise's toolchain will drown in alerts it cannot triage. Match the practice to what your position can actually keep alive.

2.2.5 Where it shows up

A weak Supply Chain condition rarely announces itself in Supply Chain. It surfaces at Runtime, as the incident you trace back to a dependency you never inventoried. And it is cultivated, or skipped, in Process: a build pipeline that emits provenance and an SBOM as a byproduct is tending Supply Chain for free.

2.2.6 Next Steps

2.3 Third-Party

Third-Party asks a harder question than Supply Chain: do you know what you handed off, and who holds the bag when it fails? This is the function you delegated rather than the code you embedded. The payments processor, the data warehouse, the identity-verification provider, the cloud, the SIEM. Some of these are infrastructure. Others are core to how your product delivers value at all. Either way, you do not run them, and you cannot see inside them.

That is the defining trait of this condition: opacity. You will never inspect your cloud provider's hypervisor or audit your SIEM vendor's internal controls directly. Third-Party is weather. You cannot change it. You can only prepare for it. The way it fails is that they got breached, and you inherited the consequence anyway.

2.3.1 Liability moves; responsibility does not

A shared-responsibility model splits the work, and a contract can move a capped slice of the financial loss to the vendor through indemnities, a security super-cap, and a matched cyber policy. That money moves only after the failure, and only the part you could put a number on. It does not move the work of preventing the failure, or the duty to answer for it. Responsibility shifts; accountability never leaves you. Regulators treat it the same way: a board stays responsible for an outsourced function as if it ran in-house. When your identity provider leaks your users' credentials, your users do not call the vendor. They call you.

This is where compliance gets misread. Compliance is a market-access key, not a security proof. A vendor earns FedRAMP to unlock the federal market, not to become safe to depend on. The attestation expands their addressable market and retires none of your risk. Read a SOC 2 or a FedRAMP authorization as evidence that a vendor cleared a bar, never as a transfer of the responsibility that stays with you.

This is the Foundation accountability premise narrowed to the functions you delegate. No law lets you transfer the responsibility itself, only a capped slice of its cost, and that is why it cannot be argued away.

2.3.2 Operating is not delegating

The agent era splits the vendor relationship in two. A **provider** ships the model or tool; you take it on the same terms as any other opaque dependency, and everything above applies. An **operator** runs an agent in its own environment, wired to its own credentials, data, and systems. Most organizations running AI agents are operators, and operating is not delegating. You did not build the model and cannot see how it reasons, but the authority it acts with is something you assembled: the token it carries, the systems it can reach, the actions it can take.

When you delegate a function you hand off the work, and a contract can move a capped slice of the cost back. When you operate an agent you keep the work, and usually carry the cost alone. How it goes wrong depends on what feeds it. Exposed to input you do not control, the agent is the confused-deputy case. Kept to internal, trusted inputs, the risk runs through a compromised account, an insider, or its own error. Either way the agent widens the reach of each. Either way it acts with the authority you gave it, and that reach is yours to bound.

But you can only bound it as finely as the platforms beneath it allow: some reach you scope, some the platform sets for you. Your agent needs to read one project and the platform's token reads them all. To post a comment, it has to hold a scope that also lets it delete the repository. You did not choose that breadth and you often cannot remove it. The provider defines what is expressible in the authorization vocabulary; you stay accountable for what you express, and for choosing that substrate at all. The residual is yours to answer for twice over. A capability limit is only as fine as the platform's model allows; boundary enforcement is where you attenuate within that floor. Bound what you can at the layers you control, compensate at runtime for what the substrate cannot express, and count coarse primitives as a real cost when you choose what to build on.

Regulators are drawing the same line. The EU AI Act (https://digital-strategy.ec.europa.eu/en/policies/regulatory-framework-ai) separates the provider that builds and places a system from the deployer, the operator in this chapter's terms, that uses it under its own authority, and it shifts a deployer toward provider obligations precisely as it modifies and repurposes the system. The more you wire an agent into your operation, the more of it lands on you.

2.3.3 The operator beneath the operator

The chain rarely stops at one link. A provider you operate is often itself an operator, running a service or an agent wired to sub-providers you never contracted: a native integration, an OEM model under the vendor's label, a sub-processor three hops back. The vendor's AI feature turns out to be someone else's model, and your data is routed to a party whose name is on no contract you signed.

Three things compound down that chain. Opacity deepens: you could not see inside your direct provider, and you can see less inside the provider it leans on. Reach widens: the access the composition can exercise does not stop where your contract does, and it lands at a layer where you hold no controls and no recourse. And the liability cap reaches only the link you signed, so the matched cyber policy answers for your direct vendor, never for the sub-operator that actually leaked your data.

You cannot inventory what you cannot see, and here you cannot see in advance two layers down. That is the case that shows most cleanly why the answer is containment, not inspection. You do not certify the sub-dependency; you bound what the whole composition can reach and spend, so a failure at a link you never contracted spends only the authority you granted at the boundary. What the boundary cannot do is claw back data the sub-operator already holds and leaks; that residual is carried by contingency and contract, not by the boundary. Boundary enforcement is built for exactly the dependency you cannot inventory.

2.3.4 When a vendor dies

A breach is the failure you rehearse for most, but it is not the only way a vendor leaves you holding the bag. A vendor can also simply end: it goes bankrupt, it sunsets the product, an acquirer buys it and turns it off. The outage you fail over from is temporary; this one is permanent. When Facebook shut down Parse (https://techcrunch.com/2017/01/30/facebooks-parse-developer-platform-is-shutting-down-today/), roughly six hundred thousand apps had a year to migrate or go dark. That sunset ended softly, because Facebook open-sourced the server so dependents could host it themselves, which is what source escrow is meant to deliver. Most do not end that softly. The preparation has the same shape as the breach plan and is rarely written beside it: know which vendors are load-bearing, what your product does the day one is gone for good, and what continuity you secured while the vendor still existed to sign it. That continuity is the concrete part: source escrow, a self-host right, a wind-down clause.

2.3.5 What cultivating it looks like

Because you cannot inspect the vendor, you cultivate this condition by preparing for the day it fails.

- **Contain the blast radius before you need to.** Scope what each vendor can reach to the minimum the function requires. The vendor breach you survive is the one whose reach you bounded in advance.
- **Assume the breach and rehearse it.** Know which vendors are load-bearing, what happens to your product when one goes dark, for a day or for good, or goes hostile, and how you fail over. A contingency plan you have never tested is a hope.
- **Make the shared-responsibility line explicit.** Most third-party incidents trace to a boundary nobody owned because each side assumed the other had it. Write down who secures what, then check the assumption against reality.
- **Monitor the surface you can see.** You cannot watch their internals, but you can watch what they expose to you: the access they hold, the data crossing the boundary, the certifications lapsing.

2.3.6 How tending differs by position

A Craft shop tends Third-Party with a short list of critical vendors and a tested plan for the two or three it cannot live without. A Lean enterprise tends it with vendor risk scoring, pre-approved integration patterns, and failover rehearsed as a matter of course. Both are doing the same thing: bounding what an opaque dependency can do to them.

2.3.7 Where it shows up

Third-Party failures are bounded by Runtime containment, which is why the two conditions are read together. And the line between Third-Party and Supply Chain is the line between what you delegated and what you embedded. Keep them apart on the page so they stay apart in the work: one is an opacity problem you prepare for, the other a comprehension problem you can fix.

2.3.8 Next Steps

2.4 Process

Process asks whether the way you build produces security, or bolts it on afterward. The other three conditions are mostly about things: dependencies, vendors, running systems. This one is about the machine that makes the things. A build pipeline either emits security as a byproduct of how it works, or it leaves security to a review at the end that everyone has learned to route around.

The lever is feedback. Security that arrives while the work is happening gets fixed while the work is happening. Security that arrives in a gate three days later arrives as an interruption, and interruptions get bypassed.

2.4.1 Security as a property of the build

The aim is a build that makes the insecure version harder to ship than the secure one, not simply more checks. When provenance, dependency inventory, secret scanning, and policy checks run as part of the pipeline rather than alongside it, they stop being security activities at all and become properties of how code moves to production.

This is also where the other conditions get cultivated for free. A pipeline that emits an SBOM is tending Supply Chain. A pipeline that refuses a hardcoded secret closes a Runtime exposure before it exists. Process is the leverage point because a single change to the build strengthens more than one condition at once.

2.4.2 What cultivating it looks like

- **Put the feedback where the work is.** Checks that run in minutes, inside the pull request, with a clear path to the fix. A scan that takes two hours to return runs once and then gets disabled.
- **Make secrets structurally hard to leak.** Secrets that never enter source control, scanning that catches the ones that try, short-lived credentials over long-lived ones. The goal is a pipeline where leaking a secret takes effort.
- **Codify the secure default.** Paved roads, vetted templates, configurations that are safe before anyone tunes them. The secure path should be the path of least resistance.
- **Watch for the bypass.** A rising rate of skipped checks is the signal that the process has become friction instead of feedback. Measure it, and treat a bypass as a defect in the process rather than in the developer.

2.4.3 How tending differs by position

A Craft shop tends Process with a handful of checks wired into one pipeline. A Studio shop, simple but operationally ready, can automate from the start rather than building manual steps it will replace within a year. A Lean enterprise tends it with a platform team whose product is the paved road itself. The trap is the Mass pattern: heavy process that produces ceremony instead of security, gates that delay releases without reducing risk.

2.4.4 Where it shows up

Process is the condition through which the others are either cultivated or skipped. Its own failure mode is quiet. Nothing breaks today: the build keeps shipping, the checks keep passing, and the security those checks were meant to produce slowly stops being produced because the pipeline drifted and no one was watching the pipeline itself.

2.4.5 Next Steps

2.5 Runtime

Runtime asks whether you can sense what production is doing right now and respond before a user does. Everything upstream of here is preparation. Runtime is where the system meets real adversaries, real load, and real consequence, and where the question stops being whether you built it well and becomes whether you can tell what it is doing.

The lever is observability in the full sense: not only seeing, but seeing in time to act. Detection that arrives after the customer has already noticed is just confirmation.

2.5.1 Sense and respond

A healthy Runtime condition is a short loop. Something anomalous happens, you see it quickly, you contain it, you recover, and the system keeps serving everyone it was already serving. The three intervals that matter are time to detect, time to contain, and time to recover. Each one measures how tight the loop is, not a grade you pass.

The failure mode is rarely a missing tool. It is noise. A monitoring surface that emits a thousand low-value alerts trains its responders to ignore the one that matters. One high-fidelity signal that triggers a real response is worth more than full coverage no one can read.

2.5.2 What cultivating it looks like

- **Detect on behavior, not only signatures.** Know what normal looks like for your system so the abnormal stands out. Static rules catch yesterday's attack; a sense of baseline catches the one you have not seen.
- **Rehearse the response.** A playbook no one has run is a document, not a capability. The teams that contain incidents quickly are the ones that have practiced containing them.
- **Protect the data at the boundary.** Most breaches turn on a credential or a misconfiguration, not an exotic exploit. Least privilege, encrypted data, and audited access close the paths that actually get used.
- **Design for graceful failure.** Systems that degrade rather than collapse under pressure buy you the minutes response needs. Recovery is part of the condition, not a separate discipline.

2.5.3 How tending differs by position

A Craft shop tends Runtime with the monitoring its cloud provider gives it and a plan for the incidents it can actually foresee. A Lean enterprise tends it with correlated observability, automated response for known patterns, and threat hunting for the unknown ones. Both run the same loop of sense and respond, at different radii.

2.5.4 Where it shows up

Runtime is where the other conditions come to be judged. The uninventoried dependency from Supply Chain, the vendor breach from Third-Party, the gap a drifting Process stopped catching: each of them, if it is going to hurt you, hurts you here. A strong Runtime condition is the last containment when an upstream condition was weak. It does not substitute for tending them.

2.5.5 Next Steps

2.6 Adaptive Capacity

Adaptive Capacity is whether the system as a whole can absorb a surprise it was not designed for and keep working. It is the condition that used to be called Continuous Learning, named now for what it actually is. The other four conditions each map to a function you can put on an org chart. This one does not. You assess whether the system has it. You cannot hand it to a team.

2.6.1 The idea comes from ecology

In 1973 the ecologist C.S. Holling drew a line between two things people had been treating as one. Stability is how fast a system returns to where it was after a small disturbance. Resilience is something else: how much a system can absorb and reorganize around before it becomes a different system altogether. A forest that burns and regrows is not stable while it burns; it is resilient. The capacity lives in the whole web rather than in any single tree, and it is never finished, because the system and the things stressing it keep changing against each other.

Security works the same way. A field of practice grew up carrying Holling's insight out of ecosystems and into engineered ones: Erik Hollnagel reframed safety as the presence of adaptive capacity rather than the absence of failure, David Woods described resilience as graceful extensibility at the brittle edges of a system, Richard Cook catalogued how complex systems fail and keep running anyway, and Kelly Shortridge made the case for treating security as ecology rather than enforcement. The full citations are in the references. The throughline is one claim: a system you certify once and trust to hold is already falling behind, because the things trying to break it keep changing while it stands still.

That throughline is a bet, and it is worth saying what would make it wrong. Adaptive Capacity earns its place only in a world where the threats keep moving. If the landscape ever froze, a system you certified once would hold, and this whole condition would be dead weight. The wager SF^2 makes is that it does not freeze: adversaries pick up the same automation defenders do, and the pace of change is itself the thing being defended against. Treat that as the assumption to check, not a law of nature.

The sturdier half of this needs no adversary at all. Your own system does not hold still: it grows, and as it grows it outruns the people who understand it, so the picture you hold of your own attack surface goes stale on its own. That alone would keep you tending, even against an enemy that never improved. The threat side only adds to it, and in more ways than the obvious one. Attackers do not have to get smarter for the landscape to change. There can simply be more attackers, because the same automation that helps you lowers the cost of attacking you, and old tools recombine into attacks no one had to invent. The mechanism does not rest on attackers getting cleverer. It rests on two things that move on their own: a system you cannot fully hold, and a threat surface that widens even when no single attacker sharpens.

2.6.2 What it looks like, present and absent

You cannot install Adaptive Capacity, but you can see it. It shows up as a blameless post-incident review that actually changes something, not as a document filed and forgotten. It shows up as feedback loops that shorten over time, and as the organization sensing a shift in the threat landscape and adjusting before it gets hit rather than after. It is the difference between learning from an incident and merely surviving it.

When it is missing, the other four conditions can each look fine on a maturity chart while the system stays brittle, because nothing is teaching it to bend. The reviews happen but nothing changes. The same class of incident recurs under a different name. The program is busy and not adapting. Without it, you are just repainting the same four walls on a fixed cadence.

2.6.3 Why it runs across the other four

Adaptive Capacity is how all four of the other conditions improve at all. It is the health of all four at once rather than a fifth lane staffed beside them, the question of whether Supply Chain, Third-Party, Process, and Runtime are getting better faster than their failure modes are. A maturity chart can tell you whether each condition is filled in. Only Adaptive Capacity tells you whether the system is still adapting faster than the things trying to break it, and that is the question worth asking.

2.6.4 The seam to the Coadaptive layer

This is where the base framework meets the Coadaptive Security layer. The contest between a system that keeps changing and adversaries who keep changing in response is the same predator-and-prey pressure that runs through any living system, and it sharpens once the

system includes AI that writes, decides, and acts. Adaptive Capacity is the condition that carries the base framework up into that layer. It is the reason SF^2 treats security as something a system keeps doing rather than a state a system reaches.

This chapter says security is never finished. The Coadaptive layer says something that sounds like the opposite: you can prove a hard limit on what one component is allowed to do, and that limit holds without tending. Both are true, because they describe different things. You can prove a part. You cannot finish the whole. A single component has a fixed job, so you can bound it and trust the bound. The system those components add up to keeps meeting new surprises, and they land at the seams between the parts, where no single proof reaches. So you prove each piece and you tend the system. Neither move does the other's job.

2.6.5 Next Steps

You have now worked through the five universal security conditions. The next section is how you decide which version of tending each one your organization can actually sustain.

3. Strategic Positioning

3.1 Two-Axis Positioning Model

3.1.1 Understanding Your Strategic Starting Point

Rather than using traditional maturity models that assume linear progression, this framework positions software factories on **two independent dimensions** that determine your optimal security approach.

Why Not Maturity Models?

Traditional security maturity models imply everyone should follow the same path. SF² recognizes that a 10-person startup with modern cloud infrastructure shouldn't implement security the same way as a 5,000-person enterprise with legacy systems, even if both need strong security.

This is contingency theory applied to security: organizational research has held for decades that there is no single best way to organize, only the structure that fits an organization's environment, scale, and technology (Donaldson 2001; Horne, Maynard & Ahmad 2017). SF² carries that finding into security program design: your position sets your strategy, not your rung on a ladder everyone is told to climb.

3.1.2 The Strategic Positioning Framework

Organizations can be assessed along two critical axes:

Blast Radius Axis (What a Failure Can Reach)

The horizontal axis is **inherent blast radius**: how far a failure could reach if containment fails, given everything your systems are allowed to do. It is set by the authority you have handed your automation and agents, not by how many people you employ. Headcount and team count were the old stand-in, and the stand-in held while a bigger system meant more people to run it. AI broke that correlation. A fifteen-person shop running fifty agents with broad tool access has the reach of an enterprise and the headcount of a studio, and a security strategy keyed to headcount cannot see the reach. So the axis names the reach directly.

Small reach: automation that touches one bounded surface, authority scoped per task, a worst-case failure contained to a single blast cell.

Large reach: automation that crosses trust boundaries (production, data, and identity at once), standing authority broad enough that one misused credential is an enterprise event, a worst-case failure that cascades across the estate.

Headcount survives here as a legacy proxy: it used to predict reach, and where a small team still holds small authority it still does. AI is what severs headcount from reach, and the axis follows the reach.

This is *inherent* reach, measured as if containment failed. What you build to stop a failure reaching that far, the boundary enforcement that holds a compromised component to the authority it was granted, is the other axis: readiness. Keeping the two apart is what keeps the model honest. Your reach is mostly what you have decided to let your systems do; your readiness is what you build to contain it. The containment floor that bounds the blast is the same boundary enforcement the coadaptive layer specifies at the substrate: this chapter names the position, that chapter builds the floor.

This runs asymmetrically, and the asymmetry is the point. A small shop can leap to large reach by granting fifty agents broad authority, but a large enterprise cannot shrink below its own surface area: inherent reach is the sum of every authority you have granted, and a big estate has granted a great deal. Past a certain scale that sum is simply large, and the horizontal axis stops telling enterprises apart. So the two ends of the model play different games. Small shops fight to stay on the left, one over-scoped agent fleet away from teleporting to Mass. Large enterprises take their reach as given and fight the vertical: the whole move is proving containment and climbing to Lean.

A test keeps the two axes from blurring. Place yourself in two sentences: a reach sentence using only the verbs of authority (what the automation can reach, is granted, is allowed to touch) and a readiness sentence using only the verbs of control (what is contained, caught, reviewed, segmented). If a containment word is carrying your reach claim, you have measured the wrong axis.

Operational Readiness Axis (How You Operate)

Lower Readiness:

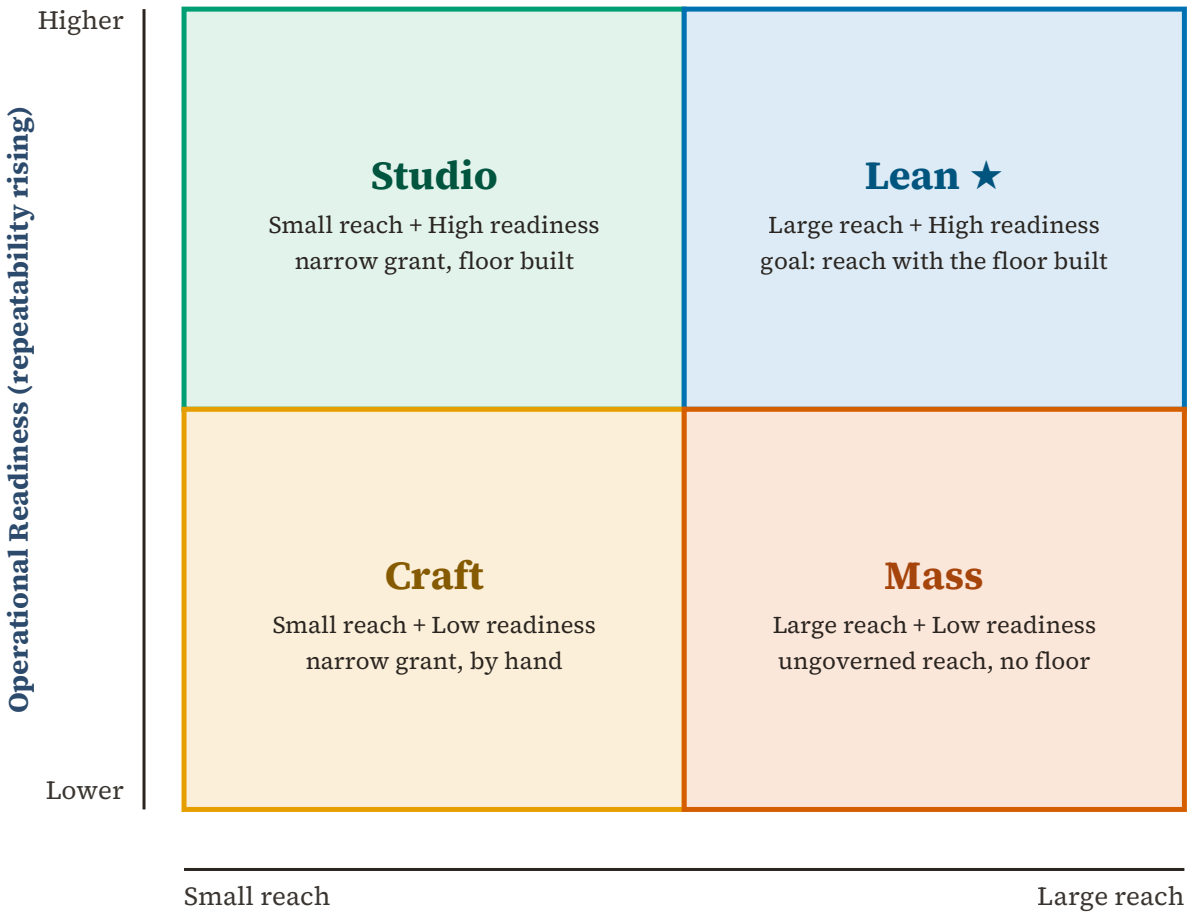
- Manual processes
- Legacy infrastructure
- Limited automation
- Tribal knowledge
- Reactive operations

Higher Readiness:

- Automated pipelines
- Modern infrastructure
- API-driven operations
- Documented processes
- Proactive operations

3.1.3 The Four Strategic Positions

These two axes create four distinct strategic positions, each with different security approaches:



*The horizontal axis is inherent blast radius (reach if containment fails); the vertical axis is the lean transformation, repeatability and proven containment rising. **Mass** is Lean's large reach without Lean's floor: ungoverned scale, not big batch.*

Reading the two axes

The horizontal axis is **blast radius**: the inherent reach of a failure if containment fails, set by what authority your automation holds. The vertical axis is the **lean transformation**: moving up means the work becomes more repeatable and a failure provably stays contained. The goal position is **Lean**: large reach with the floor built.

The names are production modes, about flow and containment

The four names map to production modes, and the analogy is about flow and containment, not volume. Lean is the cleanest fit: Toyota's line stops the instant a defect appears so it cannot propagate, which is large flow contained by construction, exactly large reach with the floor built. Craft is the artisan touching one or two things by hand. Studio is the small, bounded, modern shop. **Mass is Lean's large reach without Lean's floor: ungoverned scale, not big batch.** Under the reach axis the border that matters is against Lean, not against Ford's assembly line.

Studio (Small reach + High Readiness)

Characteristics:

- Small teams with modern technology stack
- Cloud-native infrastructure
- Automated from inception
- DevOps/Platform engineering culture
- Fast iteration and experimentation

Security Approach: Enable security through modern tooling and self-service capabilities

Strategic Focus: Use technology advantages while building organizational scale

Typical Organizations:

- Modern startups (post-Series A)
- Cloud-native SaaS companies
- Platform teams in larger organizations

> **Studio in Action**
>
> A fifteen-person SaaS company whose automation was never granted broad authority: each service's deploy actor is allowed to touch only its own service, so the worst failure can reach a single surface. Per-credential scoping and human review on consequential actions keep it that way, so the small reach stays a contained one. Small reach, high readiness. A small team is not automatically Studio: grant fifty agents broad tool access to production and the reach is enterprise-scale on startup discipline, which is Mass.

Lean (Large reach + High Readiness)

Characteristics:

- Large-scale operations
- Sophisticated DevOps practices
- Comprehensive observability
- Enterprise processes
- Continuous learning culture

Security Approach: Orchestrate enterprise security architecture with continuous learning

Strategic Focus: Optimize security operations at enterprise scale while maintaining innovation

Typical Organizations:

- Cloud-native enterprises
- Modern financial services
- Advanced SaaS companies
- Tech giants

> **Lean in Action**
>
> A two-thousand-person organization whose automation is granted authority across the whole estate, so a failure could reach broadly. Every request runs under per-request attenuation enforced in the platform, so a compromised component is held inside the authority it was granted. Large reach, contained by construction. Lean is large blast radius with the floor already built.

Craft (Small reach + Low Readiness)

Craft is a method, not a price tag

Craft here describes non-repeatable hand-work: snowflake-per-build, every deployment a little different. It names the *method* (low repeatability), not quality, premium positioning, or boutique branding. A Craft organization can do excellent security work; it just does it by hand each time.

Characteristics:

- Small-scale operations
- Legacy systems
- Manual processes
- Limited automation
- Constrained resources

Security Approach: Focus on operational readiness foundations while managing essential security controls

Strategic Focus: Build operational capabilities while maintaining security coverage

Typical Organizations:

- Early-stage startups
- Small businesses
- Non-tech companies with limited IT
- Organizations in regulated industries with legacy systems

Craft in Action

A twenty-person company on legacy infrastructure whose automation is granted little: one or two systems, deployed by hand. Nothing proves a failure stays put; it relies on the systems behaving. Small reach, low readiness.

Mass (Large reach + Low Readiness)

Characteristics:

- Large scale with legacy constraints
- Technical debt
- Mixed manual/automated processes
- Organizational complexity
- Transformation in progress

Security Approach: Pragmatic security within constraints while enabling gradual modernization

Strategic Focus: Balance current operational demands with strategic modernization investments

Typical Organizations:

- Traditional enterprises undergoing digital transformation
- Financial institutions with legacy infrastructure
- Healthcare organizations
- Government agencies

Mass in Action

A five-thousand-person enterprise whose automation is granted broad authority across legacy and cloud, and equally the fifteen-person shop that handed fifty agents production access. Neither can prove a compromised component stays inside its grant. Both are large reach, low readiness. Mass is ungoverned reach, not size.

3.1.4 Assessing Your Position

Use these questions to determine your organization's position:

Blast Radius Assessment (inherent reach)

Question	Simple (small reach)	Complex (large reach)
Reach of most-capable automation: the largest set of systems any one automated actor (pipeline, agent, service account) can touch without a human in the loop?	One bounded surface (single service/datastore)	Crosses trust boundaries (prod + data + identity); org-wide
Worst-case propagation: if your single most-privileged non-human identity were fully compromised now, how far does damage reach before something *not also compromised* stops it?	Contained to one blast cell	Cascades across the estate
Autonomy depth: how much can automation *do*, not just read, without a human checkpoint (open and merge code, move money, grant access, drop data)?	Read or propose only; humans commit consequential actions	Acts and commits consequential actions unattended
Authority concentration: does any single credential, role, or agent hold standing authority broad enough that its misuse is an enterprise event?	No; authority attenuated per task	Yes; broad standing authority exists

Operational Readiness Assessment

Question	Lower Readiness	Higher Readiness
Containment verifiability: can you *prove* (not assert) that a compromised component cannot exceed the authority you granted it?	No; you rely on it behaving	Yes; enforced at the boundary
Deployment process?	Manual	Fully automated
Infrastructure?	Legacy/on-prem	Cloud-native/hybrid
Documentation?	Tribal knowledge	Comprehensive docs
Observability?	Limited/reactive	Comprehensive/ proactive
Change velocity?	Weeks/months	Hours/days

3.1.5 Why This Matters for Security

Position is a sequencing and funding diagnostic, not an architecture one. It does not decide whether you adopt the containment floor; every quadrant owes the same one. It decides how fast you reach it and what you fund first. Within that, your position determines:

1. **Funding order**: Which security investments to make first, and which to defer
2. **Rollout pace**: How fast you can stand up capabilities without outrunning the organization
3. **Timeline expectations**: How long transformation realistically takes
4. **Mechanism fit**: Which implementation meets the containment floor at your scale
5. **Success metrics**: What good looks like at your stage

Common Mistake

Implementing Lean-level security programs in a Mass or Craft organization often leads to: - Failed tooling implementations - Frustrated security and development teams - Wasted budget on capabilities you never put to use - Security becoming a bottleneck instead of enabler

3.1.6 Strategic Movement Paths

Most organizations benefit from moving toward the Lean position, but the path depends on your starting point:

Movement Strategies

Current Position	Optimal Path	Primary Investments	Timeline	Notes
Craft Studio	Build the floor without widening reach	Per-service scoping, boundary enforcement, grants held narrow	12-18 months	Single-axis climb (up); reach stays small
Studio Lean	Grow reach with the floor traveling	Scale automation reach while attenuation and boundaries hold	24-36 months	Reach grows right, containment travels with it
Mass Lean	Build the floor under reach you already hold	Containment over the estate: per-request attenuation, boundary enforcement, retire ungoverned grants	36-60 months	The enterprise's main game: single-axis climb (up)
Mass Craft	Shrink reach by cutting granted authority	Retire systems, narrow grants, reduce agent scope	18-30 months	Single-axis move (left); hard to surrender reach the business uses
Craft Mass	Reach outran the floor	Broad automation or agents added without containment	fast, often involuntary	Drift, not a goal: the move to guard against

Executive Insight

The Mass Lean path is the most common enterprise move. Your reach is already large; the entire job is vertical, proving containment over the estate you already hold. That is hard, but it is one axis, not two.

3.1.7 Using Position to Guide Security Strategy

The lists below are four sequences toward the same containment, each with tooling fit to its scale. Every quadrant owes deny-by-default limits on the authority review cannot police at scale; what changes below is the order of investment and the tooling that fits the operational reality. Each list is a funding order for your scale, not the security you end up with.

For Studio:

- Use cloud-native security services
- Implement policy-as-code from inception
- Build security into platform capabilities
- Enable developer self-service

For Lean:

- Orchestrate enterprise security architecture
- Build internal security platforms
- Optimize at scale with automation
- Continuous security improvement programs

For Craft:

- Focus on foundational security controls
- Manual but systematic approaches
- Gradual capability building
- Use managed security services

For Mass:

- Pragmatic hybrid security approaches
- Risk-based prioritization (critical systems first)
- Incremental modernization
- Balance legacy and modern security controls

3.1.8 Next Steps

Now that you understand strategic positioning, explore the specific characteristics and recommended approaches for each position:

Naming note (v0.5 v0.6)

These four positions were named Visionaries, Leaders, Niche Players, and Challengers in v0.5. They are now **Studio, Lean, Craft, and Mass**. See the quadrant rename mapping for the full crosswalk and the reasoning behind the change.

3.2 Four Strategic Positions

3.2.1 Understanding Where You Are Determines What You Should Do

The two-axis positioning model creates four distinct strategic positions. Each position faces different security challenges and requires different sequencing and pace to scale security capabilities. What it does not change is what the program must ultimately contain.

Critical Insight: Your position changes almost everything about your security program. It does not change what the program is for. Every quadrant owes the same containment: deny-by-default limits on authority. Code review and human comprehension were never able to police that authority at the scale enterprises already run. That containment is the floor. How you enforce it fits your scale: a small shop on a single workload contains it with host isolation and tight access scoping; a large mesh contains it with workload identity. Same floor, mechanism fit to scale. What your position decides is how fast you reach that containment, what you fund first, and what pace the rest of the program can sustain. A Studio organization that funds security like a Lean enterprise creates coordination overhead it does not need; a Mass organization that promises Studio timelines will miss them.

3.2.2 The Four Strategic Positions

Studio: Small Reach + High Readiness

> **Who You Are**:
>
> • Small teams (typically under 50 engineers)
>
> • Cloud-native infrastructure from inception
>
> • Automated CI/CD pipelines and modern DevOps practices
>
> • API-driven operations and infrastructure-as-code
>
> • Minimal legacy technical debt
>
> **Your Security Approach**: Enable security through modern tooling and self-service capabilities. Use your technology advantages rather than building processes that assume operational constraints you don't have.
>
> **Strategic Focus**: Build security automation from day one. Don't create manual processes you'll need to replace in 18 months. Your high operational readiness makes automation achievable now, and your small reach means there's little cross-boundary authority to coordinate and govern.

Key Investment Priorities:

1. **Automated Supply Chain Security**: Dependency scanning and update automation with minimal manual intervention
2. **Pipeline-Native Security**: Security checks integrated directly into CI/CD with immediate developer feedback
3. **Self-Service Security Capabilities**: Cloud platform security features that "just work" without security team involvement
4. **Observability-First Runtime Security**: Cloud-native monitoring with automated alerting and incident response

Common Pitfalls to Avoid:

> **Don't Build for Future Complexity**
>
> **Risk**: Implementing enterprise-scale security architecture "because we'll need it eventually"
>
> **Symptoms**: Multi-week security design reviews, approval workflows for two-person teams, committee decision-making
>
> **Solution**: Build for your current reality. Add governance weight only when your reach actually widens.

Success Indicators:

- **6 Months**: Automated dependency scanning covers 95%+ of codebases, security checks integrated in all pipelines with <5 minute feedback loops
- **12 Months**: Zero manual security reviews for standard deployments, developers resolve 80%+ security issues without security team involvement
- **24 Months**: Security automation enables 3-5x growth without proportional security team expansion

Movement Path: As you grow, you'll naturally grant more authority and widen your reach. Your challenge is keeping containment ahead of that reach: transition to Lean rather than falling into the Mass trap of widening reach faster than you can contain it.

Lean: Large Reach + High Readiness

> **Who You Are**:
>
> - Large-scale operations (100+ engineers, multiple teams)
> - Sophisticated DevOps and platform engineering practices
> - Comprehensive observability and automation across the enterprise
> - Mature change management and incident response processes
> - Strong engineering culture with continuous learning focus
>
> **Your Security Approach**: Orchestrate enterprise security architecture with integrated learning culture. Enable security at scale through platform engineering and federated champions who configure the boundaries the platform enforces.
>
> **Strategic Focus**: Optimize security operations at enterprise scale while maintaining innovation velocity. Your advantage is operational sophistication. Use it to embed security into engineering platforms rather than building centralized security bottlenecks.

Key Investment Priorities:

1. **Security Platform Engineering**: Self-service security capabilities available across all teams and products
2. **Federated Security Champions**: Engineers inside each team who set the team-level boundaries the platform then enforces: capability scopes, deny-by-default defaults, the guardrails their team ships inside. The platform team authors the paved road; champions set the boundaries within it and feed the threat models that shape it. The leverage is configuration, not inspection, so enforcement scales without a human in the path of every change. The boundary-setting is itself threat modeling and risk judgment, not a config chore: the enforcement is delegated to the platform, the judgment is not.
3. **Enterprise Architecture Integration**: Security embedded in platform decisions and organizational standards
4. **Continuous Security Intelligence**: Automated threat detection with predictive insights and proactive response
5. **Cultural Scaling Mechanisms**: Learning from incidents and scaling security knowledge across the organization

Common Pitfalls to Avoid:

> **The Coordination Overhead Trap**
>
> **Risk**: Security program becomes a bottleneck due to enterprise coordination requirements
>
> **Symptoms**: Multi-week security architecture reviews, declining developer satisfaction, increasing workarounds
>
> **Solution**: Federate security decision-making to engineering teams with clear guardrails. Centralize platform capabilities, not every security decision.

Success Indicators:

- **6 Months**: Security platform adoption across 70%+ of teams, measurable reduction in security review wait times
- **12 Months**: Security champions program active in all major teams, 60%+ of security issues resolved without central team involvement
- **24 Months**: Security capabilities scale automatically with organizational growth, security becomes competitive advantage

Movement Path: Stay in Lean by continuously evolving security capabilities to match organizational scale. Falling into Mass happens when operational readiness can't keep pace with complexity growth, so maintain platform investment velocity.

Craft: Small Reach + Lower Readiness

> **Who You Are**:
>
> • Small teams with limited resources
>
> • Legacy systems or manual deployment processes
>
> • Limited automation and observability
>
> • Straightforward product or service offering
>
> • Often resource-constrained or bootstrapped
>
> **Your Security Approach**: Focus on operational readiness foundations while managing essential security controls. Accept manual security processes initially, but invest strategically in readiness improvements that enable future automation.
>
> **Strategic Focus**: Build operational capabilities systematically while maintaining security coverage. Your small reach means a failure stays contained even when caught by hand, so you can afford some manual processes temporarily. Use this breathing room to invest in readiness infrastructure.

Key Investment Priorities:

1. **Critical Supply Chain Controls**: Basic dependency scanning and critical vulnerability management
2. **Foundational Readiness Infrastructure**: CI/CD basics, infrastructure documentation, change management fundamentals
3. **Essential Runtime Monitoring**: Basic production monitoring and incident response capabilities
4. **Process Documentation**: Record what works to enable eventual automation and knowledge transfer

Common Pitfalls to Avoid:

> **The Permanent Manual Process Trap**
>
> **Risk**: Treating manual processes as acceptable long-term state rather than temporary necessity
>
> **Symptoms**: "This is how we've always done it," no automation roadmap, resistance to process changes
>
> **Solution**: Be explicit that manual processes are temporary. Document current state as foundation for future automation. Invest consistently in readiness improvements.

Success Indicators:

- **6 Months**: Critical dependencies monitored, basic CI/CD pipeline operational, documented security processes
- **12 Months**: 30-50% automation of security checks, measurable reduction in manual security work
- **18-24 Months**: Operational readiness sufficient to support automation investments, movement toward Studio position

Movement Path: Move toward Studio by investing in operational readiness. This single-axis movement is achievable. Prioritize cloud migration, DevOps tooling, and automation infrastructure over widening your reach.

Mass: Large Reach + Lower Readiness

> **Who You Are**:
>
> • Large-scale operations with legacy constraints
>
> • Significant technical debt and mixed infrastructure (legacy + modern)
>
> • Manual processes coexist with automated systems

- Complex compliance and regulatory requirements
- Multiple organizational changes or acquisitions

Your Security Approach: Pragmatic security within constraints while enabling gradual modernization. Accept that full transformation takes 3-5 years. Don't promise 12-month miracles.

Strategic Focus: Balance current operational demands with strategic modernization investments. Your complexity prevents rapid transformation, but your scale justifies investment in foundational improvements.

Key Investment Priorities:

1. **Pragmatic Supply Chain Controls**: Risk-based approach given limitations: focus on critical paths and crown jewels
2. **Hybrid Security Architecture**: Solutions that work across legacy and modern systems without requiring full modernization
3. **Strategic Technical Debt Reduction**: Systematic elimination of highest-risk legacy constraints enabling future automation
4. **Change Management and Communication**: Organizational alignment on multi-year transformation reality and sequencing
5. **Quick Wins for Credibility**: Visible improvements that build organizational confidence in long-term transformation

Common Pitfalls to Avoid:

The Diagonal Transformation Trap

Risk: Attempting to shrink reach AND build containment at the same time, the highest failure risk

Symptoms: Aggressive transformation timelines, simultaneous re-platforming and process overhauls, widespread disruption

Solution: Sequence changes carefully. Typically, build the containment floor first (the Mass Lean climb) to enable automation at scale. Accept 36-60 month timeline.

Success Indicators:

- **12 Months**: Hybrid security solutions operational, critical technical debt reduction projects started, transformation roadmap with executive buy-in
- **24 Months**: Measurable automation improvements in high-value areas, improved operational readiness metrics, cultural momentum toward transformation
- **36-60 Months**: Substantial progress toward Lean position, security increasingly enabling business rather than constraining it

Movement Path: Most Mass should move toward Lean by investing in operational readiness while managing existing complexity. This is difficult but achievable with sustained executive support and realistic timelines. Attempting to simplify operations (Mass Craft) rarely succeeds. It requires major business restructuring.

3.2.3 Position Assessment Tool

Use these questions to confirm your strategic position:

Blast Radius Assessment

Measure inherent reach: how far a failure could travel if containment failed, given everything your automation is allowed to do. Answer with the verbs of authority (can reach, is granted, is allowed to touch), not the verbs of control.

Question	Small reach	Large reach
Reach of most-capable automation: largest set of systems any one automated actor (pipeline, agent,	One bounded surface (single service or datastore)	Crosses trust boundaries (prod, data, and identity at once); org-wide

Question	Small reach	Large reach
service account) can touch without a human in the loop?		
Worst-case propagation: if your single most-privileged non-human identity were fully compromised now, how far does damage reach before something not also compromised stops it?	Contained to one blast cell	Cascades across the estate
Autonomy depth: how much can automation do, not just read, without a human checkpoint (open and merge code, move money, grant access, drop data)?	Read or propose only; humans commit consequential actions	Acts and commits consequential actions unattended
Authority concentration: does any single credential, role, or agent hold standing authority broad enough that its misuse is an enterprise event?	No; authority attenuated per task	Yes; broad standing authority exists

Operational Readiness Assessment

Question	Lower Readiness	Higher Readiness
Deployment Automation	Manual or partially automated deployments	Fully automated CI/CD with infrastructure-as-code
Infrastructure	Legacy systems, manual configuration	Cloud-native or hybrid with API-driven management
Observability	Limited monitoring, reactive incident response	Comprehensive observability with proactive alerting
Process Documentation	Tribal knowledge, inconsistent documentation	Well-documented, standardized processes

Your Position: Plot your answers to identify which quadrant best describes your current state.

3.2.4 Strategic Position and Investment Strategy

Your strategic position determines your optimal approach to the investment portfolio:

Position	BAU Approach	Scaling Investment Focus	Timeline to ROI
Studio	Minimal BAU burden initially	Automation-first from inception	6-12 months
Lean	Systematic BAU constraint	Platform capabilities at scale	12-18 months
Craft	Manageable manual BAU	Foundational readiness infrastructure	18-24 months
Mass	High BAU burden requiring constraint	Strategic debt reduction + readiness	24-48 months

3.2.5 Common Position Misidentification

Startup Founder Self-Assessment: "We're obviously Craft, we're small and scrappy!"

Reality Check: If you're cloud-native with automated deployments, you're Studio. Your small size keeps your reach small, but your modern stack is high readiness. Don't build manual processes just because you're small.

Established Enterprise Self-Assessment: "We're Lean, we have mature security programs!"

Reality Check: If you're running significant legacy infrastructure with manual deployments, you're Mass regardless of security team size. Your complexity is high, but operational readiness is constrained by technical debt.

High-Growth Startup Self-Assessment: "We're Studio scaling successfully!"

Reality Check: If you're widening reach faster than you can contain it, you're moving toward Mass (large reach, low readiness). Maintain containment investments or you'll face the diagonal transformation trap.

3.2.6 Next Steps

1. **Confirm Your Position**: Use the assessment tools above to validate your quadrant
2. **Review Movement Paths**: Understand how to transition strategically
3. **Read Your Implementation Guide**: Apply position-specific guidance from `06-implementation/`
4. **Evaluate Contextual Modifiers**: Understand how your specific situation affects implementation

---

3.3 Strategic Movement Paths

3.3.1 How Organizations Transition Between Positions

Understanding your current strategic position is only the first step. Most organizations benefit from moving toward the Lean position (Large reach + High Readiness), but **the path you take determines your likelihood of success**.

The Transformation Reality

Quadrant transitions are organizational transformations, not technology projects. The tractable moves change one axis at a time. But one axis is not automatically quick: widening reach is fast and cheap (you just grant the authority), while building the containment floor under reach you already hold is slow. Climbing from Mass to Lean is a single-axis move on paper and still takes 36-60 months, because the floor has to cover the whole estate. The moves that fail most often are the ones that widen reach and build containment at the same time.

That gap between fast and slow is the install clock against the absorb clock: how fast you can stand a change up versus how fast the organization actually runs on it. Granting authority installs in a day, building the floor the organization runs on takes quarters (the two clocks get their full treatment in Change Capacity). And the reason doing both at once fails most is that they draw on one shared absorption budget, not two. The same people, sponsor, and approval path carry both, so each absorbs worse. Sequence the heavy moves; do not stack them.

3.3.2 The Strategic Movement Framework

The Goal Position: Lean

Why Lean: The Lean quadrant combines enterprise-scale capabilities with operational excellence. Organizations in this position achieve:

- Security capabilities that scale automatically with organizational growth
- Developer experience improvements that increase rather than constrain velocity
- Platform approaches that enable innovation while maintaining security controls
- Cultural integration where security is a competitive advantage, not a constraint

Reality Check: Not every organization needs to reach Lean immediately. Your optimal target depends on your business trajectory and organizational change capacity.

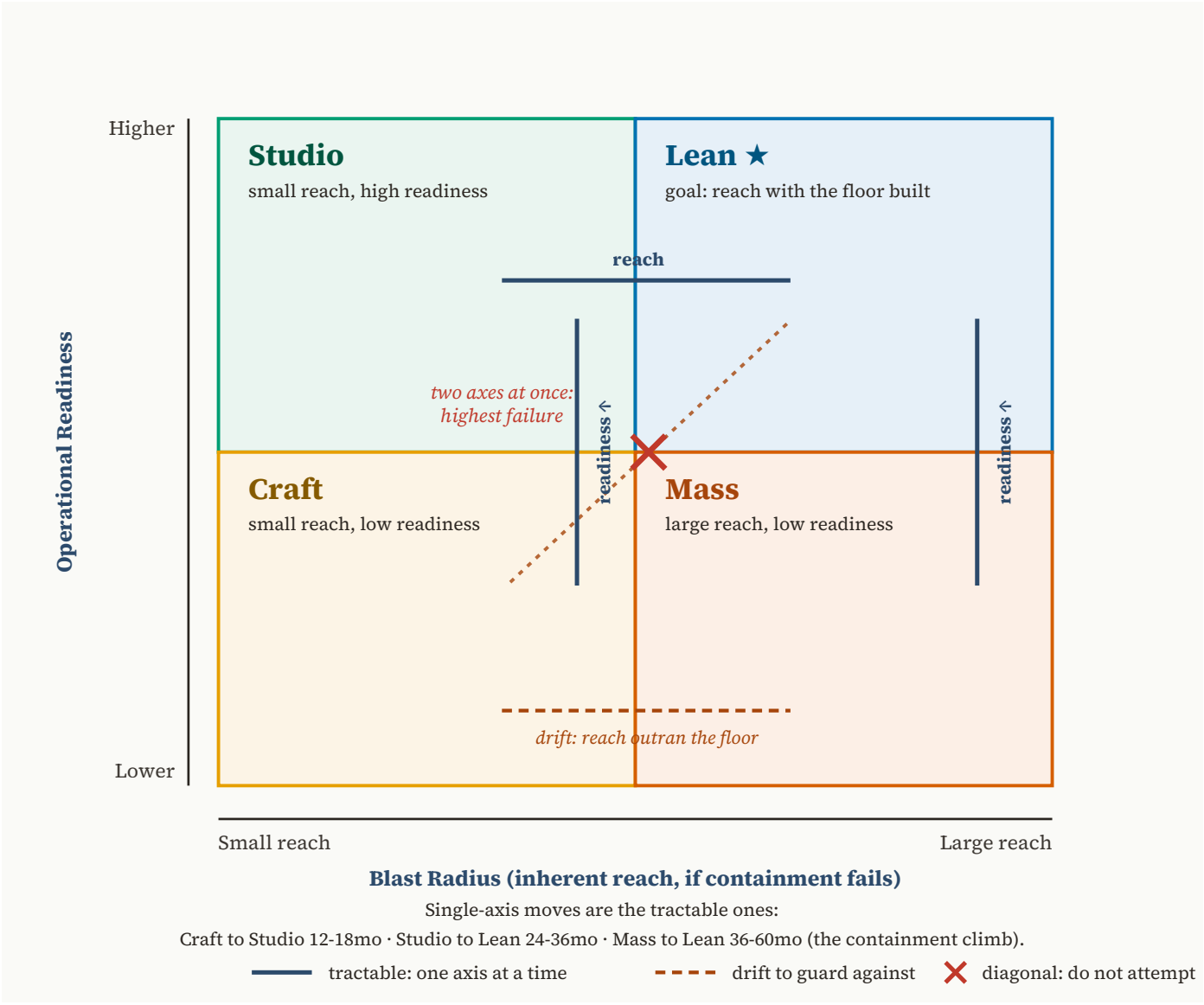
3.3.3 Six Strategic Movement Paths

From To	Movement Type	Timeline	Success Probability	Strategic Guidance
Craft Studio	Single-axis (Readiness ↑)	12-18 months	High	Operational Readiness Path
Craft Mass	Single-axis (Reach ↑)	18-24 months	Moderate	Reach Scaling Path
Studio Lean	Single-axis (Reach ↑)	24-36 months	High	Enterprise Scaling Path
Mass Craft	Single-axis (Reach ↓)	18-30 months	Low	Simplification Path
Mass Lean	Single-axis (Readiness ↑, large reach)	36-60 months	Moderate	The Containment Climb
Lean Lean	Position Maintenance	Ongoing	High	

From To	Movement Type	Timeline	Success Probability	Strategic Guidance
				Excellence Sustainment

The timelines above are install-and-build floors. A move is not done until it is absorbed, and two major moves that run through the same quarters draw on one absorption budget, not two. Sequence by that budget rather than by what your calendar can install (see Change Capacity).

These ranges and likelihood ratings are practitioner estimates synthesized from transformation experience, not measured outcomes. SF² treats them as hypotheses to validate rather than findings, and refining them against real transformations is part of the framework's research agenda.



The three tractable moves change one axis at a time. The dashed path is drift to guard against; the crossed diagonal is the two-axis move that fails most often. Lean is the goal position.

3.3.4 Path 1: Craft → Studio (Operational Readiness)

> **Movement Type**: Single-axis (Increasing Operational Readiness)
>
> **Timeline**: 12-18 months
>
> **Success Probability**: High

When to Choose This Path

Yes, if:

- Your business remains relatively simple (single team or product focus)
- You have budget/appetite for infrastructure modernization
- Legacy technical debt is constraining your business velocity
- You want to enable future automation before your reach widens

No, if:

- You're simultaneously scaling teams and organizational complexity rapidly
- Legacy systems have significant customer dependencies requiring careful migration
- Business model requires immediate complexity scaling

Key Investment Priorities

1. **Cloud Migration**: Move from on-premises or manual infrastructure to cloud platforms
2. **CI/CD Implementation**: Automated build, test, and deployment pipelines
3. **Infrastructure as Code**: Terraform, CloudFormation, or similar tooling
4. **Observability Foundation**: Centralized logging, metrics, and monitoring
5. **Security Automation**: Dependency scanning, SAST/DAST integration into pipelines

Sequencing Strategy

Months 1-6: Foundation

- [ ] Cloud platform selection and initial migration planning
- [ ] Basic CI/CD pipeline for new services or non-critical systems
- [ ] Observability platform implementation with critical system coverage
- [ ] Team training on cloud-native practices and DevOps principles

Months 7-12: Acceleration

- [ ] Expand CI/CD coverage to 70%+ of systems
- [ ] Infrastructure as Code for new deployments
- [ ] Security tooling integrated into pipelines with automated feedback
- [ ] Legacy system migration planning with risk assessment

Months 13-18: Completion

- [ ] 90%+ systems on modern infrastructure
- [ ] Automated security scanning comprehensive
- [ ] Manual deployment processes eliminated for standard changes
- [ ] Team operating with cloud-native mindset and capabilities

Success Indicators

- **Technical**: Deployment frequency increases 5-10x, infrastructure provisioning time reduced from days to minutes
- **Security**: Vulnerability detection moves from quarterly to continuous, dependency management automated
- **Cultural**: Team enthusiasm for new capabilities, reduced resistance to process changes

Common Pitfalls

> **The 'Lift and Shift' Trap**
>
> **Risk**: Moving legacy systems to cloud without architectural modernization. You get cloud bills without operational readiness benefits
>
> **Solution**: Modernize incrementally. Start with new services cloud-native, migrate legacy strategically with re-architecture where beneficial

---

3.3.5 Path 2: Craft → Mass (Reach Scaling)

> **Movement Type**: Single-axis (Increasing Reach)
>
> **Timeline**: 18-24 months
>
> **Success Probability**: Moderate

When to Choose This Path

Yes, if:

- Business growth requires rapid team scaling and organizational complexity
- Market opportunity demands enterprise features and capabilities
- Investment in operational readiness infrastructure isn't immediately feasible

No, if:

- You can afford to invest in readiness first (Craft Studio Lean is better)
- Current manual processes are already creating business constraints
- Team has capacity to focus on foundational improvements

Strategic Warning

This path creates technical debt. You're widening reach (more automation authority, more autonomous actors, broader scope across teams and features) without building the containment to match. Plan for eventual Mass Lean transformation requiring 36-60 months.

Key Investment Priorities

1. **Team Scaling**: Hiring and organizational structure for multiple teams
2. **Process Sophistication**: Coordination mechanisms, change management, incident response
3. **Compliance Capabilities**: Regulatory frameworks, audit readiness, documentation
4. **Manual Security Scale**: Security team growth to match increased reach

Success Indicators

- **Business**: Successfully serving enterprise customers, meeting regulatory requirements
- **Operational**: Multiple teams operating with coordination processes
- **Security**: Maintaining security coverage despite increased complexity

Recommended Next Steps

Plan immediately for Mass Lean transformation. Don't stay in Mass long-term. It's unsustainable.

3.3.6 Path 3: Studio → Lean (Enterprise Scaling)

> **Movement Type**: Single-axis (Increasing Reach)
>
> **Timeline**: 24-36 months
>
> **Success Probability**: High

When to Choose This Path

Optimal scenario for sustainable growth: You're widening reach while maintaining operational readiness, so containment scales with the reach. This is the high-success transformation path.

Key Investment Priorities

1. **Platform Engineering**: Self-service security capabilities for multiple teams
2. **Federated Security Model**: Security champions program with central guidance
3. **Enterprise Architecture**: Standardized patterns and reusable security components
4. **Organizational Design**: Matrix management, cross-functional coordination
5. **Cultural Scaling**: Learning culture that scales with organizational growth

Sequencing Strategy

Months 1-12: Platform Foundation

- [ ] Security platform vision and initial capabilities
- [ ] Security champions program launch in 2-3 pilot teams
- [ ] Enterprise architecture patterns documented
- [ ] Cross-team coordination mechanisms established

Months 13-24: Scaling

- [ ] Platform capabilities covering 50%+ of common security needs
- [ ] Security champions in all major teams
- [ ] Federated decision-making with clear guardrails
- [ ] Organizational structure supporting scale

Months 25-36: Optimization

- [ ] Platform capabilities comprehensive and self-service
- [ ] Security embedded in engineering culture
- [ ] Continuous improvement processes mature
- [ ] Security as competitive advantage realized

Success Indicators

- **Platform Adoption**: 70%+ of teams using self-service security capabilities
- **Developer Satisfaction**: Measurable improvement in security experience scores
- **Security Outcomes**: Capabilities scale automatically with team growth

- **Cultural**: Security champions viewed as career development opportunities

---

3.3.7 Path 4: Mass → Craft (Simplification)

> **Movement Type**: Single-axis (Reducing Reach)
>
> **Timeline**: 18-30 months
>
> **Success Probability**: Low

Strategic Reality Check

This is the hardest path and rarely succeeds. Shrinking inherent reach means clawing back authority you have already granted across the estate, and a large enterprise cannot easily shrink below its own surface area. It typically requires:

- Major business restructuring or product simplification
- Customer migration from complex to simple offerings
- Organizational downsizing or significant reorganization
- Market repositioning from enterprise to SMB or niche focus

When This Might Be Necessary

- Business pivot from enterprise to SMB market
- Divestiture or spin-off creating smaller organization
- Post-acquisition rationalization eliminating complexity
- Strategic decision to focus on core simplified offering

Why This Usually Fails

- **Customer Commitments**: Existing customers expect continued enterprise capabilities
- **Revenue Dependency**: Complex offerings often generate significant revenue
- **Organizational Resistance**: Teams resist simplification seen as "scaling back"
- **Market Perception**: Simplification can be viewed as retreat or failure

Alternative: Consider Mass → Lean Instead

Most Mass should invest in operational readiness rather than attempting to shrink reach. The Mass Lean path is difficult but more achievable than clawing back granted authority across the estate.

3.3.8 Path 5: Mass → Lean (The Containment Climb)

> **Movement Type**: Single-axis (Increasing Readiness under fixed large reach)
>
> **Timeline**: 36-60 months
>
> **Success Probability**: Moderate

When to Choose This Path

Reality: Most Mass organizations must take this path. You can't shrink reach you've already granted across the estate, so you must build containment under it.

The Challenge

You're simultaneously:

- Holding large inherent reach (broad automation authority across legacy and modern systems, enterprise requirements)
- Improving operational readiness (automation, cloud migration, technical debt reduction)

This requires sustained executive support, significant investment, and realistic timeline expectations.

Critical Success Factors

1. **Executive Sponsorship**: Sustained leadership commitment over 3-5 years
2. **Realistic Timelines**: Accept 36-60 months; don't promise 12-month miracles
3. **Hybrid Solutions**: Technology that works with legacy AND modern systems
4. **Strategic Debt Reduction**: Systematic elimination of highest-risk constraints
5. **Quick Wins**: Visible improvements that maintain organizational momentum

Sequencing Strategy

Phase 1 (Months 1-12): Stabilize and Plan

- [ ] Comprehensive assessment of current state and transformation requirements
- [ ] Hybrid security architecture supporting legacy and modern systems
- [ ] Quick wins demonstrating transformation value and building confidence
- [ ] Executive alignment on 36-60 month realistic timeline
- [ ] Transformation roadmap with clear milestones and success metrics

Phase 2 (Months 13-24): Foundation Building

- [ ] Critical technical debt reduction enabling future automation
- [ ] Modern platforms deployed alongside legacy systems
- [ ] Automation pilots in high-value areas demonstrating ROI
- [ ] Cultural initiatives building transformation momentum
- [ ] Change management reducing organizational resistance

Phase 3 (Months 25-36): Acceleration

- [ ] Significant automation coverage with measurable benefits
- [ ] Legacy system migration or modernization showing progress
- [ ] Platform capabilities emerging enabling self-service
- [ ] Organizational capability development sustaining transformation

Phase 4 (Months 37-48): Optimization

- [ ] Lean-level capabilities operational across organization
- [ ] Legacy constraints substantially eliminated or managed
- [ ] Security automation enabling business velocity
- [ ] Competitive advantage realization from transformation

Common Pitfalls

The Aggressive Timeline Trap

Risk: Promising 12-18 month transformation when 36-60 months is realistic

Symptoms: Burnout, partial implementations abandoned, organizational skepticism about security competence

Solution: Be honest about timelines. Under-promise and over-deliver. Secure executive commitment for realistic multi-year transformation.

Success Indicators

- **12 Months**: Hybrid solutions operational, transformation roadmap with executive buy-in, visible quick wins
- **24 Months**: Measurable automation improvements, improved readiness metrics, cultural momentum
- **36-60 Months**: Substantial Lean-level capabilities, security enabling rather than constraining business

---

3.3.9 Path 6: Maintaining Lean Position

Movement Type: Position Maintenance and Continuous Evolution

Timeline: Ongoing

Success Probability: High (with continued investment)

The Challenge

You're not done. The Lean position requires continuous investment to maintain as:

- Organizational complexity continues evolving
- Technology platforms change and require adaptation
- Threat landscape shifts requiring capability updates
- Competitive pressure demands ongoing innovation

Key Investment Priorities

1. **Platform Evolution**: Continuous improvement of security self-service capabilities
2. **Cultural Sustainment**: Learning culture maintenance and psychological safety preservation
3. **Innovation Integration**: Incorporating new technologies and security practices
4. **Competitive Advantage**: Translating security capabilities into market differentiation
5. **Talent Development**: Growing security and engineering capability across organization

Common Pitfall: Complacency

The 'We've Arrived' Trap

Risk: Treating Lean position as destination rather than ongoing commitment

Symptoms: Platform investment declining, manual processes creeping back, cultural erosion

Solution: Continuous investment in capabilities matching organizational evolution. Falling from Lean to Mass happens when containment can't keep pace with widening reach.

3.3.10 Movement Path Selection Guide

Decision Framework

Use these questions to select your optimal path:

1. What is your current position?

- Accurately assess using strategic positions assessment tool

2. What is your business trajectory?

- Remaining simple vs scaling complexity
- Revenue growth supporting transformation investment
- Market pressures requiring rapid change

3. What is your organizational change capacity?

- Executive sponsorship sustainability over multi-year timeline
- Team capacity for transformation alongside operational demands
- Cultural readiness for significant change

4. What are your critical constraints?

- Budget availability for transformation investment
- Technical debt severity limiting automation potential
- Regulatory requirements affecting technology choices

Recommended Paths by Situation

High-growth startup (Craft):

- Best: Craft Studio (12-18 months) Lean (24-36 months)
- Avoid: Craft Mass (creates technical debt requiring later remediation)

Established enterprise with legacy (Mass):

- Best: Mass Lean (36-60 months)
- Avoid: Attempting Mass Craft (usually fails)

Cloud-native startup scaling (Studio):

- Best: Studio Lean (24-36 months)
- Maintain: High operational readiness during reach scaling

Industry leader (Lean):

- Best: Lean Lean (continuous evolution)
- Avoid: Complacency leading to readiness degradation

3.3.11 Next Steps

1. **Confirm Your Current Position**: Use strategic positions assessment
2. **Select Your Target Path**: Based on business trajectory and change capacity
3. **Review Contextual Modifiers**: Understand how your specific constraints affect timeline
4. **Read Your Implementation Guide**: Position-specific guidance in `06-implementation/`
5. **Develop Transformation Roadmap**: Use sequencing strategies above to create your plan

4. Investment Portfolio

4.1 Investment Portfolio Framework

4.1.1 The Scaling Challenge

Picture this scenario: Your development teams want faster security reviews. Customer success needs immediate responses to security questionnaires. Compliance requires detailed audit preparations. Meanwhile, a critical vulnerability just dropped, and your CEO is asking about your incident response plan.

Sound familiar? You've hit the **scaling crisis**, the inevitable moment when demand for security services grows exponentially while your team capacity grows linearly.

> **The Inflection Point**
>
> Most security leaders respond to scaling challenges by hiring more people and working longer hours. While this provides temporary relief, it becomes increasingly difficult to sustain long-term and doesn't address the fundamental capability gap.

4.1.2 A Different Approach

Past a certain point, the solution isn't necessarily doing more security work. It's **deliberately constraining some activities while investing in capabilities that reduce future manual effort**. This section shows you how to make this strategic shift while maintaining security outcomes.

4.1.3 The BAU Scaling Crisis

The Capability Gap Reality

The BAU scaling crisis isn't a resource problem. It's a capability mismatch. When adversaries automate attacks at internet scale while defenders remain manual, no amount of hiring closes the gap.

The Structural Mismatch:

- **Manual defender processes**: Quarterly vulnerability scans, manual asset discovery, individual security reviews
- **Scaled adversary automation**: Continuous probing, automated reconnaissance, industrial-scale exploitation

This structural mismatch, manual defender processes versus scaled adversary automation, makes the crisis inevitable.

Exponential Demand Growth

As software factories grow, traditional security activities face a mathematical scaling challenge:

- **Security reviews** increase with feature velocity
- **Threat modeling** requests scale with system complexity
- **Customer security inquiries** grow with customer base expansion
- **Incident response** requirements increase with system surface area
- **Compliance activities** expand with regulatory scope

Meanwhile, security team capacity grows linearly at best. Hiring requires time and creates temporary productivity reduction during onboarding. Communication overhead increases with team size.

A word on *exponential*: that growth rests on AI capability continuing to compound, and the pre-training mechanism behind it may be slowing, with the public-text supply it runs on exhausting this decade. It matters less than it sounds. The demand here tracks how much

code ships and how much authority automation holds, both of which keep rising on deployment alone, and the comprehension gap above is already booked. A plateau buys more time, not a different answer. The BAU vs Scaling chapter works this through.

The Inflection Point

Organizations reach a point where demand for BAU security services exceeds sustainable capacity, creating constraints on both security effectiveness and business velocity.

4.1.4 Strategic Choice Point

Organizations approaching this inflection point can choose between different resource allocation strategies:

Traditional Scaling Approach

- Hire additional security personnel for manual work
- Attempt to maintain current service levels across all requests
- Build custom solutions for individual use cases
- Maintain primarily reactive security posture

Result: Temporary relief followed by recurring capacity crises

Strategic Scaling Approach

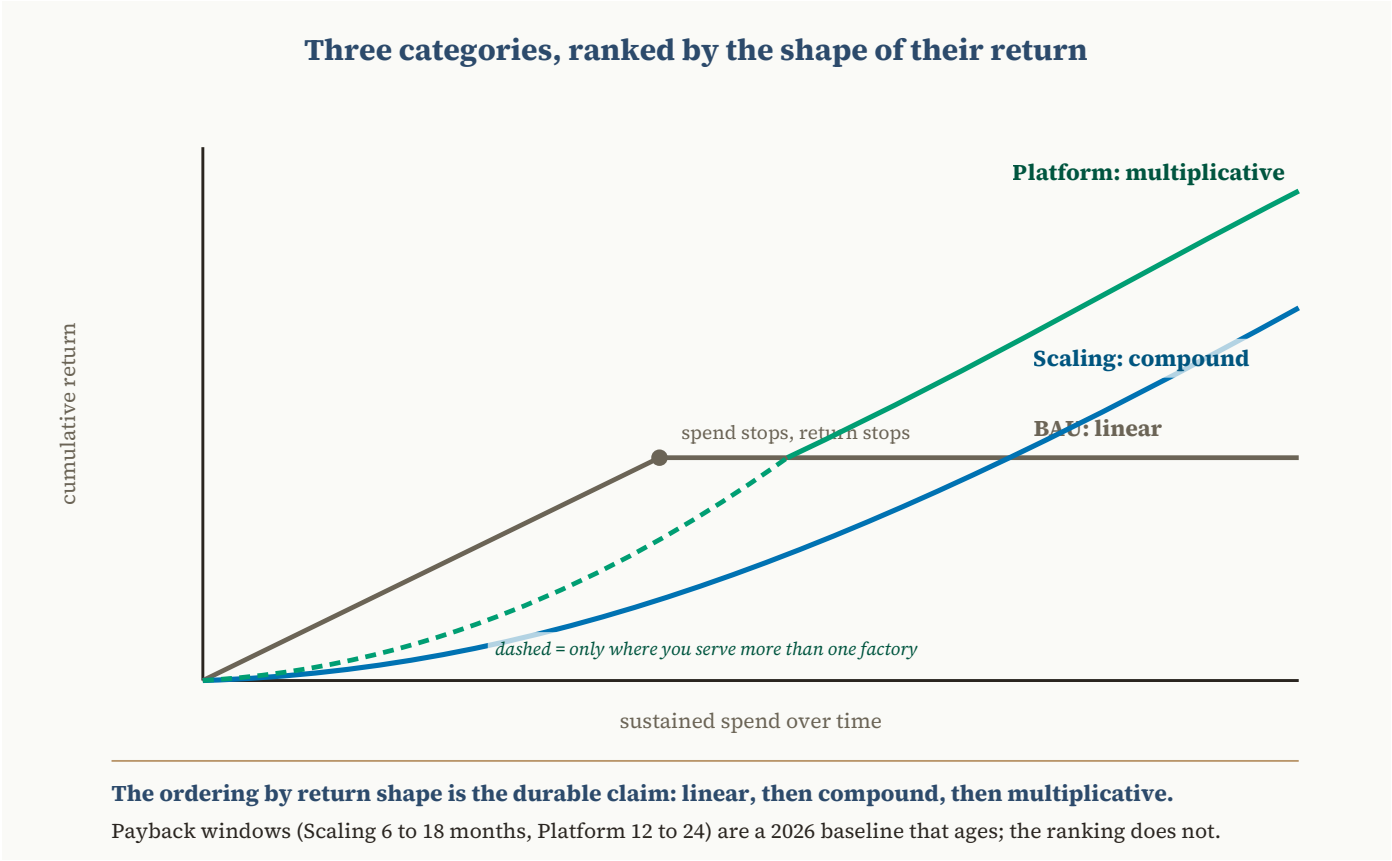
- Deliberately constrain capacity for some BAU activities
- Develop automation and self-service capabilities
- Create standardized approaches for common security needs
- Shift toward proactive, scalable security architecture

Result: Sustainable security that improves with scale

4.1.5 Investment Portfolio Categories

These three categories rank by the shape of their return. BAU is linear-effort work whose return is bounded by the hours spent and stops when the spending stops; Scaling investments keep paying after the spend ends (BAU vs Scaling works that economics in full). Platform Effects pay across more than one factory at once, but only where you have more than one factory to serve. Usually that means factories you do not run: a product your customers build on, an open-source tool the field adopts. A large enterprise running many factories of its own can reach the same multiplication internally. An organization that serves a single factory tops out at Scaling, which is the right ceiling for it. The ranking by return shape is the durable claim. Which rungs are open to you is a matter of context.

The payback windows below are a 2026 baseline, calibrated to today's tooling and a typical rate of adoption. They will age. The ordering they sit under does not move on the same clock.



The three categories ranked by return shape. BAU returns nothing once the spend stops; Scaling keeps compounding after it; Platform multiplies across more than one factory, where you have more than one to serve. The ordering is durable; the payback windows are a 2026 baseline that ages.

How fast any of this returns is set by absorption capacity: how fast the organization adopts a paved road, retires the manual work it replaces, and keeps a working understanding of what changed. Absorption has a measurable cost. DORA's 2024 State of DevOps research (https://dora.dev/research/2024/dora-report/) found that standing up a platform first cost roughly an eight percent dip in delivery throughput before the platform matured: the gain arrived after a temporary dip, paced by how fast teams absorbed the change, not by the rollout date.

One thing the return-shape ranking does not say on its own: the category it tells you to prioritize, compounding scaling investment, is also the most discretionary line on the budget, which makes it the first one a downturn cuts. The 2023 round of security budget cuts landed hardest on exactly this kind of future-facing spend. That year, HackerOne found 63 percent of organizations cut their security budgets, and 39 percent cut headcount (https://www.hackerone.com/press-release/economy-slows-headcount-and-resource-cuts-harm-security-teams-ability-combat-threats). Two instructions follow, and both are about sequencing, not reassurance. If you are funding in an up-cycle, pre-fund the compounding capability now, because it is the line that vanishes when the cycle turns. Whatever the cycle, bias the portfolio toward capability-based controls that keep enforcing with fewer hands, because that is the spend whose value does not leave with the headcount. What a cut cannot repossess is the durable enforcement that keeps running with fewer hands, which is the Lean guide's subject. This paragraph is about what to fund first. A plateau may give you more time on the demand side; a downturn gives you less on the funding side, so build for the clock that runs out first.

BAU Activities (Constrain)

Characteristics:

- Manual work that scales with growth
- Security reviews, threat modeling, incident response

- Customer security questionnaires
- Individual risk assessments

Evaluation Criteria:

- Operational necessity
- Customer impact
- Constraint sustainability

Resource Allocation: Deliberately limited capacity post-crisis

Expected ROI: Immediate but unsustainable scaling

Constraining BAU Strategically

Constraint doesn't mean abandonment. It means providing self-service alternatives, automation, and clear prioritization criteria.

Scaling Investments (Prioritize)

Characteristics:

- Capabilities that reduce manual effort exponentially
- Automation platforms, self-service capabilities, policy-as-code
- Developer security platforms
- Continuous security validation

Evaluation Criteria:

- Manual effort reduction potential
- Developer experience improvement
- Time to value
- Cultural alignment
- Organizational change requirements

Resource Allocation: Primary investment focus past crisis point

Expected ROI: 6-18 months with compound returns (2026 baseline, modeled; paced by absorption capacity, not a fixed calendar)

Scaling Investment Examples

- **Paved Roads**: Secure templates that eliminate security review needs
- **Self-Service Platforms**: Automated environments with security baked in
- **Policy-as-Code**: Automated compliance validation
- **Automated Dependency Management**: Continuous monitoring without manual effort

Platform Effects (Multiply)

Characteristics:

- Benefits both internal and customer software factories
- Security capabilities that create multiplicative value
- Open-source security tools
- Security-as-a-service offerings

Evaluation Criteria:

- Internal business case + multiplicative customer value
- Competitive differentiation
- Market amplification potential

Resource Allocation: Enhancement to scaling investments

Expected ROI: 12-24 months with market amplification (2026 baseline, modeled; same caveat as above)

4.1.6 Investment Evaluation Framework

When evaluating security investments, consider these criteria:

Criteria	Description	Why It Matters
Manual Effort Reduction	Will this eliminate repetitive work permanently?	Primary driver of sustainable scaling
Developer Experience Impact	Does this reduce security friction or create new complexity?	Critical for organizational adoption
Time to Value	How quickly will benefits become measurable?	Affects organizational confidence
Cultural Alignment	Does this support learning culture and psychological safety?	Determines long-term sustainability
Organizational Change Requirements	What adoption challenges should we anticipate?	Affects implementation success probability
Adversary Economics	Does this close the surface it claims, or only raise the cost on paths already covered?	Coverage is the test. Cost-raising counts on top of a boundary that contains the breach, not instead of it.

4.1.7 Designing Security Capabilities That Compound

The "Catch and Store" Principle

The most sustainable security investments do more than solve immediate problems. They capture organizational effort and store it in reusable capabilities that serve future needs without additional manual work.

> **Renewable Energy Analogy**
>
> Like renewable energy systems that provide ongoing value after initial investment, effective scaling investments become self-sustaining and increasingly valuable over time.

Examples of Compound Capabilities:

Paved Roads:

- Secure templates and baselines that engineers reuse without security review
- Each use provides security value without marginal security team effort
- Templates improve based on lessons learned

Self-Service Platforms:

- Automated environments and policy-as-code
- Eliminates recurring security review requests
- Scales to thousands of deployments without proportional security team growth

Automated Dependency Management:

- Continuous monitoring without manual scanning
- Automated vulnerability detection and remediation
- Improves security posture while reducing security team workload

Security-Quality Integration:

- Process improvements that serve both goals simultaneously
- Single investment, dual benefits
- Reinforcing improvements over time

Avoid Energy-Consuming Tools

Some security tools create new maintenance burdens without proportional value. Favor investments that become more valuable and less demanding over time, capabilities that store energy rather than consume it.

4.1.8 BAU Constraint Strategy by Organizational Stage

Organizational Stage	BAU Constraint Approach	Communication Strategy	Alternative Provision
Pre-Crisis	Maintain current service levels	"We're investing in better capabilities"	Gradual self-service introduction
Crisis Point	Strategic capacity limits	"We're shifting to sustainable scaling"	Clear self-service alternatives
Post-Crisis	Systematic constraint with alternatives	"Improved capabilities now available"	Comprehensive self-service platform

4.1.9 Financial Model Considerations

Security leaders must work within inherited financial constraints while building toward more strategic integration:

Cost Center Context

Reality: Security viewed as overhead to minimize

Strategy: Focus on compliance cost avoidance and operational efficiency

Communication: Emphasize business risk reduction and efficiency gains

Investment Opportunities: Crisis events create windows for scaling investment approval

Shared Services Context

Reality: Security funded through chargeback model

Strategy: Develop strong business cases emphasizing internal customer satisfaction

Communication: Highlight operational improvements that reduce business friction

Investment Opportunities: Service level improvements and efficiency gains

R&D Integration Context

Reality: Security integrated into product development budget

Strategy: Frame security investments as competitive advantages

Communication: Measure success through business outcomes rather than security-specific metrics

Investment Opportunities: Product security capabilities that differentiate in market

4.1.10 Strategic Conversation Template

When proposing the shift from traditional to strategic scaling:

Executive Communication

"We've identified that our current security approach may become a business constraint as we continue to scale. Rather than only adding capacity through hiring, which provides temporary relief, we recommend investing in capabilities that reduce manual effort requirements permanently.

This approach can improve both security outcomes and business velocity over time. The initial investment will create capabilities that compound: each use provides security value without proportional security team effort.

We'll measure success through developer satisfaction, security coverage, and time-to-market improvements, demonstrating that security enables rather than constrains business growth."

4.1.11 Implementation Roadmap

Run it in this order: assess, pilot, constrain, expand. The quarter labels are a 2026 baseline cadence. An organization with high absorption capacity may move through it in two quarters; one stabilizing heavy BAU debt may need a year. Pace the moves to what the organization can absorb, not to the calendar.

Quarter 1: Assessment and Planning

- Measure current BAU demand and capacity
- Identify scaling investment opportunities
- Assess developer pain points
- Build business case for strategic shift

Quarter 2: Pilot Scaling Investments

- Select highest-impact scaling investment
- Implement pilot with small team
- Measure manual effort reduction
- Collect developer feedback

Quarter 3: Constrain BAU + Scale Alternatives

- Introduce strategic BAU constraints
- Provide self-service alternatives
- Communicate clearly about transition

- Monitor adoption and satisfaction

Quarter 4: Iterate and Expand

- Review pilot results and iterate
- Expand successful scaling investments
- Build compound capabilities
- Demonstrate ROI to stakeholders

4.1.12 Next Steps

Explore specific aspects of the investment portfolio framework:

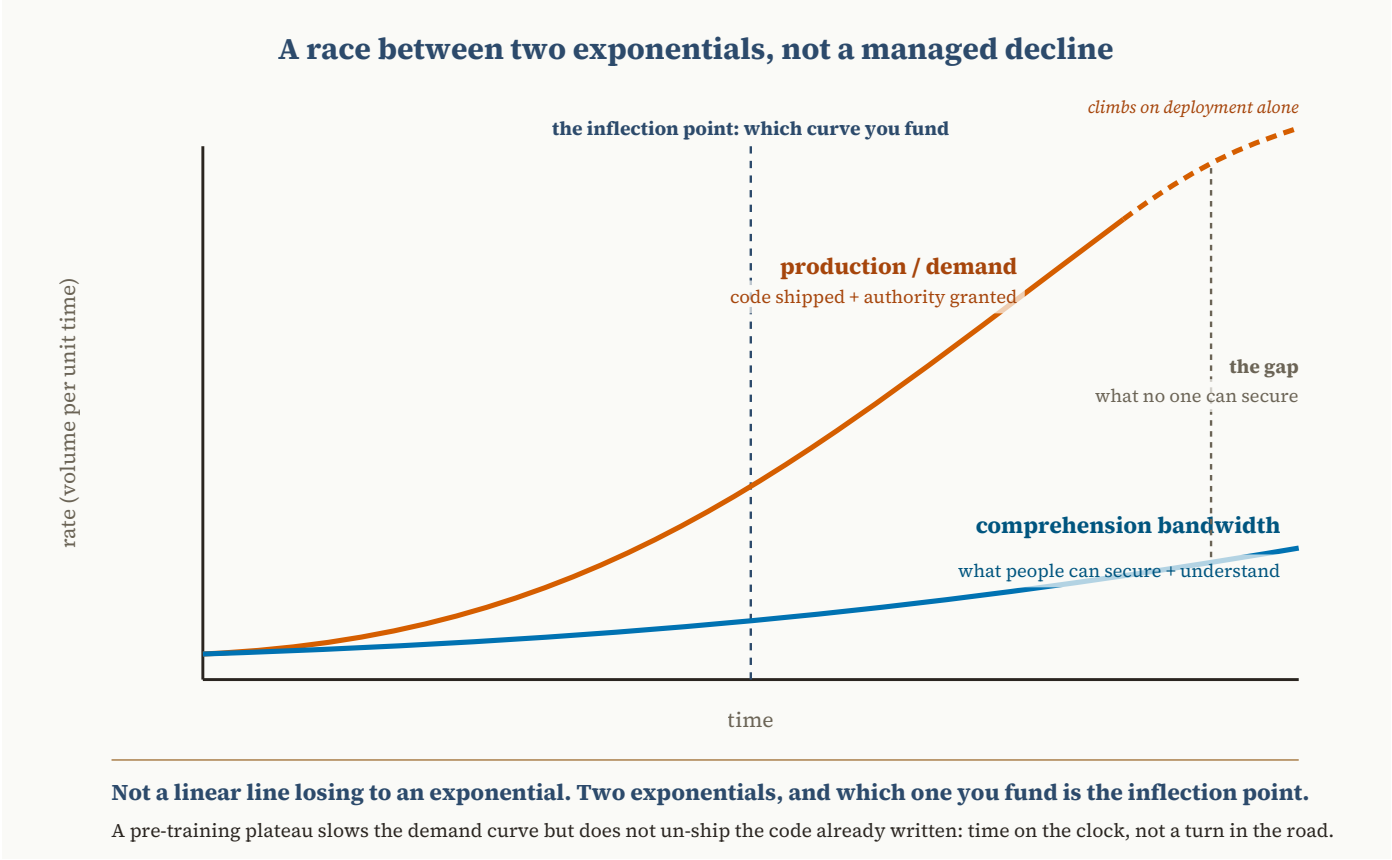
4.2 BAU vs Scaling Investments

4.2.1 The Inevitable Choice

Demand for security services grows exponentially. The instinct is to read the constraint as capacity: not enough reviewers, not enough hours. Several things do bind a scaling security program, but the one more headcount cannot buy is comprehension bandwidth, the rate at which your people can actually understand what your factory ships. This is not a structural gap where a linear line loses to an exponential one and you manage the decline. It is a race between two exponentials: how fast the factory produces, and how much of that output your people can actually secure and understand. Which curve you fund is the inflection point.

One honest caveat on the word *exponential*. The demand curve climbs as long as the factory ships more code and hands automation more authority, and that has compounded because AI capability has compounded. That engine is not guaranteed. Most people, when they say the models keep getting better, mean pre-training scaling. That mechanism is showing diminishing returns, and the public human text it trains on is being used up sometime this decade. Ilya Sutskever, who led much of that scaling, told NeurIPS in 2024 (https://analyticsindiamag.com/ai-features/the-end-of-pre-training-era-begins/) that "pre-training as we know it will end." A slowdown in that one mechanism is a real possibility, not a tail risk.

It changes less than it looks like it should, for three reasons in rising order of certainty. The least certain first: the mechanism people will point to, pre-training, may stall. Next, and surer: even if it does, the demand this framework tracks is not model cleverness; it is the volume of code shipped and the breadth of authority handed to automation, and both keep climbing on deployment alone, through test-time reasoning, agent fleets, and wider rollout, with no new scaling breakthrough required. Surest of all: the comprehension debt and the automated adversary are already here, since a plateau does not un-ship the code your people already cannot read, or call off the scanners already probing you. The destination is the same. What a plateau buys is time on the clock, not a turn in the road.



Production climbs faster than comprehension; the widening gap is the share of output no one can secure or understand. Which

curve you fund is the inflection point. A plateau slows demand
but does not change the destination.

The Traditional Response: Hire more security professionals, work longer hours, accept growing backlogs.

The Strategic Response: Deliberately constrain some activities to invest in capabilities that reduce future manual effort.

This section explains how to make this strategic shift while maintaining security outcomes.

4.2.2 The BAU Scaling Crisis

The Mathematical Reality

As software factories grow, traditional security activities face a scaling challenge that hiring alone cannot solve:

Demand Grows Exponentially:

- Security review requests increase with feature velocity (more teams = more requests)
- Threat modeling needs scale with system complexity (more services = more models)
- Customer security inquiries grow with customer base (enterprise customers demand more)
- Incident response requirements increase with system surface area (more systems = more incidents)
- Compliance activities expand with regulatory scope (new markets = new requirements)

Capacity Grows Linearly:

- Hiring requires time (3-6 months per role typically)
- Onboarding creates temporary productivity reduction (new hires require training)
- Communication overhead increases with team size (coordination costs rise)
- Maintaining quality becomes challenging during rapid scaling (standards slip under pressure)

The Inflection Point: Organizations reach a point where demand for BAU (Business-as-Usual) security services exceeds sustainable capacity, creating constraints on both security effectiveness and business velocity.

The Capability Gap: Beyond Just Capacity

Volume isn't the real problem in the BAU scaling crisis. The deeper issue is a capability mismatch between manual defenders and automated adversaries.

The Adversary Capability Shift: In recent years, adversaries evolved from targeted reconnaissance to automated discovery at internet scale. Using techniques inspired by bug bounty programs and internet-wide scanning, attackers can now:

- Discover unknown assets (forgotten servers, shadow IT, unmanaged dependencies) faster than organizations can inventory them
- Exploit known vulnerabilities within hours or days of disclosure
- Conduct credential stuffing at scale against thousands of targets simultaneously
- Probe continuously while defenders scan quarterly

One Break, Two Bottlenecks: Organizations conducting quarterly vulnerability scans face adversaries who probe continuously. Manual asset discovery can't keep pace with automated reconnaissance. Underneath both sits one break showing up in two places. On the attacker's side, automation outran a human-bound defensive process, so manual throughput loses the speed race. On the production side, automation now outruns people too: code and systems are generated faster than anyone can understand them, so manual comprehension loses the understanding race. One root cause, two human bottlenecks, neither closed by hiring alone.

Once a program has enough repeated work to amortize, the dollar that makes the secure path the easy path, by automating a step or packaging it into a guardrail engineers plug into, buys more than the dollar that adds one more linear reviewer, including the dollar that pays the engineer who builds it.

Critical Insight: Supply Chain as #1 Priority

Supply chain security became the #1 priority not because dependencies increased, but because adversary capability evolved. When attackers can discover your unknown assets faster than you can catalog them, supply chain security becomes existential regardless of your other security investments.

4.2.3 The Strategic Choice Point

At the scaling inflection point, organizations face two fundamentally different resource allocation strategies:

Traditional Scaling Approach (Unsustainable)

Strategy: Attempt to maintain current service levels through capacity increases

Typical Actions:

- Hire additional security personnel for manual work
- Extend working hours to cover growing demand
- Build custom solutions for individual use cases
- Maintain primarily reactive security posture
- Accept growing backlogs as "normal"

Why This Fails:

- Hiring doesn't close the capability gap against automated adversaries
- Linear capacity growth can't match exponential demand growth
- Custom solutions create maintenance burden without scaling benefits
- Team burnout and quality degradation become inevitable
- Security becomes a business constraint and bottleneck

Strategic Scaling Approach (Sustainable)

Strategy: Deliberately constrain capacity for some BAU activities to create investment cycles for automation and self-service capabilities

Typical Actions:

- Set explicit capacity limits for manual security activities
- Develop automation and self-service capabilities systematically
- Create standardized approaches for common security needs
- Shift toward proactive, scalable security architecture
- Measure and communicate ROI from scaling investments

Why This Succeeds:

- Automation closes the capability gap against scaled adversaries
- Capabilities create compound returns over time
- Self-service enables teams without security team growth

- Developer experience improves rather than degrades
- Security becomes a competitive advantage enabler

The Compound Interest Principle

Just as financial investments generate compound returns, security scaling investments create exponential value. An automation capability used 100 times costs the same to build as one used once, but delivers 100x the value. Manual security work scales linearly. Each review costs the same effort.

4.2.4 Investment Portfolio Categories

Security investments fall into three categories with fundamentally different scaling characteristics:

BAU Activities (Constrain Past Crisis Point)

Definition: Manual work that scales linearly with organizational growth

Examples:

- Manual security design reviews for each new service
- Threat modeling sessions requiring security team participation
- Individual incident response investigations
- Customer security questionnaire responses
- Ad-hoc compliance evidence collection

Characteristics:

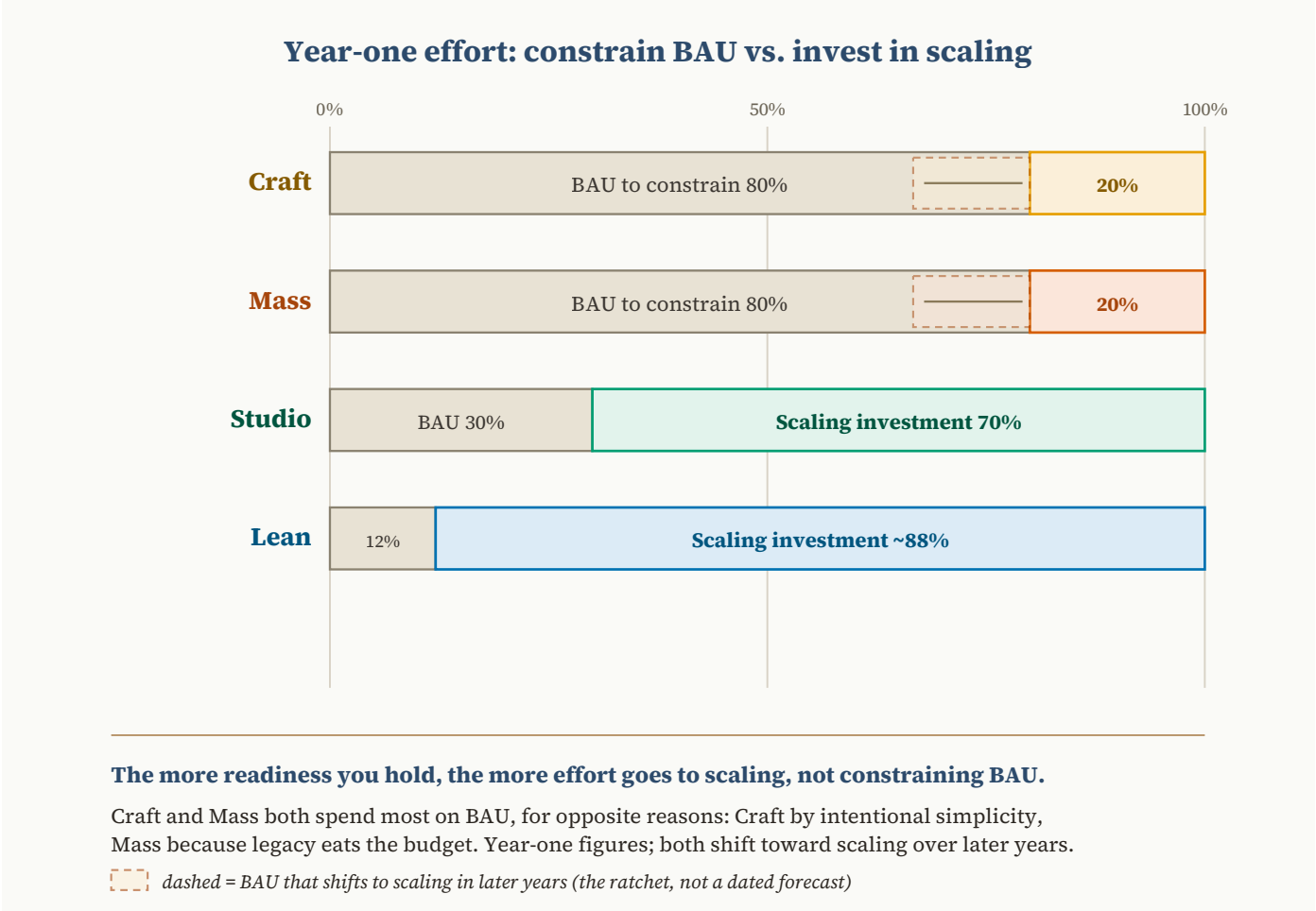
- Required for business operations (can't eliminate entirely)
- Demand grows with organizational scale
- Each instance requires similar effort (limited efficiency gains)
- Creates capacity constraints at scale

Strategic Approach:

- **Pre-Crisis**: Maintain current service levels while building alternatives
- **Crisis Point**: Set explicit capacity constraints with communication strategy
- **Post-Crisis**: Systematic constraint with clear self-service alternatives

Constraint Strategy by Position:

Position	BAU Approach	Constraint Mechanism
Studio	Minimal BAU burden initially	Automation-first, avoid creating manual processes
Lean	Systematic constraint with alternatives	Self-service platform + clear escalation paths
Craft	Manageable manual processes	Document processes while building readiness
Mass	High burden requiring constraint	Triage system + strategic automation pilots



Year-one effort split per quadrant. The more readiness you hold, the more goes to scaling rather than constraining BAU. Craft and Mass spend most on BAU for opposite reasons: Craft by intentional simplicity, Mass because legacy eats the budget.

Scaling Investments (Prioritize Post-Crisis)

Definition: Capabilities that reduce manual effort exponentially or enable self-service

Scaling Investments compound in two different ways, and the difference decides what each one buys. **Automation** takes a human out of a repeated step, so the work runs hands-free and throughput stops being bound by hours. **Composable guardrails** keep the human but change what they have to understand: instead of reviewing N bespoke implementations of mTLS, base images, scoped roles, or secure pub/sub, your people understand one packaged path that teams plug into. The comprehension does not disappear. It concentrates into one durable review, amortized across every team that adopts the path, which is a real economy only on the traffic that actually takes it. A guardrail also does what automation does not: because it is a boundary, it limits what goes wrong when something slips through, not just what your reviewers have to read. Two benefits from one artifact.

Examples:

- Automated dependency scanning with auto-remediation
- Self-service security environment provisioning
- Policy-as-code with automated enforcement
- Security champions program enabling distributed expertise

• Developer security training and tooling integration

Characteristics:

• Initial investment required (time, money, organizational change)

• Benefits compound over time (more usage = more value)

• Reduces future manual effort requirements systematically

• Enables organizational scaling without proportional security team growth

Evaluation Criteria:

Criterion	Why It Matters	Assessment Question
Manual Effort Reduction	Primary driver of sustainable scaling	Will this eliminate repetitive work permanently?
Developer Experience	Critical for adoption	Does this reduce security friction or create new complexity?
Time to Value	Affects organizational confidence	How quickly will benefits become measurable?
Cultural Alignment	Determines sustainability	Does this support learning culture and psychological safety?
Adversary Economics	Real, but judged by coverage	Does this close a path, or only raise the price on paths already closed?

Expected ROI Timeline: 6-18 months with compound returns increasing over time

---

Platform Effects (Multiply)

Definition: Investments that benefit both your organization AND customer software factories (platform companies only)

Examples:

• Security platform features customers can use

• Open-source security tools serving broader ecosystem

• Security standards that become industry practices

• Shared threat intelligence benefiting community

Characteristics:

• Internal business case must justify investment independently

• Customer value creates additional strategic benefits

• Competitive differentiation potential

• Market influence and thought leadership opportunities

Evaluation: Internal benefit × customer multiplier + competitive advantage

See Also: Platform Effects for detailed guidance for platform companies

---

4.2.5 Designing Security Capabilities That Compound

The "Catch and Store Energy" Principle

The most sustainable security investments do more than solve immediate problems. They **capture organizational effort and store it in reusable capabilities** that serve future needs without additional manual work.

Like renewable energy systems that provide ongoing value after initial investment, effective scaling investments become self-sustaining:

Paved Roads: Secure templates and baselines that engineers reuse without security review

- **One-time effort**: Design secure baseline architecture, document patterns
- **Ongoing value**: Every team using the template saves 10-20 hours of security reviews
- **Compound effect**: As templates improve based on feedback, all users benefit automatically

Self-Service Platforms: Automated environments and policy-as-code eliminating recurring requests

- **One-time effort**: Build security environment provisioning automation
- **Ongoing value**: Teams provision secure infrastructure in minutes instead of days
- **Compound effect**: Platform improvements benefit all users without additional security team effort

Automated Dependency Management: Continuous monitoring without manual intervention

- **One-time effort**: Integrate automated dependency scanning and update tools
- **Ongoing value**: Vulnerabilities detected and remediated automatically
- **Compound effect**: Coverage expands automatically as new services are built

Security-Quality Integration: Process improvements serving both objectives simultaneously

- **One-time effort**: Integrate security checks into CI/CD quality gates
- **Ongoing value**: Security and quality issues detected together in development
- **Compound effect**: Quality improvements enhance security, security improvements enhance quality

---

What to Avoid: Security Tools That Consume Energy

The Maintenance Burden Trap

Some security tools create ongoing maintenance costs that exceed their security value. Avoid investments that:

- Require continuous manual tuning to remain effective
- Generate high false-positive rates demanding constant triage
- Need specialized expertise that creates key-person dependencies
- Don't integrate with existing development workflows
- Create new manual processes rather than automating existing ones

Favor investments that:

- Become more valuable and less demanding over time
- Store organizational knowledge in reusable form
- Enable self-service without security team involvement
- Integrate seamlessly into existing workflows
- Improve developer experience while improving security

4.2.6 BAU Constraint Communication Strategy

Constraining BAU activities requires clear communication to maintain organizational support and developer relationships.

Communication by Organizational Stage

Pre-Crisis (Building Alternatives):

Message: "We're investing in improved capabilities that will provide faster, more consistent security support."

Actions:

- Maintain current service levels while building alternatives
- Gradual introduction of self-service options
- Measure baseline metrics for later comparison
- Build organizational confidence in new approaches

Crisis Point (Implementing Constraints):

Message: "We've reached a scaling inflection point. To ensure sustainable security support, we're shifting from manual processes to self-service capabilities. Here's what's changing, here are the alternatives, and here's the timeline for improved capabilities."

Actions:

- Set explicit capacity limits with clear justification
- Provide immediate self-service alternatives (even if basic)
- Establish escalation paths for critical needs
- Regular updates on scaling investment progress

Example Communication:

"Our security review process has reached capacity constraints. Starting next quarter, we're implementing a self-service security baseline that will enable most teams to deploy securely without security review wait times. For projects outside this baseline, we'll use a triage process prioritizing business-critical initiatives. We expect this transition to take 6 months, after which your experience will significantly improve."

Post-Crisis (Systematic Operations):

Message: "Improved self-service capabilities are now available. Most teams can now [specific capability] without security team involvement, and we've measured [specific improvement metric]."

Actions:

- Demonstrate ROI realization from scaling investments
- Showcase developer experience improvements
- Adjust capacity constraints based on capability maturity
- Continuous improvement of self-service platforms

4.2.7 Financial Model Considerations

Security leaders operate within inherited financial contexts that affect investment strategies:

Cost Center Context

Constraints: Security viewed as overhead, budget scrutiny, ROI skepticism

Strategy:

- Focus on compliance cost avoidance (failed audits cost money)
- Emphasize operational efficiency (automation reduces labor costs)
- Use crisis events as opportunities for scaling investment approval
- Frame investments as risk reduction with quantifiable business impact

Conversation Template:

"Our current security approach will become a business constraint as we scale. Rather than only adding capacity through hiring, which provides temporary relief at increasing cost, we recommend investing in capabilities that reduce manual effort requirements permanently. This approach improves both security outcomes and business velocity while managing cost growth."

Shared Services Context

Constraints: Internal customer satisfaction, operational metrics, service level expectations

Strategy:

- Develop business cases emphasizing internal customer satisfaction
- Measure and communicate developer experience improvements
- Show operational improvements reducing business friction
- Position security as enabling faster, safer delivery

Conversation Template:

"Our internal customers currently wait an average of 5 days for security reviews. By investing in self-service security baselines, we can reduce this to under 1 hour for 70% of projects while improving security consistency. This enables faster delivery without compromising security outcomes."

R&D Integration Context

Constraints: Security competes with feature development for resources and attention

Strategy:

- Frame security investments as competitive advantages
- Measure success through business outcomes, not security metrics
- Demonstrate security capabilities enabling business opportunities
- Position security as accelerating time-to-market for secure products

Conversation Template:

"Security automation does more than manage risk. It lets our engineers move faster with confidence. By investing in security platforms, we can support 3x growth without proportional security team expansion while improving both security posture and delivery velocity."

4.2.8 Success Metrics for Investment Shifts

Track these metrics to validate your BAU constraint and scaling investment strategy:

Leading Indicators (Early Signals)

- **Scaling investment velocity**: Projects started, adoption rates, developer feedback
- **Alternative capability usage**: Self-service adoption rates, platform utilization
- **Developer satisfaction trends**: Survey scores, friction reports, voluntary participation
- **Investment pipeline health**: Approved projects, executive support, resource allocation

Lagging Indicators (Results)

- **Manual effort reduction**: Hours saved per activity type, capacity freed for strategic work
- **ROI realization**: Measurable benefits vs investment costs, compound return evidence
- **Security outcomes**: Vulnerability detection rates, incident response times, risk posture improvements
- **Business velocity**: Time-to-market improvements, deployment frequency increases, developer productivity gains

---

4.2.9 Next Steps

1. **Assess Your Position**: Are you pre-crisis, at crisis point, or post-crisis in your scaling journey?
2. **Evaluate Current Portfolio**: Catalog BAU activities and identify scaling investment opportunities
3. **Review Evaluation Criteria**: Systematic framework for prioritizing investments
4. **Understand Platform Effects**: Additional considerations for platform companies
5. **Develop Communication Strategy**: Prepare stakeholder messaging for investment shifts

---

4.3 Platform Effects

4.3.1 Multiplicative Value for Platform Companies

Platform companies, organizations that provide capabilities to other software factories, have unique opportunities to create value that extends beyond their immediate organization. However, these opportunities come with important constraints and strategic considerations.

Critical Principle: Platform effects should serve as investment multipliers, not primary drivers. The internal business case must work first.

4.3.2 Who This Section Is For

This guidance applies specifically to:

Platform Companies: Organizations whose products serve other software factories

- Cloud platforms (AWS, Azure, GCP, etc.)
- Development tools and CI/CD platforms (GitLab, GitHub, CircleCI, etc.)
- Security platforms and tools (vulnerability scanners, SIEM, etc.)
- Developer productivity platforms (observability, monitoring, etc.)
- Infrastructure and container platforms (Kubernetes, Docker, etc.)

Not applicable to:

- End-user application companies (even large ones)
- Internal platform teams serving only their own organization
- Security vendors selling traditional enterprise software

If you're not a platform company serving other software factories, focus on BAU vs Scaling and Evaluation Criteria instead.

4.3.3 The Platform Effects Principle

Primary Responsibility: Solve Your Own Problems First

Non-Negotiable Foundation: Security investments must first make business sense for your direct needs. Platform effects are **enhancement factors**, not justifications.

Why This Matters:

- Your security team's primary responsibility is securing your software factory
- Platform features that don't solve your internal problems create maintenance burden
- Customer value without internal value is unsustainable
- Failed internal implementations can't become successful external products

> **The Platform Distraction Trap**
>
> **Risk**: Building security features for customers that your own organization doesn't use or value
>
> **Symptoms**: Platform features your internal teams avoid, customer requests driving roadmap over internal needs, security theater for market positioning
>
> **Solution**: Eat your own dog food. If your security team won't use it internally, don't build it for customers.

Multiplicative Opportunity: When Internal and Customer Value Align

The Strategic Multiplier: When security investments that solve your internal problems also benefit customer software factories, this creates additional value justifying:

- Higher investment levels than internal-only business case supports
- Faster implementation timelines with cross-functional priority
- Different prioritization relative to competing initiatives
- Market differentiation and competitive advantage opportunities

Example: Automated dependency scanning

- **Internal value**: Reduces your security team's manual effort, improves your supply chain security posture
- **Customer value**: Enables your customers to secure their supply chains using your platform
- **Platform effect**: 10,000 customers using your feature = 10,000x the security value created
- **Competitive advantage**: Security capability differentiating your platform from competitors

---

4.3.4 Platform Investment Prioritization Framework

Use this systematic framework to evaluate security investments when you're a platform company:

Evaluation Dimensions

Dimension	Assessment Criteria	Weight	Strategic Implication
Direct Security Benefit	Does this solve our internal security problems?	Primary	Business case foundation, must pass threshold independently
Multiplicative Customer Value	Does this create additional customer value?	Enhancement	Multiplier on internal business case, breaks ties between competing investments
Competitive Differentiation	Does this create defensible advantages?	Strategic	Market positioning, pricing power, customer acquisition/retention
Industry Influence	Does this position us as thought leaders?	Market	Long-term market development, standard adoption, ecosystem benefits

---

Decision Matrix

Strong Internal + Strong Customer Value (Highest Priority):

- Example: Automated vulnerability management solving your supply chain problems that customers can use
- Priority: Accelerate implementation, allocate cross-functional resources, market aggressively
- Timeline Impact: May justify 30-50% faster implementation than internal-only business case
- Investment Level: Can justify 2-3x higher investment than internal-only features

Strong Internal + Weak Customer Value (Standard Priority):

- Example: Internal compliance automation specific to your regulatory environment

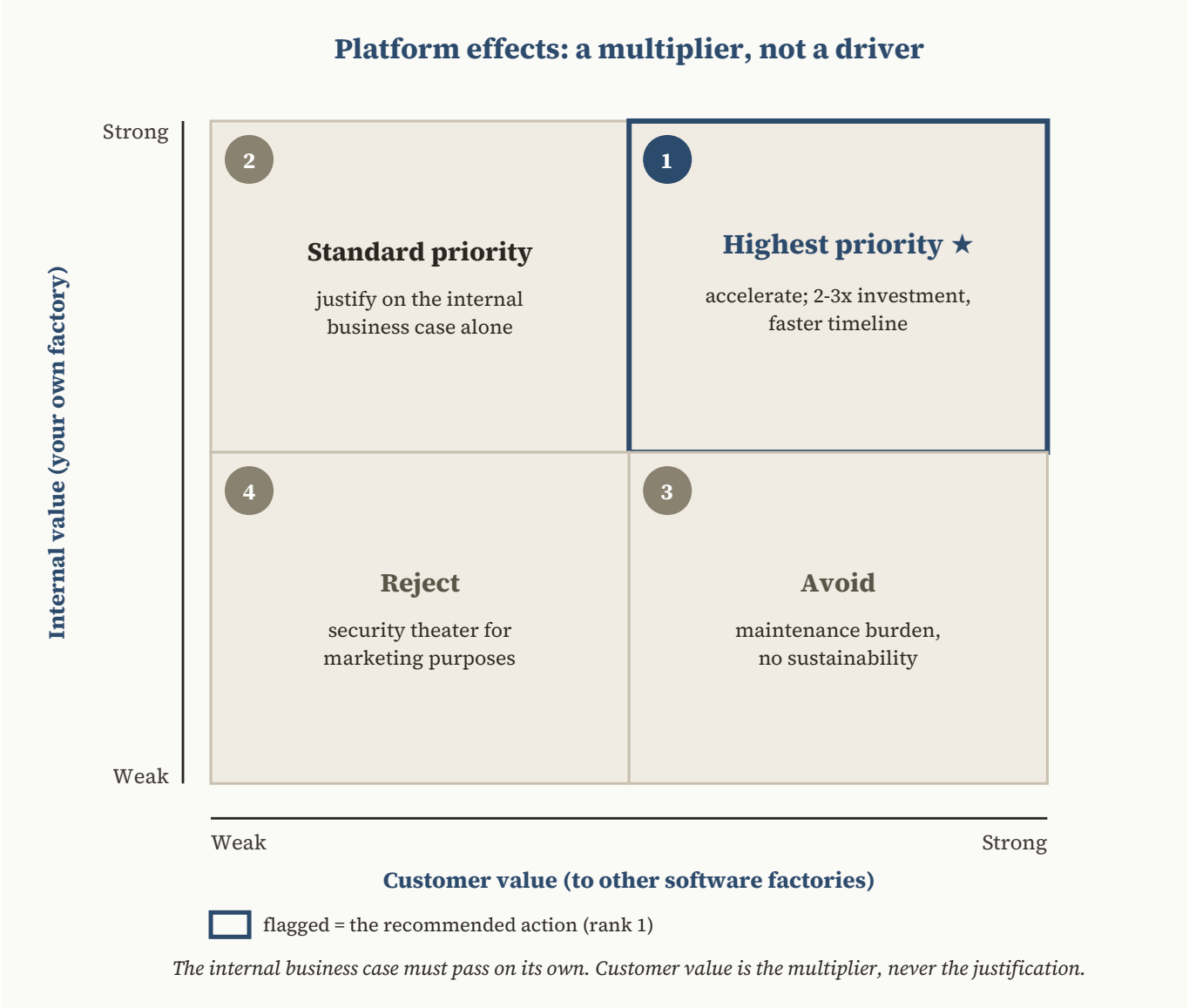
- Priority: Standard internal security investment evaluation using evaluation criteria
- Timeline Impact: Normal implementation sequencing
- Investment Level: Internal business case justification required

Weak Internal + Strong Customer Value (Avoid):

- Example: Security features customers request but your team doesn't use
- Priority: Decline or deprioritize, likely maintenance burden without sustainability
- Warning: High risk of poor execution, customer disappointment, wasted investment

Weak Internal + Weak Customer Value (Reject):

- Example: Security theater features for marketing purposes
- Priority: Hard reject, diverts resources from valuable work
- Risk: Damages credibility with both internal teams and customers



Priority is carried by the rank numerals and labels, not color. The internal business case must pass on its own; customer value is the multiplier, never the justification.

4.3.5 Customer Zero Insights and Strategic Weighting

The "Customer Zero" Advantage

Platform companies often serve as sophisticated users of their own products, providing unique insights into security challenges and solutions. Your internal security pain often represents market opportunities.

Adversary Evolution Insights: When your platform company experiences security challenges that mirror broader industry shifts, these Customer Zero insights carry additional strategic weight.

Example: Modern Supply Chain Security

Your platform company discovers that:

- Automated dependency scanning significantly reduces your manual security effort
- Your security team's productivity improves 3x with automated vulnerability management
- Unknown dependency discovery becomes systematic rather than reactive

Strategic Insight: If this internal pain point reflects modern adversary capabilities (automated discovery at scale), it likely represents a market opportunity affecting thousands of customer software factories facing the same capability gap.

Customer Zero Cost Impact Assessment

When evaluating internal security pain points, weight them by total organizational impact including the Customer Zero amplification factor:

Cost Category	Internal Cost Assessment	Customer Zero Multiplier	Strategic Priority
Direct Incident Costs	Breach response, system downtime, recovery efforts	1x (internal impact)	High
Security Research/ Bug Bounty	External researcher rewards, program costs	2x (customer trust impact)	Medium
Vulnerability Management	Discovery, assessment, patching cycles	1.5x (operational efficiency)	High
Business Disruption	Customer communication, support escalation, sales impact	3x (customer experience amplification)	High
Engineering Cycles	Feature development diverted to security firefighting	2x (opportunity cost amplification)	Medium
Customer Trust Erosion	Churn risk, sales pipeline impact, competitive disadvantage	5x (platform company trust multiplier)	Critical

Strategic Application: If a security issue category costs your organization $5M annually across these factors, and Customer Zero insights suggest this affects thousands of customers similarly, the platform investment justification becomes compelling:

- Internal problem: $5M annual cost
- Customer impact: Thousands of factories facing similar issues
- Platform solution value: Addresses market need while solving internal problem
- Competitive differentiation: Security capability competitors may lack

4.3.6 Implementation Prioritization for Platform Companies

When you've identified a security investment with both internal and customer value, use this sequencing approach:

Phase 1: Internal Implementation and Validation (Critical)

Objective: Solve your own problem first, learn from real usage

Activities:

- Implement security capability solving your internal team's pain
- Use internally with realistic production workloads
- Measure actual impact on your security team's effectiveness
- Iterate based on internal feedback and usage patterns

Success Criteria:

- Your security team actively uses and values the capability
- Measurable improvement in internal security metrics
- Positive feedback from internal stakeholders
- Sustainable operational model established

Timeline: 3-6 months typically

Phase 2: Customer Zero Learnings Integration

Objective: Refine based on internal experience before customer exposure

Activities:

- Document lessons learned from internal implementation
- Identify operational challenges and edge cases discovered
- Develop customer-facing documentation based on internal experience
- Create support and troubleshooting guidance from real issues

Success Criteria:

- Clear understanding of capability limitations and tradeoffs
- Documented best practices from internal usage
- Realistic customer expectations established
- Support burden manageable based on internal experience

Timeline: 1-2 months typically

Phase 3: Limited Customer Beta (If Customer Value Validated)

Objective: Validate customer value hypothesis with friendly customers

Activities:

- Select 3-5 sophisticated customers for beta program
- Provide hands-on support during initial adoption

- Measure customer outcomes and gather detailed feedback
- Validate that customer experience matches your internal experience

Success Criteria:

- Customers achieve measurable security improvements
- Feedback validates value hypothesis
- Support burden matches predictions
- Customers actively use the capability rather than only enabling it

Timeline: 3-6 months typically

Phase 4: General Availability (Only If Beta Succeeds)

Objective: Scale capability to broader customer base

Activities:

- Polish customer experience based on beta feedback
- Scale support infrastructure for broader adoption
- Market capability emphasizing customer outcomes
- Monitor adoption and iterate based on usage patterns

Success Criteria:

- Adoption rates meet targets
- Customer satisfaction scores strong
- Support burden sustainable
- Competitive differentiation realized

4.3.7 Security as Competitive Advantage

Market Differentiation Questions

Evaluate whether security investments create defensible competitive advantages:

1. Does this capability create defensible competitive advantages?

- Is this difficult for competitors to replicate?
- Does this build on unique platform strengths?
- Can this become customer-facing product value driving adoption?

2. Does this enable market expansion?

- Does improved security enable entry into regulated industries?
- Can this support enterprise customer requirements?
- Does this address blockers in sales pipeline?

3. How does this compare to competitor security offerings?

- Do competitors offer similar capabilities?
- Is your implementation meaningfully better?
- Can customers easily switch to competitors?

4. What is the thought leadership opportunity?

- Can this position you as security innovation leader?
- Does this enable industry standard or best practice development?
- Will this attract top security talent to your organization?

Communication Strategy for Platform Companies

Critical Balance: Platform companies must clearly communicate that platform security capabilities serve dual purposes, protecting the platform while enabling customer security, without creating unrealistic expectations about security responsibility transfer.

> **The Shared Responsibility Clarity Requirement**
>
> Platform security features enhance customer capabilities but don't transfer security accountability. Customers remain responsible for their security decisions and implementations.

Good Communication Example:

"Our automated dependency scanning capability protects our platform and is available to help you secure your software factory. This tool provides vulnerability detection and remediation guidance, but you remain responsible for reviewing findings and making security decisions appropriate to your risk tolerance and operational context."

Bad Communication Example (Avoid):

"Our platform handles your security, so you don't need to worry about vulnerabilities." [Implies accountability transfer]

4.3.8 Platform Effect Measurement

Track these metrics to validate platform effects investment strategy:

Internal Metrics (Primary)

- **Internal security improvement**: Your team's security posture improvements
- **Internal operational efficiency**: Reduced manual effort, faster incident response
- **Internal developer satisfaction**: Your engineers' experience with security capabilities

Customer Metrics (Secondary)

- **Customer adoption rates**: Percentage of customers using platform security capabilities
- **Customer security outcomes**: Measurable improvements in customer security postures
- **Customer satisfaction scores**: NPS or similar for security features
- **Customer retention impact**: Reduced churn attributable to security capabilities

Competitive Metrics (Strategic)

- **Market differentiation**: Customer acquisition attributable to security capabilities
- **Thought leadership indicators**: Conference talks, standards adoption, media coverage
- **Talent attraction**: Security engineer recruiting success, employer brand strength

---

4.3.9 Common Platform Company Pitfalls

Pitfall 1: Building for Customers Before Internal Validation

Symptom: Platform features your own security team doesn't use

Consequence: Poor quality, unsustainable support burden, customer disappointment

Solution: Always implement internally first, validate value, then consider customer availability

---

Pitfall 2: Overselling Security Responsibility Transfer

Symptom: Marketing language implying customers can "let platform handle security"

Consequence: Customer misunderstanding, blame when security issues occur, legal/liability concerns

Solution: Clear shared responsibility communication, explicit customer accountability

---

Pitfall 3: Platform Effect as Primary Driver

Symptom: Investments justified by customer value without internal business case

Consequence: Unsustainable features, internal team resistance, maintenance burden

Solution: Require internal business case independence, use customer value as multiplier only

---

Pitfall 4: Ignoring Customer Zero Insights

Symptom: Internal security pain dismissed as "just our problem"

Consequence: Missed market opportunities, competitive disadvantage

Solution: Systematic assessment whether internal pain represents broader market need

---

4.3.10 Next Steps

1. **Assess Platform Company Status**: Confirm whether platform effects guidance applies to your organization
2. **Evaluate Current Investments**: Review security investments through platform effects lens
3. **Prioritize Internal Value**: Ensure internal business cases are solid before considering customer value
4. **Review Evaluation Criteria**: Systematic framework works for all security investments
5. **Consider Customer Zero Insights**: Identify internal pain representing market opportunities

---

4.4 Investment Evaluation Criteria

4.4.1 Systematic Framework for Prioritizing Security Investments

Security leaders face overwhelming numbers of potential investments: automation projects, tooling purchases, process improvements, platform capabilities. Without systematic evaluation criteria, investment decisions become reactive or driven by whoever shouts loudest.

This framework provides objective criteria for evaluating and prioritizing scaling investments.

4.4.2 When to Use This Framework

Appropriate for:

- Scaling investments (automation, self-service, platform capabilities)
- Tool and technology selection decisions
- Process improvement prioritization
- Resource allocation across competing security initiatives

Not designed for:

- Emergency incident response (requires immediate action)
- Regulatory compliance requirements (non-discretionary)
- Executive-mandated initiatives (political reality)
- BAU activity triage (different evaluation model)

4.4.3 The Six Evaluation Criteria

Use these criteria to score and compare potential security investments systematically:

1. Manual Effort Reduction

The Question: Will this eliminate repetitive work permanently?

Why Primary: Manual effort reduction is the fundamental driver of sustainable scaling. Investments that don't reduce manual work don't solve the scaling crisis. They may improve security outcomes but won't enable organizational growth without proportional security team expansion.

Hours saved is a proxy for value, not value itself. This criterion counts the work an investment displaces, never the work it creates downstream: the generated code someone still has to understand, the dependencies it adds, the authority it now holds, the maintenance it needs. Credit a high score here only net of that cost. The risk is structural, not careless. Once hours saved carries a 2x weight it becomes a target, and a measure under target pressure stops tracking the value it once stood for (https://en.wikipedia.org/wiki/Goodhart-27s_law). An investment that saves hours while widening what can go wrong has not reduced effort. It has moved the effort somewhere this number cannot see.

Assessment Framework:

Score	Manual Effort Reduction	Annual Hours Saved	Example
5 - Exceptional	Eliminates entire category of manual work	2,000+ hours annually	Automated dependency scanning replacing manual reviews

Score	Manual Effort Reduction	Annual Hours Saved	Example
4 - High	Reduces 70-90% of manual effort in domain	1,000-2,000 hours	Self-service environment provisioning
3 - Moderate	Reduces 40-70% of manual effort	500-1,000 hours	Automated security questionnaire responses
2 - Low	Reduces 10-40% of manual effort	100-500 hours	Partial process automation
1 - Minimal	Reduces <10% of manual effort	<100 hours	Security tool with high manual overhead

Assessment Questions:

- What manual security work does this eliminate completely?
- How many person-hours per month does this category consume currently?
- Will benefits compound as the organization scales?
- Does this create capacity for strategic work, or just shift manual effort?
- What new authority, attack surface, or maintenance burden does this automation create, and is that cost already netted in your Adversary Economics score?

Red Flags:

- "This tool will help us work more efficiently" (without specific hour reduction)
- Automation that requires extensive manual tuning or maintenance
- Solutions that create new categories of manual work

---

2. Developer Experience Impact

The Question: Does this reduce security friction or create new complexity?

Why Critical: Scaling investments require developer adoption to deliver value. Security capabilities that degrade developer experience face resistance, workarounds, or abandonment regardless of security benefits. Developer experience is not a "nice to have." It determines adoption success.

Assessment Framework:

Score	Developer Experience	Time Impact	Adoption Likelihood
5 - Exceptional	Dramatically improves developer workflow	Saves developers 5+ hours/week	Enthusiastic adoption, voluntary expansion
4 - High	Noticeably improves workflow	Saves 2-5 hours/week	Willing adoption, positive feedback
3 - Neutral	Minimal workflow change	±30 minutes/week	Passive acceptance, follow guidelines
2 - Moderate Friction	Some workflow disruption	Costs 1-2 hours/week	Resistance, some workarounds

Score	Developer Experience	Time Impact	Adoption Likelihood
1 - High Friction	Significant workflow disruption	Costs 5+ hours/week	Active resistance, widespread workarounds

Assessment Questions:

- Does this integrate seamlessly into existing development workflows?
- How much additional time or effort does this require from developers?
- Do developers see personal benefit, or only security team benefit?
- What happens when developers encounter edge cases or issues?
- Is this faster/easier than current alternatives (including workarounds)?

Developer Experience Testing:

- Pilot with friendly team before broad rollout
- Measure time-to-first-value (how long until developers see benefits)
- Track adoption voluntarily vs via mandate
- Monitor for workaround creation or process avoidance

---

3. Time to Value

The Question: How quickly will benefits become measurable?

Why Important: Time to value affects organizational confidence in scaling investment strategy. Long-running projects with delayed benefits create skepticism, while quick wins build momentum and executive support. This doesn't mean always choosing fastest options, but timeline awareness is critical for communication and sequencing.

Assessment Framework:

Score	Time to Value	Measurable Benefits	Organizational Impact
5 - Immediate	<3 months to measurable impact	Quick win visible to stakeholders	Builds confidence quickly, enables next investments
4 - Fast	3-6 months to measurable impact	Benefits emerging, metrics improving	Sustains momentum, validates strategy
3 - Moderate	6-12 months to measurable impact	Significant effort before payoff	Requires sustained commitment, regular updates
2 - Slow	12-18 months to measurable impact	Long investment before returns	Requires strong executive support, milestone communication
1 - Delayed	18+ months to measurable impact	Benefits uncertain or far future	High risk of abandonment, requires exceptional justification

Assessment Questions:

- What is minimum viable implementation that delivers measurable value?
- Can we phase this to deliver incremental benefits?
- What metrics will demonstrate value, and when can we measure them?
- How do we communicate progress during implementation?

Sequencing Strategy:

- Start investment portfolio with high time-to-value projects (score 4-5)
- Build organizational confidence before tackling longer-term investments
- Mix quick wins with strategic long-term capabilities
- Communicate realistic timelines; don't over-promise to get approval

---

4. Cultural Alignment

The Question: Does this support learning culture and psychological safety?

Why Critical: Security investments succeed or fail based on organizational culture. Capabilities that punish developers, create blame dynamics, or undermine psychological safety will be resisted or circumvented regardless of security benefits. Cultural alignment isn't "soft." It's a hard requirement for adoption success.

Assessment Framework:

Score	Cultural Alignment	Psychological Safety Impact	Learning Culture Support
5 - Exceptional	Actively builds learning culture	Enhances safety, enables experimentation	Facilitates continuous improvement, knowledge sharing
4 - High	Supports existing culture	Maintains safety, no blame creation	Enables learning opportunities
3 - Neutral	Culturally compatible	Doesn't affect safety dynamics	Neither helps nor hinders learning
2 - Friction	Cultural resistance likely	May create defensive behavior	Could limit experimentation
1 - Damaging	Undermines culture	Reduces safety, creates blame	Inhibits learning, creates fear

Assessment Questions:

- Does this create opportunities for learning and improvement?
- Will this tool/process be used to blame individuals for security issues?
- Does this support experimentation and innovation, or constrain it?
- How does this affect trust between security and engineering teams?
- Does this enable self-service and autonomy, or create dependencies and gatekeeping?

Cultural Red Flags:

- Tools that "catch" developers making mistakes (blame framing)
- Processes that require security approval for experimentation
- Metrics that rank individuals or teams on security "scores"
- Automation that blocks without explanation or education
- Systems that feel like surveillance rather than enablement

---

5. Organizational Change Requirements

The Question: What adoption challenges should we anticipate?

Why Important: Even technically excellent capabilities fail if organizational change requirements exceed capacity. Understanding change burden helps with sequencing (tackle easier wins first), resource planning (what support is needed), and communication strategy (how to frame the change).

Assessment Framework:

Score	Change Requirements	Adoption Difficulty	Success Requirements
5 - Minimal	Works with existing processes	Easy adoption, minimal training	Basic communication, standard rollout
4 - Low	Minor workflow adjustments	Some training required	Clear documentation, support availability
3 - Moderate	Significant workflow changes	Moderate training, practice needed	Change management, pilot programs, champions
2 - High	Major process transformation	Extensive training, culture shift	Sustained executive support, dedicated change resources
1 - Extreme	Organizational restructuring	Fundamental way-of-working change	Multi-year transformation, may require external help

Assessment Questions:

- How much does this change existing ways of working?
- What training or skill development does this require?
- Do we have change management resources to support adoption?
- What is organizational appetite for change right now?
- Are there competing changes creating change saturation?

Change Management Success Factors:

- Start with willing early adopters, not mandates
- Provide hands-on support during initial adoption
- Celebrate early successes and share learnings
- Address failures constructively, iterate based on feedback
- Ensure executive visible support throughout adoption

---

6. Adversary Economics

The Question: Does this close the surface it claims, or only raise the cost on paths already covered?

Why Important: Adversary economics is real, but it is easy to measure the wrong part of it. An attacker does not pay the average cost of attacking you. They pay for the cheapest way in that still works. Harden nine of an attacker's ten ways in and leave the tenth open, and that tenth path still costs what it always did. Your average goes up. The real cost to breach you does not move. This is why the breaches that hurt so often trace back to one dull thing: a storage bucket left public behind a well-run security program, an aging server no one re-checked. The damage hides in the seams between the parts you hardened, not in the parts themselves. So the test is coverage, not price. Does this capability actually close the surface it claims, or does it just raise the toll on paths that were already covered? Cost-raising still earns its

place, but on top of a boundary that contains the breach when one gets through, never in place of one. An expensive attack that slips through an open seam owns you just as completely as a cheap one.

One kind of cost-imposition earns full credit on its own: friction that comes with an alarm you can act on. Plant something that has no real use and watch who touches it. A fake admin login, a decoy database, a credential that should never be used. The moment anyone uses it, you know, because no legitimate person ever would. That is not raising the average toll. It is closing the blind spot where an intruder moves unseen. So the rule is simple: cost-imposition counts when it comes with a signal and a way to contain whatever trips it. Friction with nothing watching does not, because the attacker just reroutes to the next cheapest path and you never find out.

Deception is the clearest case

Honeypots, canary tokens, and honeytokens are the standard tools (the discipline MITRE Engage (https://engage.mitre.org/) organizes). A common real version: drop a fake cloud access key into a private repository. It does nothing, so the instant it shows up in your logs you have a true alarm and a rough location for the intruder, with almost no false positives. Deception works because of that signal, not the friction. It closes the detection surface while covering no prevention surface, which is also why Bejtlich's Intruder's Dilemma (https://taosecurity.blogspot.com/2009/05/defenders-dilemma-and-intruders-dilemma.html) ("the defender only needs to detect one indicator") is really a detection argument, not a cost one.

Assessment Framework:

Score	Adversary Impact	Attack Economics Change	Strategic Security Value
5 - Exceptional	Forces adversary pivot to different vector	Makes current attacks infeasible	Closes entire attack category
4 - High	Significantly increases attack cost/time	Requires sophisticated adversary capability	Substantially raises bar for successful attacks
3 - Moderate	Moderately increases attack difficulty	Deters opportunistic attackers	Reduces risk from common threat actors
2 - Low	Slightly increases attack effort	Minimal impact on determined adversaries	Limited practical security improvement
1 - Minimal	No meaningful adversary impact	Attacks remain economical	Compliance theater, negligible risk reduction

Assessment Questions:

- Does this eliminate an attack vector completely, or just make it harder?
- Do attackers need to develop new capabilities to succeed?
- Does this protect against automated/scaled attacks, or only targeted efforts?
- Will adversaries simply move to easier targets or different techniques?
- Does this address current adversary capabilities, or historical threats?

Adversary Evolution Context:

Adversaries evolved from targeted reconnaissance to automated discovery at internet scale. Investments that address this capability gap have exceptional adversary economics impact:

- Automated asset discovery (defenders find assets before attackers)
- Continuous vulnerability management (close windows of exposure)
- Supply chain automation (detect unknown dependencies)
- Real-time threat detection (match adversary speed)

Community-Level Success: Once in a while a whole field closes a path for good, and attackers leave it because it stops paying. Software has done this once already, at the transport layer. The web moved to encryption by default (https://letsencrypt.org/stats/), with free certificates and browsers warning on plain HTTP, and the old trick of sniffing someone's login over open wifi mostly died. Attackers did not keep trying it. They moved on.

At the level of the code itself, software has not done this yet. The best attempt so far is the shift to memory-safe languages. When you write in a language that makes an entire class of memory bugs impossible, those bugs stop appearing at all, instead of being found and patched one at a time. The early numbers are real: as Android moved new code to memory-safe languages, the share of its vulnerabilities that were memory-safety bugs fell from about three-quarters to under a quarter in six years (https://security.googleblog.com/2024/09/eliminating-memory-safety-vulnerabilities-Android.html).

That is the shape of the highest form of security success. Not the average cost of attack going up, but one whole attack method deleted from every target at once. It is rare, it is slow, and at the code level it is still a direction more than a destination. And even when a field finishes the job, it never closes your own seams for you. The industry can retire a path for everyone, and one team with a forgotten gap still loses through it.

4.4.4 Investment Scoring and Prioritization

Scoring Method

For each potential investment, score across all six criteria (1-5 scale):

1. **Manual Effort Reduction** (weight: 2x)
2. **Developer Experience** (weight: 1.5x)
3. **Time to Value** (weight: 1x)
4. **Cultural Alignment** (weight: 1.5x)
5. **Organizational Change** (weight: 1x, reverse scored, lower is better)
6. **Adversary Economics** (weight: 2x)

Total Score Calculation:

```
Total = (Manual Effort × 2) + (Developer Experience × 1.5) + (Time to Value × 1) +
        (Cultural Alignment × 1.5) + ((6 - Organizational Change) × 1) + (Adversary Economics × 2)

Maximum Possible Score: 47
Minimum Possible Score: 13.5
```

Priority Thresholds

Total Score	Priority	Action
38-47	Must Do	Implement immediately, highest resource priority
30-37	Should Do	Strong business case, sequence strategically
22-29	Consider	Evaluate context, may be valuable in specific situations
13.5-21	Avoid	Poor fit, likely low ROI or high failure risk

The Risk Override

One rule sits on top of the score. A high Manual Effort Reduction score cannot by itself lift an investment past **Should Do** when Adversary Economics is low. Effort saved is not risk retired, and the spine of this framework is risk, not activity. An investment that reduces real toil but closes no surface and contains no breach is a productivity buy, not a security one, and it should rank as the productivity buy it is.

The override has a second clause, for the failure the first cannot see. An automation can score high on effort saved and honestly high on Adversary Economics while still introducing a new credential, a new trust boundary, or a delegation path that nothing has accounted for. A score gates on a threshold. It never subtracts. So when the assessment question above surfaces a new authority or surface that is not already netted in Adversary Economics, that investment does not clear **Should Do** on effort alone, whatever the total reads. The override keeps the arithmetic honest to the spine: risk-realization decides the tier, not toil-realization.

The spend-side companion to this override is Defender cost economics: once the cost of the next control exceeds the risk it retires, you accept and sign the residual rather than keep adding controls.

Example Investment Evaluation

Proposed Investment: Automated dependency scanning with auto-remediation

Scoring:

- Manual Effort Reduction: 5 (eliminates 2,000+ hours of manual reviews annually)
- Developer Experience: 4 (automatic security fixes reduce developer toil)
- Time to Value: 5 (measurable benefits within 8 weeks of deployment)
- Cultural Alignment: 4 (enables learning, no blame dynamics)
- Organizational Change: 4 (low change requirement, integrates with existing CI/CD)
- Adversary Economics: 5 (addresses modern adversary capabilities, closes supply chain gaps)

Total Score: `(5×2) + (4×1.5) + (5×1) + (4×1.5) + ((6-4)×1) + (5×2) = 10 + 6 + 5 + 6 + 2 + 10 = 39`

Priority: Must Do (Score 39/47)

Justification: Exceptional score across all criteria. Addresses fundamental scaling challenge while improving developer experience and closing critical security gaps created by adversary evolution.

4.4.5 Position-Specific Evaluation Adjustments

Your strategic position affects criteria weighting:

Studio (Small reach + High Readiness)

Adjust weights:

- Increase Time to Value weight (rapid iteration preferred)
- Increase Developer Experience weight (preserve innovation culture)
- Decrease Organizational Change concern (high change capacity)

Prioritize: Quick-win automation, developer-centric tools, modern platform capabilities

---

Lean (Large reach + High Readiness)

Adjust weights:

- Increase Adversary Economics weight (sophisticated threat model)
- Increase Cultural Alignment weight (learning culture critical at scale)
- Maintain balanced approach across all criteria

Prioritize: Platform capabilities, federated solutions, cultural scaling mechanisms

---

Craft (Small reach + Lower Readiness)

Adjust weights:

- Increase Organizational Change sensitivity (limited change capacity)
- Prioritize readiness-building over automation breadth
- Focus on foundational capabilities enabling future investments

Prioritize: Readiness infrastructure, documentation, basic automation proving value

---

Mass (Large reach + Lower Readiness)

Adjust weights:

- Dramatically increase Organizational Change sensitivity (change saturation risk)
- Increase Time to Value weight (need quick wins for credibility)
- Focus on hybrid solutions working with legacy constraints

Prioritize: Strategic debt reduction, hybrid architecture, visible quick wins building momentum

---

4.4.6 Comparative Investment Analysis

Use this framework to compare competing investments:

Example Scenario: Choose between three automation investments with limited budget

Investment	Manual Effort	Dev Experience	Time to Value	Cultural	Org Change	Adv Eco
Dependency Scanning	5	4	5	4	4	5
Security Review Automation	4	3	3	3	3	3
Compliance Documentation	2	2	4	3	4	1

Decision: Prioritize dependency scanning: highest total score, addresses adversary evolution, exceptional ROI.

4.4.7 Common Evaluation Mistakes

Mistake 1: Optimizing for Single Criterion

Problem: Choosing investments based solely on one factor (usually cost or vendor relationship)

Consequence: Miss investments with exceptional overall value but moderate cost

Solution: Systematic multi-criteria evaluation, weighted scoring

---

Mistake 2: Ignoring Developer Experience

Problem: Selecting security tools security team loves but developers hate

Consequence: Poor adoption, workarounds, wasted investment

Solution: Pilot with developers first, weight developer experience appropriately

---

Mistake 3: Underweighting Cultural Alignment

Problem: Treating culture as "soft" factor, prioritizing technical capabilities

Consequence: Technically excellent solutions that fail organizationally

Solution: Recognize cultural alignment as hard requirement, not nice-to-have

---

Mistake 4: Chasing Vendor Hype

Problem: Investing based on vendor marketing rather than internal evaluation

Consequence: Tools solving problems you don't have, creating new maintenance burden

Solution: Start with your pain points, evaluate vendors against your criteria

---

4.4.8 Next Steps

1. **Catalog Potential Investments**: List scaling investments under consideration
2. **Score Systematically**: Evaluate each using six criteria framework
3. **Prioritize by Position**: Apply position-specific weighting adjustments
4. **Sequence Strategically**: Consider BAU vs Scaling investment timing
5. **Review Platform Effects**: Additional considerations if you're a platform company

---