Defending Against Autonomous Attacks:

Established Mechanisms, Verified Change,
and Safe Renewal

Hugo Latapie
Taijitu AI Inc.

November 24, 2026
Technical survey

Abstract

Reports published by Google and Amazon in November 2026 document AI-agent-driven exploit development and intrusions against real systems. They describe autonomous patch-to-exploit workflows and a successful agent-driven zero-day compromise. Automated patch production addresses only part of the defensive problem: a repair must close the vulnerability, preserve required behavior and be safe to deploy. This focused technical survey examines model-based construction and renewal as a defensive option, comparing its requirements and supporting evidence with those of agentic patch generation and autonomous deployment. A formal model specifies the required behavior for each accepted implementation. Structurally different implementations can satisfy that model, while renewal changes information on which an attack depends. The computational opportunity is to construct a correct change more cheaply than an attacker can recover the information needed for compromise. The analysis states the conditions under which that opportunity becomes a defensive advantage, including the timing condition $\tau < X$, retained attacker knowledge, safe state migration and actual retirement of old versions. Research implementations, open-source components and commercial mechanisms provide examples of these techniques. These precedents and the current limits of agentic repair make model-based construction and safe renewal important options to consider alongside human-supervised patching. A deployment must establish correctness, attack-cost separation and feasible renewal together.

1 Documented agentic intrusions

Google and Amazon's November 2026 reports describe agents developing exploits and executing intrusions against real systems [1, 2]. Humans supplied targets and goals, while agents selected actions, invoked tools and revised their approach from results.

On November 16, 2026, Google Threat Intelligence Group reported an attacker-operated framework called `exploit_agent`. It compared application code before and after patch announcements, identified potential vulnerabilities, drafted exploits and executed them without human intervention in that loop. The exposed staging server also contained exfiltrated victim data [1]. In this workflow, published fixes supplied information for an automated search for vulnerabilities in installations that had not yet been updated.

Amazon's November 20 report described Cairn activity observed on September 8–10. An operator supplied university targets and a goal of obtaining access; the agents planned, executed and adjusted their approach. In one operation, they progressed from API exploration to arbitrary file writing in about 11 minutes. A later session achieved root-level execution using the same vulnerability, which Amazon assessed with high confidence as a zero-day. Other operations continued for hours without succeeding [2].

The defensive pressure is immediate. In these operations, agents kept testing, interpreting responses and revising their approach; some obtained access, while others persisted for hours without success. Against a stable implementation, knowledge gained in one attempt can remain useful in the next. Businesses therefore face pressure to automate their response, including patch production. An attacker needs one way to compromise the system. A repair must close the vulnerability while preserving required service.

The incidents raise two defensive questions: how to correct vulnerabilities reliably, and when implementation changes can make an attacker's accumulated knowledge less useful. This survey focuses on model-based construction and renewal, comparing its requirements and supporting evidence with those of agentic patch generation and autonomous deployment.

Research on program evolution and code renewability connects changing implementations, retained behavior and attack costs [22, 26]. Specification-driven synthesis builds implementations from an abstract model, using refinement to show that they have the required behavior [12, 13]. Commercial products also use memory diversification and application hardening; their documented mechanisms are catalogued in Appendix H.

Each proposed change must be checked against the required behavior and deployment conditions. Renewal can select or construct an implementation that satisfies the existing specification, change relationships an attack depends on, and retire the old implementation under an established replacement policy. Agents can assist with construction and checking in either approach; the comparison concerns how changes are justified and deployed.

Section 2 distinguishes repair from diversification and examines the evidence on agentic coding. Sections 3–6 explain the computational opportunity, verification obligations and renewal conditions. Sections 7–8 review implemented mechanisms and the evidence needed to evaluate a deployment. Formal detail and the reference catalogue are in the appendices.

Figure 1 summarizes the two approaches.

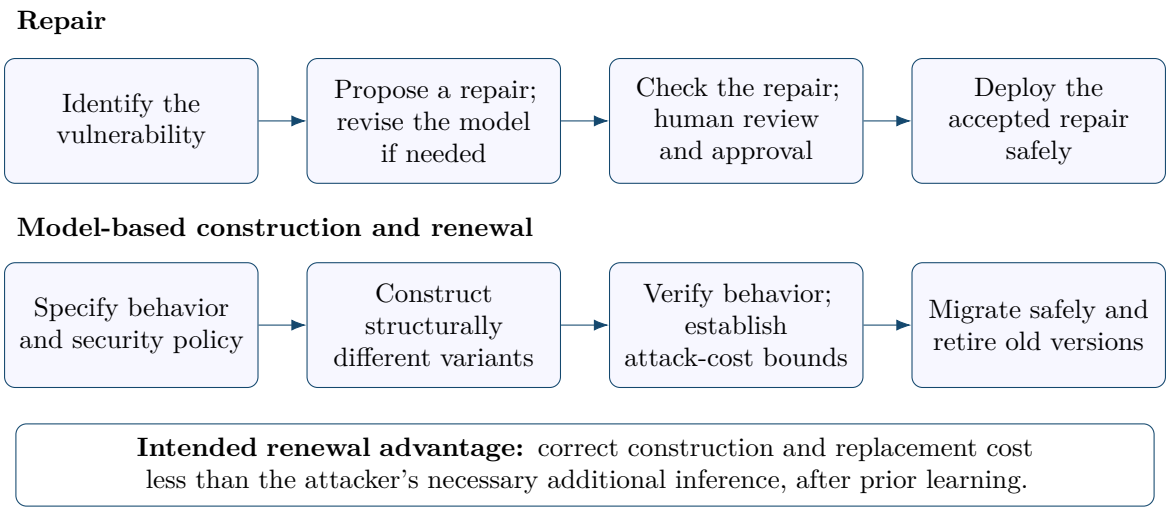


Figure 1: Two complementary uses of software change. Agents can assist both paths. Human review and approval are recommended for current agent-generated security repairs. The intended renewal advantage requires evidence for both the cost separation and a safe, timely replacement process.

2 Checking automated repairs

Repair and diversification. A repair removes erroneous or unauthorized behavior. It may require correcting the specification itself; preserving an inadequate model would preserve the defect. Diversification instead changes the implementation of an accepted contract while retaining its required behavior. Both can be automated, but they require different evidence. A diversified program can retain a logic flaw, and a repaired program can still expose stable implementation details useful to other attacks.

What the coding evidence supports. For agentic repair, a basic question is how often a generated change resolves its target problem. Ben Sghaier and colleagues evaluated 35 Qwen Code releases on 50 SWE-bench Verified tasks with a fixed model. Across releases, non-empty patches were produced for 52–94% of tasks, while 23–39% met the study's test-based resolution criterion [3]. These are ranges across releases, not paired endpoints or production-security failure rates.

In ML-engineering benchmarks, Brilliantov and colleagues found that a minimal autonomous coding agent matched or exceeded the tested elaborate harnesses with strong backbones; direct access to execution and feedback was especially valuable [4]. Neither study measures adversarial vulnerability closure or safe production rollout. Appendix I details their methods, findings and limits for security assurance.

Acceptance criteria. An agent may propose changes, generate tests and assist with proofs. Acceptance requires evidence that the change fixes the defect, preserves required behavior and is safe to deploy, including its effects on dependencies. The proposing agent should not be able to change the acceptance criteria or its deployment permissions on its own. Given the limits of the current evidence, this survey recommends human review and approval of agent-generated security repairs before production deployment. Automated tests and verification provide evidence for that review.

If a defense relies on a proof that implementations satisfy a formal model, the proof must cover each accepted variant and the replacement process. Existing patching and containment remain necessary for uncovered attack routes. Relaxing the acceptance criteria when evidence is missing leaves the corresponding security claims unsupported.

3 The computational asymmetry

The sought advantage is a cost asymmetry: constructing a correct implementation more cheaply than an attacker can recover the information necessary to compromise it. Preparation and renewal determine how that asymmetry can be used during operation.

3.1 Formal specification and constructive knowledge

A notation such as Z can specify states, invariants, operations, inputs, outputs and the security policy [6]. An equivalent specification in another formal framework can serve the same purpose. The specification may be written directly or obtained through a justified abstraction of existing code. Each accepted implementation must satisfy that specification.

Refinement relates concrete states and operations to that abstract meaning. One implementation may represent a state directly; another may distribute it among components and coordinate a different sequence of internal operations. The generator carries the correspondence

forward, with reusable family proofs or instance evidence. Appendix A states the specification and refinement obligations. Generation algorithms and proof tooling supply the construction process; Z supplies a language in which its obligations can be expressed.

This is the defender's constructive knowledge. The attack model specifies how much of it is also available to the attacker, including the formal specification, generator, proof metadata and current or prepared implementations.

Figure 2 shows the common model and different implementation structures.

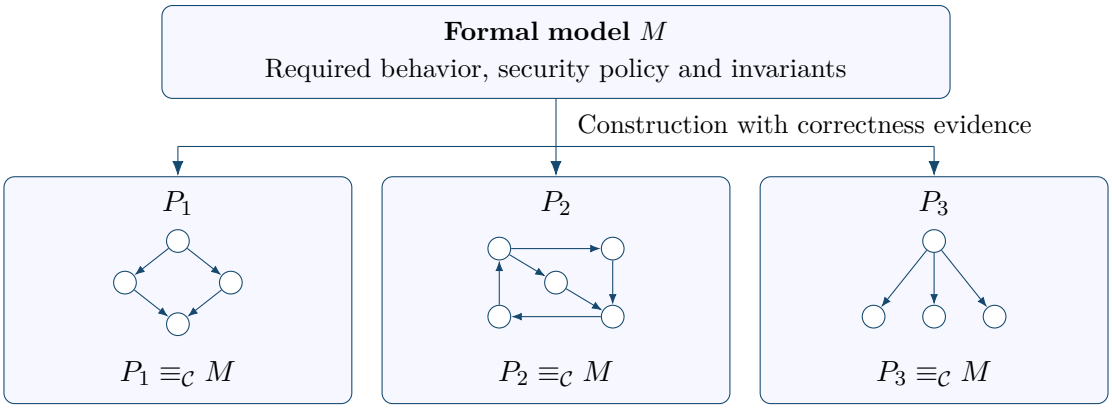


Figure 2: Model-based construction with a common behavioral contract. Each accepted realization satisfies $P_i \equiv_{\mathcal{C}} M$, the contract-relative equivalence defined in Appendix A. The graphs schematically represent different internal organizations. Any attack-cost advantage must be established for the chosen family.

3.2 Attack-relevant inference cost

An attacker needs information sufficient for a successful attack. That may be a small relationship within the implementation, found entirely by automated analysis. We call the work needed to obtain it *attack-relevant inference cost*. The cost is defined for a specific information-recovery task and success criterion.

The desired separation compares a bounded cost of correct construction and replacement with a lower bound on necessary attack work. Both accounts include reusable preparation and retained knowledge, under declared resource conventions. A correctness proof supplies the behavioral guarantee. Establishing the cost separation additionally requires showing which information every covered attack needs and how hard that information is to recover. Appendix B states the conditions for converting a recovery-cost bound into an attack-risk bound.

3.3 Which information matters?

The game-theoretic question is whose necessary information can be changed cheaply, and who must pay to recover it. The physical examples below distinguish information dependencies, not implementation techniques.

Consider an attack for which a known, fixed geolocation is sufficient. The attacker's information requirement is already satisfied, and the location does not become stale. This is not a suitable case for a defense whose advantage depends on repeatedly invalidating necessary attacker knowledge. Visual concealment may help during discovery or against an attack that also needs visual identification, but it is not a general solution when the known location is enough.

The asymmetry can instead favor the attacker. A fixed location's defender may need to

distinguish genuine threats from false targets or cues before allocating a response. An attacker able to create those cues cheaply can impose greater discrimination and response costs on the defender, without having to relearn the target's location. Here the defender bears the costly inference burden.

Aircraft flares illustrate the opposite direction. When successful action requires identifying and tracking an infrared source, competing cues can disrupt that necessary task [8]. Knowing an approximate location is not sufficient for this class of attack. Where producing the cues costs less than overcoming their effect, the information-cost advantage favors the defender.

Verified software renewal is relevant when it can create this latter asymmetry: the defender retains the knowledge needed to construct a correct realization while changing facts the attacker must recover. Descriptive complexity alone does not establish either player's advantage.

Situation	Information needed for action	Information-cost effect
Location-sufficient threat	Already known coordinates of a fixed target.	No renewed attacker inference task; concealment is not a general remedy.
Fixed-location defense	Defender must distinguish genuine threats from false cues.	Cheap attacker-generated cues can impose costly discrimination and response.
Infrared tracking	Attacker must identify and track the relevant source.	Defender-generated flares can burden that necessary tracking task.
Polymorphic software	Realization-specific facts required for compromise.	Verified renewal changes those facts while preserving service.

Table 1: The same information-cost comparison applies across domains. Either player may have the advantage; changing information matters only when the opponent needs it for successful action.

3.4 Renewal and the lifetime of useful knowledge

In model-governed diversification, generation changes attack-relevant relationships while preserving the formal contract. The changed relationships must leave prior knowledge insufficient for a new compromise without additional work. Each renewal must sustain a remaining-work bound after the attacker's accumulated learning, including knowledge of the generation family.

Model development and family proofs can be amortized, and a prepared inventory can separate generation cost from deployment latency. These measures help the defender use a computational gap at a useful cadence. Translating work into the time bound X also requires the attacker's allowed processing rate, parallelism and other resources.

Turning the cost gap into a deadline. Let X be a lower bound on the additional time required by the fastest attack in the threat model, after the attacker's prior learning. Let τ be the time until the affected attack opportunity is closed by replacing and retiring the relevant implementation. The target condition is

$$\tau < X. \tag{1}$$

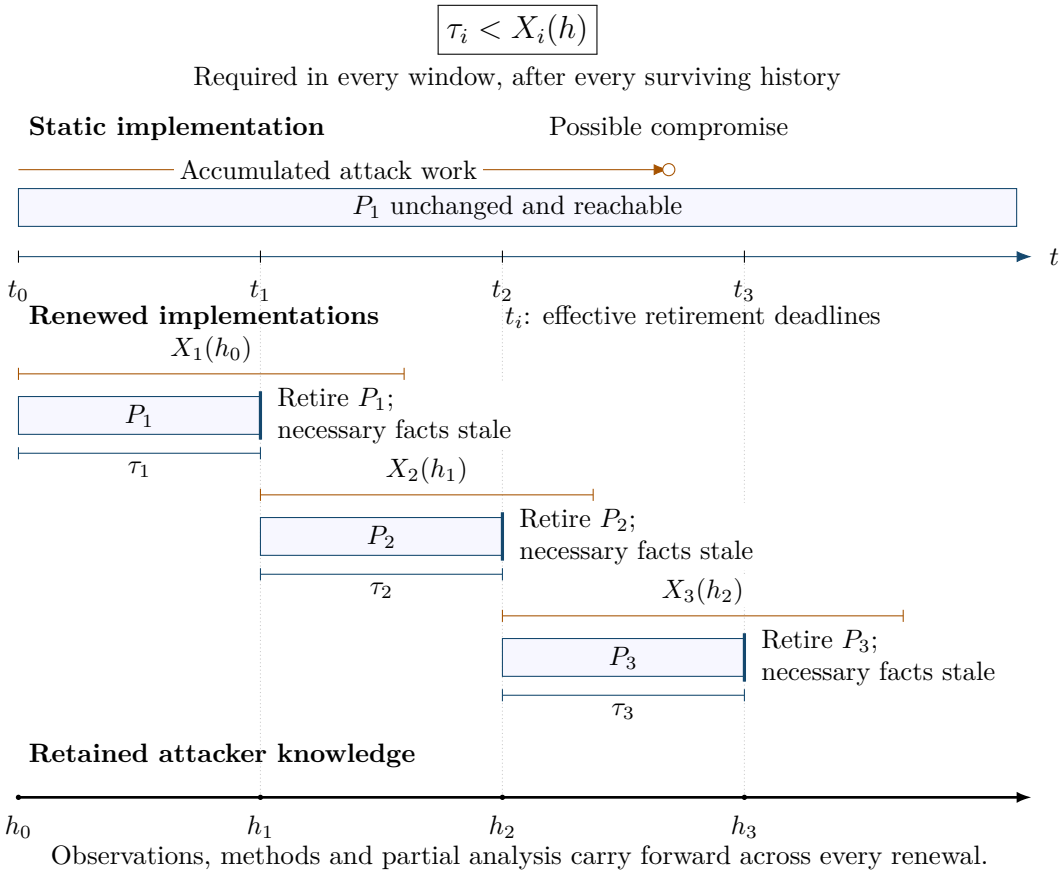


Figure 3: Effective retirement precedes the remaining compromise-time bound in every window. Here $\tau_i = t_i - t_{i-1}$, and h_{i-1} contains the attacker's accumulated history at t_{i-1}. The orange brackets show lower bounds $X_i(h_{i-1})$ after that learning, rather than predicted completion times. Renewal makes necessary implementation-specific information stale; the attacker retains its accumulated knowledge. The static row shows one possible compromise; lengths and non-overlapping replacements are schematic. Applying the condition requires an uncompromised start, safe migration, and bounds covering all admitted attack routes, reachable versions, early artifact access and permitted resources (Appendix D).

Figure 3 illustrates the history-aware form of this condition. A platform that can be compromised after sufficient analysis can remain secure during operation if renewal maintains this inequality for every admitted attack route and every window of the mission, starting from an uncompromised system. Appendix D states the conditions precisely. Applying the result requires a system-specific lower bound on attack time and a replacement process that meets the deadline.

The definition of X includes accumulated analysis, pre-activation artifact access and the allowed parallel resources. Each renewed state must support the remaining-time bound after that history. Effective retirement includes rollout overlap and reachable old versions.

4 Engineering proactive morphing

4.1 Preserving specified behavior

The behavioral contract specifies permitted inputs, observable outputs and effects, authorized state changes, and required progress. Timing, memory, concurrency and fault behavior must

be included wherever they matter. Correspondence is behavioral: radically different concrete structures can implement the same abstract system. The proof must cover the behavior promised by the contract. Appendices A and C give the equivalence and live-replacement conditions.

A correctness argument covers generation, compilation, linking, execution and deployment, identifying which stages are proved, checked or trusted. CompCert illustrates a scoped semantic-preservation guarantee [11]. Structural differences that matter for an attack must survive compilation and the attacker's available analysis.

4.2 Modular morphing

Renewal can replace one module or a group of coupled modules while the rest of the platform continues operating. The replacement's proof must cover its behavior in the surrounding system. Shared-state invariants, calling conventions, concurrency and migration determine which components can change independently and which must change together.

The security task is to choose groups whose changes retire the dependencies needed by attacks in the threat model. Unchanged components remain part of the coverage analysis, and the remaining-time bound must hold for every covered route after each update. Modularization can reduce generation, transfer and switching costs; those costs and the effective retirement time determine whether the timing rule is feasible. Appendices A and C state the composition and migration conditions.

4.3 Changing attack-relevant relationships

Restricted changes such as identifier renaming leave the organization of a program largely intact. A normalization pass can remove such differences without reconstructing a new representation of the system. By contrast, model-driven synthesis can change several interdependent implementation choices while preserving their common abstract meaning. An attacker may then have to resolve relationships among encoded state, control decisions and component interactions rather than reverse isolated substitutions.

The relevant measure is the work still required for a successful attack after simplification, observation and accumulated learning. A canonical structure, transferable exploit or black-box method can reduce that work substantially.

Branches and reachability. Compact code can describe a large space of possible executions. Determining whether a branch is reachable is undecidable for unrestricted programs; bounded analysis can still face severe path and constraint complexity [9, 10]. Reusable refinement rules let the defender generate structure without reproving every concrete path separately. The security objective is to make the relationships needed for an attack expensive to resolve. An attacker can use partial analysis, concrete executions or one successful path, so a lower bound must cover those options. Appendix B distinguishes general undecidability, path counts and the necessary inference task.

Semantics-based protection research models what an analysis can preserve or lose [17]. Specialization of distorted interpreters constructs equivalent programs while making specified abstract interpreters incomplete [18]. Such results explain how changed representations can require a different or more precise analysis.

Obfuscation research includes both restricted transformations and generative, structural, formally semantic methods. Comparing these methods requires identifying the specification or program whose behavior they preserve, the structures that can change, the verification method and any live-replacement mechanism.

4.4 Attack coverage

Morphing addresses attack paths that depend on the structures being changed. A flaw allowed by the formal model may persist across all implementations. Stolen credentials or control of the deployment process can also bypass implementation changes. Patching, access control, secret management and containment contribute to covering these routes.

5 The asymmetric security game

The examples in Table 1 suggest a game in which both players can change what the other must learn and what it costs to act. Advantage depends on the information needed for the objective, the available observations and the two resource budgets.

5.1 Choices and objectives

The defender chooses which implementations to prepare, which modules to change, what artifacts to expose, and when to replace and retire each version. A renewal policy is the rule for making these choices; it can follow a schedule or respond to observations. The attacker chooses how to learn from variants, divide its resources, probe the system, select targets and time a compromise. Where the threat model admits deceptive observations, the attacker can also spend resources on cues that increase the defender's decision costs.

The attacker seeks to maximize the chance of a first compromise during the mission. The defender seeks an affordable renewal policy that minimizes the risk against the strongest permitted response while maintaining legitimate service. Each side has its own resource budget. The first compromise determines the mission's security outcome. Appendix E organizes these choices into a game. It supplies an objective and comparison criteria; a particular construction supplies the dynamics needed to determine a policy.

5.2 Information changes and best responses

Renewal can change both the implementation and what an attacker observes. Flares and false targets illustrate changes to observations that affect identification or discrimination. A game model therefore tracks the underlying state separately from each player's observations, and charges for creating changes, obtaining information and responding.

The location-sufficient example is a boundary case: an attacker that already has the fixed information needed for action has no new inference task to perform. The defender may nevertheless face costly discrimination of genuine threats from cheap false cues. More generally, a player may choose a response that avoids the intended inference burden. The policy comparison includes these alternatives. Ambiguity matters only through its effect on feasible actions, costs and the chance of a violation.

5.3 Risk over a mission

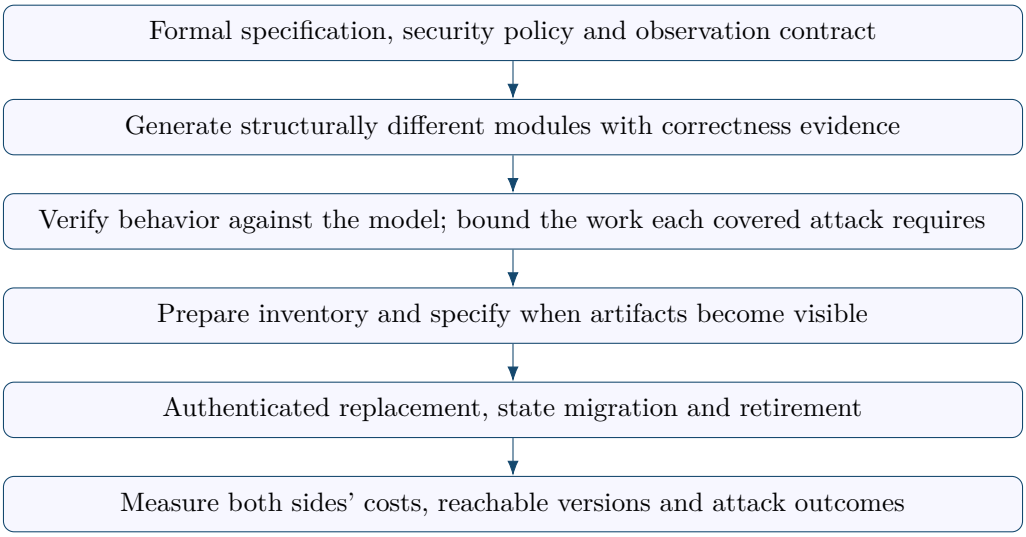
Where an attack has a chance of finishing within a window, the defender instead bounds that chance. Each window's bound accounts for what the attacker has learned, given that no compromise has yet occurred. These bounds combine into a risk bound for the whole mission, carrying knowledge and resource use forward across windows. Theorem 2 derives the accumulation rule. The bounds must cover every permitted attack and remain valid for the chosen renewal policy.

Choosing a feasible cadence. Faster renewal creates shorter, more numerous attack windows. The choice of cadence depends on how quickly the risk per window falls and on the cost of each update. A useful cadence must preserve service, fit the defender's budget and meet the mission's security target. Appendix F derives the risk and cost tradeoff.

What establishes a defensive advantage. A fair comparison gives static and renewed systems the same functionality, mission and resource budgets, and allows the attacker to adapt its methods to each. A strict advantage is established when the renewed system's worst-case risk bound falls below the success probability established for an attack on the static system. Appendix E states the comparison and its extension to a class of static policies.

6 Deployment requirements

A deployment must maintain the conditions of the analysis while serving real requests. This requires a supply of verified replacements, safe state migration and retirement of the old attack opportunities. The analysis must also track what the defender knows from construction and what the attacker can learn from exposed artifacts.



Construction and inventory. Each accepted implementation has correctness evidence, a supported state representation and resource limits. A proved generation family or translation validation can supply that evidence. The inventory must cover the mission or be replenished fast enough. Failed validation and stockouts must be handled within the service and exposure limits.

Stateful replacement. Migration must preserve authorized state, transaction semantics and progress, including in-flight requests and retries. Appendix C gives sufficient conditions for preserving behavior during replacement.

Actual retirement. Account for old replicas, sessions and implementation-bound capabilities that keep an attack opportunity open, including versions reintroduced during recovery. Early access to a prepared implementation can also give the attacker a head start.

Defensive cost. The cost account includes proof preparation, generation, execution, debugging, observation processing and renewal infrastructure. Responses to attacker-induced ambiguity

also consume the defensive budget. Generation throughput, tail latency and service continuity constrain the feasible renewal rate.

7 Choosing established mechanisms

Existing tools and systems support different parts of these deployment requirements. Specware and Fiat establish specification-driven construction; verified transformations and equivalence checkers address preservation; CROW and DivCon provide implemented diversification approaches; ASPIRE and Shuffler demonstrate renewal mechanisms [12, 13, 23, 24, 19, 20, 26, 31]. Commercial memory-randomization and application- hardening products cover further, usually narrower, deployment roles. Appendix H supplies the implementation, repository, license, maintenance and assurance details.

Selection should start with the attack dependency and the behavior that must remain correct. Load-time relocation, changes to internal program structure and continuous code replacement have different effects and deployment requirements. For each component, identify the properties it checks or preserves, its deployment requirements and its maintenance status. A proof for the combined system must cover the selected components and their interactions.

The decision model also has predecessors. FlipIt models attacker–defender competition for control of a resource under limited observations and move costs [33]. Its objective concerns time in control and costs. The security payoff here is a first violation during a finite mission. The attacker retains what it learns across renewals. Comparing policies therefore requires specifying how each replacement affects the information available to the attacker and the work still needed for a compromise.

8 Evaluating defensive options

To claim the computational advantage analyzed here, a deployment must show that verified changes cost less than the attack work they make necessary, after accounting for prior learning. This paper states the conditions for that comparison and its effect on attack risk; it does not supply a concrete generator with a proved cost separation.

For a particular platform, establish:

1. an explicit specification, such as a Z model, and refinement evidence for generated code and live replacement;
2. the information every covered attack needs and a bound on the cost of obtaining it, accounting for public artifacts, prior learning and bypasses;
3. a full defensive cost account and a justified translation from attack work to remaining time or conditional risk;
4. evidence that the defender can supply replacements, retire old versions and sustain service at the chosen renewal rate, within a budget that includes responses to attacker-generated observations;
5. coverage of attack paths through changed and unchanged components.

The refinement proof and attack-cost argument must refer to the same generated realizations and exposed information.

Useful comparisons include a static implementation, surface-only rewriting, deeper implementation transformation, model-driven synthesis without renewal, renewal without meaningful diversity, and verified model-governed renewal. An existing integrated renewal design provides a further baseline. These separate the contributions of generation and renewal. Use the common mission and resource conditions of Section 5, allowing the attacker to adapt to each policy. An

invariant-exploit control checks whether renewal leaves sufficient attack information unchanged. It tests the same information-dependency boundary illustrated by the location-sufficient example. Where deceptive observations are admitted, evaluate the defender's discrimination and response costs as well. Empirical trials characterize the tested policies; Appendix G states the sampling conditions and inference.

Complexity claims concern the actual generated distribution, information required by a successful attack, and the permitted resources and observations. General obfuscation impossibility results constrain virtual-black-box claims [34]. The security argument is specific to the chosen family and attack model.

9 Conclusion

The documented intrusions show agents developing exploits and obtaining access to real systems. The reliability limits reported in the coding studies warrant care in using agents to repair and deploy software. For current agent-generated security repairs, human review and approval should accompany testing and verification before production deployment.

Formal specification, verified transformation, diversification and renewal have research implementations and commercial counterparts at different scopes. Each accepted implementation must satisfy the specification. A repair must remove the defect, correcting the specification if necessary. Renewal must preserve service and retire the old implementations and capabilities that keep an attack possible.

Where correct change is cheaper than necessary attack-relevant inference, proactive renewal can exploit the difference. The timing condition $\tau < X$ excludes compromise only when maintained across all modeled routes and mission windows from an uncompromised start. Otherwise, justified history-aware bounds describe residual risk. The information-dependent game also permits the advantage to reverse, as the geolocation and false-cue examples illustrate. Given today's limits in autonomous patching and deployment, model-based construction and safe renewal deserve serious consideration alongside human-supervised repair.

A Formal model, contract and constructive correspondence

A.1 Contract-relative equivalence

Let M be the formal model of the required behavior, with abstract states, admissible initial states and transition semantics. Let $\mathcal{C}$ specify admissible environments, observable effects and progress requirements. The model may originate in a specification or in a justified abstraction of existing code; that origin does not constrain all future realizations to the original code structure. For a model or implementation S, let $\mathrm{Tr}(S, e)$ be its executions in environment e, and let Obs retain the contract observations. A concrete realization P_r corresponds to M when

$$P_r \equiv_{\mathcal{C}} M \quad \Longleftrightarrow \quad \forall e \in \mathcal{C} : \quad \mathrm{Obs}(\mathrm{Tr}(P_r, e)) = \mathrm{Obs}(\mathrm{Tr}(M, e)). \tag{2}$$

Traces retain termination, divergence and timing information whenever the contract distinguishes them. Progress-sensitive observations account for unbounded internal work. If an application requires refinement rather than equality, the direction of trace inclusion must be stated explicitly. Two implementations may refine the same nondeterministic specification while selecting different observable behaviors. Shared conformance alone does not imply the equality claimed here.

For interactive systems, scheduling, exceptions, concurrent effects and any permitted resource variation belong in the contract or in separately proved obligations. Undefined behavior and

trusted external components are part of the assurance boundary. Simulation, refinement or stuttering bisimulation must be chosen according to the actual behavioral claim.

A.2 Specification and refinement obligations

In a Z-style presentation, a state schema declares $x : S$ and an invariant $I(x)$. An initialization schema selects allowed initial states. An operation schema relates a before-state x, inputs u, an after-state x' and outputs v, with the state invariants included. Write this relation as $\mathrm{Op}_M(x, u, x', v)$ and its enabledness predicate as $\mathrm{pre}_M(x, u)$. Z conventionally marks inputs with ? and outputs with ! [6]; the relational notation here keeps the obligations independent of a particular specification language.

For a concrete operation Op_r and a relation R_r between invariant-satisfying concrete and abstract states, one forward-simulation obligation is

$$\begin{aligned} s\, R_r\, x \;\wedge\; \mathrm{pre}_M(x, u) \;\wedge\; \mathrm{Op}_r(s, u, s', v) \\ \Longrightarrow \exists x' :\; \mathrm{Op}_M(x, u, x', v) \;\wedge\; s'\, R_r\, x'. \end{aligned} \tag{3}$$

Concrete initialization must establish a related allowed abstract state, and abstract enabledness must imply concrete enabledness. Progress and observation obligations complete the trace-refinement argument. The equivalence in Equation (2) additionally requires matching the abstract behavior in the other direction.

Rejection and unauthorized-request behavior must be specified wherever it is covered by the assurance claim. Placing an input outside an operation's precondition leaves its treatment unspecified. The security analysis identifies the violation event V and the routes not already excluded by the behavioral assurance. A route proved impossible under all its admitted adversarial conditions needs no additional inference-cost argument.

A.3 Proved families and validated instances

A generator with policy θ and realization choice r may have the interface

$$\mathrm{Gen}(M, \theta, r) = (P_r, R_r, \pi_r), \tag{4}$$

where R_r records the applicable concrete-to-abstract correspondence. Acceptance has the form

$$\mathrm{Check}(M, P_r, \mathcal{C}, R_r, \pi_r) = \mathrm{accept}. \tag{5}$$

The evidence π_r can be an instance certificate or a family theorem whose preconditions hold for this instance. Sound acceptance implies $P_r \equiv_{\mathcal{C}} M$; the soundness premise, the actual construction proofs and the trusted stages must be supplied by the generator's assurance argument. The relation may involve many concrete states or steps per abstract state or step, permitting implementations with different graph sizes and structures.

Composition. If every accepted realization satisfies $P_r \equiv_{\mathcal{C}} M$, any two accepted realizations have the same projected traces and are therefore contract-equivalent. Likewise, a finite chain $S_0 = M, \ldots, S_k = P_r$ preserves equivalence when each step satisfies $S_{j+1} \equiv_{\mathcal{C}} S_j$ under established preconditions and the common observation contract. Both conclusions follow by transitivity.

A.4 Modular realization families

A realization index can be a tuple $r = (r_1, \ldots, r_k)$ selecting implementations of k modules. A renewal changes selected entries while preserving the others. The complete configuration must satisfy the abstract system M and observation contract $\mathcal{C}$.

Module contracts and a validated contextual composition rule must justify the required equivalence of each resulting system. The rule accounts for shared state, interfaces, scheduling and other cross-module invariants. Under those conditions, the configurations form an equivalence-preserving chain, with existing proofs reused for unchanged modules. Live replacement also satisfies the migration conditions in Appendix C.

B Attack-relevant inference and cost separation

B.1 Construction, recovery and compromise

Correctness asks whether a realization has the required abstract behavior. Semantic recovery asks for information about its internal realization. Compromise asks for a violation of a specified security property. These are different computational tasks.

The semantics-based results discussed in the main text show how a correct transformation can reduce the precision of an abstract interpreter [17, 18]. An attack-work argument must account for the additional analysis needed to recover information that a successful attack requires.

The recovery target is attack-useful information. Different histories can produce the same program, and many different explanations may be sufficient for an attack. The target accepts *any* witness adequate for the relevant attack. The attacker need not recover the complete specification or the defender's particular construction history. The information requirement is established through the attack-to-witness argument below.

B.2 Reachability and path complexity

General branch reachability is undecidable. To see the connection to halting, consider a program that simulates an arbitrary computation and reaches a marked location only if that computation halts. A general reachability decider would decide halting [9]. This establishes a limit on universal analysis. An individual execution can still reveal the branches it takes, and a particular program can have easily proved reachability properties.

With finite input, state and execution bounds, reachability is decidable, though potentially expensive. Even a loop-free Boolean program can encode satisfiability: given a propositional formula ϕ, evaluate it on input bits and reach a marked location exactly when ϕ is true. Constructing that program takes work linear in the formula representation; asking whether some input reaches the location is the SAT problem. SAT is NP-complete, while the separation of P and NP remains open [9]. This comparison concerns the worst-case decision problem, not the hardness of each generated instance.

A compact sequence of branches can represent exponentially many paths. Symbolic execution reasons about sets of inputs and constraints, and practical searches may find a violation through partial exploration [10]. Counting paths therefore does not establish that an attacker must enumerate them. For the proposed defense, correctness-preserving generation must produce a distribution of realizations on which the *necessary* attack-information task remains costly, including after simplification and prior learning. The criterion below connects that task to successful compromise.

B.3 A necessary-recovery criterion

Fix a defensive policy D and a surviving adversarial history h, the distribution of the currently reachable realizations conditional on that history, and a declared work measure. Conditional distributions are understood in their regular conditional sense, with uniform bounds interpreted

almost surely on surviving histories. Let $\mathcal{R}$ specify what counts as an attack-useful semantic witness. For example, it might identify a realization-specific state relationship required for a particular unauthorized modification. Its necessity must be established for the admitted attack class.

Write $\mathsf{Rec}(t \mid h)$ for the supremum probability of producing a valid $\mathcal{R}$ witness using at most t additional work, with the admitted observations and auxiliary information. This information includes the formal model if public, all earlier artifacts and attacks, and whatever current artifact access the threat model permits. Preprocessing and consumed resources carry forward across renewal. For the comparison below, the recovery algorithm's resource envelope includes the conversion overhead. The actual attacker keeps its original resource envelope. Dependence on D is suppressed in Rec and Succ; future realization, exposure and renewal choices follow that policy. A bound for one policy need not hold for another.

Transferring a recovery bound to attack risk. Suppose every admitted attack continuation using at most t additional work can be converted into a witness-producing algorithm using at most $t + e$ work, without access to extra secret information or unaccounted resources. Suppose a successful attack fails to yield a valid witness with joint probability at most δ, conditional on h. Then

$$\mathsf{Succ}(t \mid h) \leq \min\{1, \mathsf{Rec}(t + e \mid h) + \delta\}, \tag{6}$$

where Succ is the supremum success probability for that attack class and the conversion premise holds uniformly over it.

For any admitted attack with success probability p, the converted algorithm produces a valid witness with probability at least $p - \delta$, using at most $t + e$ work. By definition this is at most $\mathsf{Rec}(t + e \mid h)$. Rearranging, taking the supremum over attacks and capping by one proves the result.

An application supplies two inputs to this transfer rule: an attack-to-witness conversion valid for its attack class, including invariant bypasses, and a recovery bound for the actual generated distribution at the relevant budget. One route to the latter is a reduction from a precisely specified hard problem that faithfully reproduces the artifact distribution and side information.

A uniform bound on the right-hand side of Equation (6) over admissible surviving histories can supply a conditional attack bound to Theorem 2. A whole-system claim requires coverage of all admitted compromise routes. The resulting bound includes extraction overhead e and exceptions δ.

B.4 Comparing construction and inference costs

Use a declared common computation unit. Let $d(h) > 0$ bound the defender's work for the selected renewal, including generation, validation, migration and retirement. Setup and ongoing service costs remain charged in the mission account of Appendix F. Fix a proposed cost multiplier $\kappa > 1$ and a success threshold $p \in (0, 1)$. The comparison experiment permits $\kappa d(h)$ attacker work, with the charged overhead e additionally available to the witness conversion. Under the conversion premises above, the condition

$$\mathsf{Rec}\big(\kappa d(h) + e \mid h\big) + \delta < p \tag{7}$$

implies that every admitted attacker using at most $\kappa d(h)$ additional work has conditional success probability below p. This follows by substitution in Equation (6). It compares a defensive upper cost bound with an attack-work requirement at an explicit success threshold, after retained learning. Uniform use across renewals requires the corresponding bounds on every admitted surviving history.

To establish a computational gap independently of retirement, apply the argument in a comparison continuation that keeps the selected realization available, with inherited knowledge and explicitly stated observation access. Retirement alone cannot supply the claimed inference lower bound. Applying the result to live renewal then requires a bound valid for the actual policy, including any additional information exposed by other realizations.

A probability-qualified work separation supports a risk bound. The zero-compromise result requires the stronger minimum remaining-time premise in Appendix D. Translating work to elapsed time requires the permitted parallelism and processing rates; memory, tool access and other resource limits remain explicit. Equation (7) is a sufficient criterion for a chosen construction, whose recovery and cost bounds must be established.

C Stateful replacement

Live replacement connects program behavior to state migration and continued service. Let the abstract model M and each reachable variant be labeled transition systems with observable actions and internal actions. All components, including the deployment protocol, share the same contract observations.

For each variant r, let R_r relate its concrete states to states of M. For this handover result, require the stronger obligation that R_r is an observation-preserving weak bisimulation on the admissible reachable states. Require divergence-sensitive matching and preservation of any contract timing and progress obligations. The construction and deployment assurance must establish this relation.

A migration from state s of variant r to state s' of variant r' is permitted only at declared safe points and must preserve a common abstract state:

$$s\, R_r\, x \quad \Longrightarrow \quad s'\, R_{r'}\, x. \tag{8}$$

More generally, an observable migration must match the corresponding model behavior. The whole switching protocol must preserve enabled behavior and required fairness, and must not introduce an infinite internal switching sequence that prevents abstract-system progress. One sufficient operational restriction is a finite number of completed switches during the mission, with bounded handover delay within the contract and progress-preserving scheduling.

Proposition 1 (Trace preservation across renewal)**.** *Suppose the initial deployed and abstract states are related. If the execution and migration conditions above hold throughout the mission, the renewing deployment and M have the same contract-observable traces over that mission.*

Proof. Relate a deployed state in mode r to x exactly when $s\, R_r\, x$. An ordinary execution step is matched by the corresponding bisimulation. A silent migration preserves the same abstract state by Equation (8), so it is matched by stuttering. Conversely, an enabled model step remains matchable in the active variant; migration and protocol scheduling do not remove that obligation. Induction matches every finite execution prefix in both directions. The divergence, fairness and timing premises preserve admissibility and progress when extending those prefixes to complete mission traces. Projection onto the contract observations therefore gives equality. □

Deployment-specific proofs establish transaction atomicity, exactly-once effects, availability and safe retries through the execution and migration relations and the scheduling conditions. One-way simulation establishes directional trace refinement; equivalence requires bidirectional matching.

For modular renewal, r and r' may differ only in the selected components. Unchanged state can retain the identity relation. The module relations and a valid composition rule must

establish the global R_r and $R_{r'}$, including shared-state invariants. The same common-abstract-state migration condition then applies to the group being changed. Coupled components are renewed together where their invariants require it.

D Adversary and mission-risk model

D.1 Resources, histories and exposure

Fix a mission $[0, H)$, a violation event V, a feasible defensive policy D, an adversary class $\mathcal{A}$ and an aggregate resource envelope $\mathcal{B}$. Appendix E defines the defensive policy space; this appendix analyzes one fixed policy. The envelope includes preprocessing, online computation, memory, model/tool calls, probes, parallel workers and elapsed time where relevant. Any conversion among those resources must be justified. The attacker may know the formal model, any original source, construction algorithms and policy, and retain all obtained information.

Partition the mission into chronological intervals $I_i = [t_{i-1}, t_i)$, $i = 1, \dots, m$, with $t_0 = 0$ and $t_m = H$. These accounting intervals can include multiple reachable binaries. Deployment state includes all reachable replicas, rollout overlap and any still-valid implementation-bound capabilities. Early artifact access and continuing analysis are carried forward in the adversarial history.

For a fixed admissible strategy pair (D, A), let $\mathcal{F}_i$ contain the adversary's retained information at time t_i, including attacker state, prior observations, consumed resources and the information about deployment available under the threat model. Let E_i be the event of no violation before t_i, and include its indicator in $\mathcal{F}_i$ for the survival analysis. This accounting does not grant the adversary otherwise unavailable observations or reveal undisclosed future generation choices. We assume E_0 holds; a pre-existing compromise requires a separate initial-state term. Probability concerns declared generation, environment, deployment and adversary randomness.

Definition 1 (Bounded operational security)**.** A fixed defensive policy D is $(H, \mathcal{A}, \mathcal{B}, \varepsilon)$-secure if

$$\sup_{\substack{A \in \mathcal{A} \\ A \text{ respects } \mathcal{B}}} \Pr_{D,A}[V \text{ occurs in } [0, H)] \le \varepsilon. \tag{9}$$

The policy must also meet the correctness, useful-service and operational resource constraints.

D.2 A history-aware bound

Theorem 2 (Finite-horizon accumulation)**.** *For a fixed D, let $\epsilon_i = \epsilon_i(D) \in [0, 1]$ be fixed bounds valid uniformly over admissible adversaries and continuations. Suppose*

$$\Pr_{D,A}(E_i^c \mid \mathcal{F}_{i-1}) \le \epsilon_i \quad \textit{almost surely on } E_{i-1} \tag{10}$$

for every interval and every A respecting $\mathcal{B}$. Then

$$\sup_A \Pr_{D,A}(E_m^c) \le 1 - \prod_{i=1}^{m}(1 - \epsilon_i) \le \min\left\{1, \sum_{i=1}^{m} \epsilon_i\right\}. \tag{11}$$

Independence among intervals is not required.

Proof. Because $E_i \subseteq E_{i-1}$, the tower property and the conditional premise give

$$\Pr_{D,A}(E_i) = \mathbb{E}_{D,A}\left[\mathbf{1}_{E_{i-1}} \Pr_{D,A}(E_i \mid \mathcal{F}_{i-1})\right] \ge (1 - \epsilon_i) \Pr_{D,A}(E_{i-1}).$$

Iterating from $\Pr_{D,A}(E_0) = 1$ bounds survival by the product. Taking complements and then the supremum proves the first inequality. The elementary inequality $1 - \prod_i (1 - \epsilon_i) \leq \sum_i \epsilon_i$, with probabilities capped at one, proves the second. □

The almost-sure formulation avoids requiring individual histories to have positive probability; continuous histories need not do so. Each ϵ_i is supplied by an attack-specific argument valid after retained learning and preprocessing. The global resource envelope constrains all reachable histories and allocations across interval boundaries.

D.3 The zero-compromise timing condition

Use the mission intervals and initially uncompromised state above. Let $\tau_i = t_i - t_{i-1}$. Suppose that, after each surviving history h at t_{i-1}, every admitted continuation producing its first violation in I_i requires at least $X_i(h)$ additional elapsed time. The bound includes retained analysis, early exposure, all reachable versions and the permitted parallel resources. If, uniformly over admitted attackers and almost surely on surviving histories,

$$\tau_i < X_i(h) \qquad \text{for every } i, \tag{12}$$

then first-compromise probability during the mission is zero. If these bounds cover all admitted routes and the construction, handover and operational premises hold, the policy preserves the declared legitimate service and is $(H, \mathcal{A}, \mathcal{B}, 0)$-secure.

A first violation in I_i would occur less than τ_i elapsed time after t_{i-1} while requiring at least $X_i(h) > \tau_i$. Thus its conditional probability on survival is zero. Applying Theorem 2 with every $\epsilon_i = 0$ gives zero mission risk. The construction argument and handover proposition supply the service conclusion under their stated operational premises.

The constant form $\tau < X$ follows from a uniform residual-time lower bound X and windows of duration at most τ. Proactive morphing must maintain that lower bound in each resulting configuration by changing necessary attack dependencies. For modular updates, the analysis includes paths through unchanged components and any continued use of older module versions. Each window inherits the adversarial history accumulated before it.

In the game notation of Appendix E, a feasible policy meeting these conditions has $R_H(D) = 0$ and hence $v^* = 0$.

Work-budget form. A related condition uses aggregate work. After each admissible surviving history at the start of interval i, suppose every continuation producing a first violation in that interval needs at least W_i additional work. If the allowed additional work B_i satisfies $B_i < W_i$ on every such history and interval, no first violation can occur: success would require more work than the budget permits. Both quantities may depend on the retained history, including partially completed attacks and advance knowledge of artifacts.

For a probabilistic guarantee, combine the behavioral and operational conditions in Appendices A and C with Theorem 2 for the same deployment. This gives correct service and the corresponding mission-risk bound.

If different bounds cover modeled attacks, other attack routes and operational failures, a union bound can combine them without assuming independence. A whole-system claim requires explicit coverage of the declared routes, including credentials, shared logic flaws and infrastructure compromise. Unquantified residual routes remain identified as unknown.

E An information-dependent attacker–defender game

The game organizes renewal choices and attacker responses. Use the formal model and contract of Appendix A and the mission, violation, preparation and exposure model of Appendix D. The deployment state includes the current realization and its correspondence to the specification, inventory, pending handover, elapsed time and resource ledgers. It distinguishes genuine and false cues where the threat model admits deception. Each player receives only its declared observations of that state; its observation history and private memory retain acquired information.

E.1 State, observations and information costs

At decision step j, let S_j be the deployment state and a_j^D, a_j^A the players' actions. A transition law K and a joint observation law O specify

$$\begin{aligned} S_{j+1} &\sim K(\cdot \mid S_j, a_j^D, a_j^A), \\ (Y_{j+1}^D, Y_{j+1}^A) &\sim O(\cdot \mid S_j, a_j^D, a_j^A, S_{j+1}). \end{aligned} \tag{13}$$

The observation law can correlate the two players' observations. Each policy uses its own observation history rather than the full state. Costs charge construction, information acquisition, cue generation and response where applicable, with elapsed time and resource ledgers updated in the state. Assume these laws induce a well-defined trajectory distribution for every allowed strategy pair.

Software renewal changes the realization through K; its disclosures also affect O. The defender's knowledge of the model–implementation relation is part of its available information. Any part exposed to the attacker also belongs in the attacker's observations. The necessary-recovery criterion in Appendix B links these observations to information required for compromise.

The examples in Table 1 identify different model inputs. In the location-sufficient case, the attacker already knows a fixed parameter sufficient for the action producing V. Concealment adds no inference requirement unless the action also depends on an observation it changes. Yet the defender's successful response may depend on distinguishing genuine threats from attacker-generated false cues. If generating those cues costs less than the discrimination and response they induce, the information-cost advantage favors the attacker. Flares illustrate the reverse possibility when tracking is necessary for the attack. The action and violation laws determine whose discrimination task matters. Neither the number of decoys nor the number of software variants alone determines the game value.

E.2 Strategies and information

A policy is a nonanticipating, possibly randomized rule from a player's observed history, past actions and memory to its next action. Unobserved opponent state and future randomness are unavailable. The survival indicator used for the proof in Appendix D is available to a player only if its observation model supplies it.

A defender policy D specifies the construction family and its parameters, module or group selection, realization selection, inventory preparation, permitted exposure, and deployment/retirement decisions. An attacker policy A specifies preprocessing, learning, resource allocation, probes, target selection, waiting and attack actions, including admitted attacks that do not require semantic reconstruction. The action protocol and resource limits must exclude unphysical infinite activity in finite time.

Write $\mathcal{A}(\mathcal{B})$ for the allowed attackers under the aggregate budget, including preprocessing and all parallel workers. Write $\mathcal{D}_{\text{feas}}(\mathcal{B}_D)$ for defensive policies meeting the construction, handover,

service and inventory conditions within their own budget. That budget charges model and proof preparation, generation, rejected variants, deployment, runtime overhead and infrastructure. Correctness is relative to $\mathcal{C}$; attack outcomes are assessed through V. Assume both policy sets are nonempty. Defensive budget compliance includes responses to permitted deceptive cues, up to mission termination or first violation. Legitimate-service guarantees retain the contract and operating assumptions above; attack-induced violations are counted through V. Policies that avoid an expensive discrimination task remain available if they meet those requirements. Appendix F gives the cost and inventory model.

E.3 Objective

The defender commits to a policy D whose rule is available to the attacker. The attacker selects an admissible response A with that knowledge. Both policies can adapt during execution. Private random choices remain subject to the observation law.

Let L_H be one if a first violation occurs during the mission and zero otherwise. For the induced trajectory law, define

$$r_H(D, A) = \mathbb{E}_{D,A}[L_H], \qquad R_H(D) = \sup_{A \in \mathcal{A}(\mathcal{B})} r_H(D, A). \tag{14}$$

The payoffs are L_H for the attacker and $-L_H$ for the defender, with costs enforced through the separate budget constraints. The commitment-game upper value is

$$v^*(H, \mathcal{B}_D, \mathcal{B}) = \inf_{D \in \mathcal{D}_{\text{feas}}(\mathcal{B}_D)} \sup_{A \in \mathcal{A}(\mathcal{B})} r_H(D, A). \tag{15}$$

This is the infimum of $R_H(D)$ over feasible policies. The order fixes the defender's rule before taking the attacker's best response; the infimum and supremum allow for unattained optima. First compromise is an absorbing loss for this objective.

The transition and observation laws remain inputs to the framework. Solving it for a particular generator requires specifying how renewal changes attack opportunities and what learning transfers between implementations. The computational asymmetry is represented by those laws and the cost accounts.

E.4 Comparing policies

For a feasible D satisfying Theorem 2, define

$$U(D) = 1 - \prod_{i=1}^{m} (1 - \epsilon_i(D)). \tag{16}$$

Then the definition of v^* and the risk bound give

$$v^* \le R_H(D) \le U(D). \tag{17}$$

An explicit policy with $U(D) \le \varepsilon$ meets the risk target. Each candidate needs a bound valid for its own construction and renewal choices. The policy with the smallest upper bound need not have the smallest actual risk.

A static baseline. Use the same mission, contract, initial-condition model and resource envelopes. Let D_0 be a feasible static policy. Suppose some admissible attacker A_0 satisfies $r_H(D_0, A_0) \ge p_0$. If a feasible renewed policy D_1 has a valid certificate $U(D_1) < p_0$, then

$$R_H(D_1) \le U(D_1) < p_0 \le R_H(D_0). \tag{18}$$

The attack A_0 supplies the baseline lower bound; the guarantee for D_1 covers every allowed response. To compare against the best policy in an entire nonempty static class $\mathcal{D}_{\text{static}}$, a uniform baseline lower bound is needed: for every D_0 in that class, some admissible A_0 must establish the same p_0. With this uniform lower bound, the argument yields $R_H(D_1) < \inf_{D_0 \in \mathcal{D}_{\text{static}}} R_H(D_0)$.

F Renewal rate and cost

F.1 Renewal rate and risk

Suppose $H = m\tau$ and every interval has duration τ. Assume a uniform, history-aware conditional bound has been justified for a declared family of policies D_τ throughout a feasible range of cadences:

$$\epsilon(\tau) \le c\tau^\beta, \qquad c > 0, \quad \beta > 0. \tag{19}$$

The constants and units belong to a particular attack and deployment model, with the other policy choices fixed or specified by the family.

Corollary 3 (When the union bound benefits from shorter windows)**.** *Under Equation* (19) *and Theorem 2, the mission-risk bound is at most*

$$\min\{1, cH\tau^{\beta-1}\}. \tag{20}$$

The uncapped expression decreases with shorter τ when $\beta > 1$, is constant when $\beta = 1$, and increases when $\beta < 1$.

Proof. Substitute $m = H/\tau$ and $\epsilon(\tau) \le c\tau^\beta$ into the sum bound, then inspect the exponent of τ. □

Within the justified, feasible cadence range, per-window success must shrink sufficiently quickly relative to the number of windows. The comparison concerns the upper bound; actual risk may follow a different curve. Leakage, transfer and rollback affect the validity of the assumed scaling.

As a hypothetical numerical example, one day of five-second intervals gives $m = 17{,}280$. A uniform conditional bound of 10^{-9} implies mission risk at most 1.728×10^{-5}. A bound of 10^{-4} yields a product upper bound of about 0.822, insufficient to certify a low-risk objective.

F.2 Defender cost and throughput

In a simplified single-stream cost account, let C_{setup} be model development and family-proof cost, c_{gen} the average cost of constructing and validating an accepted variant, and c_{switch} the average cost of deploying and retiring it. For m accepted variants, the amortized preparation and switching cost is

$$\bar{c}_D = \frac{C_{\text{setup}}}{m} + c_{\text{gen}} + c_{\text{switch}}. \tag{21}$$

Rejected variants and failed switches must be charged to the averages; runtime overhead, observation processing, responses to deceptive cues and infrastructure costs are additional. Feasibility requires this amortized cost to fit the defensive budget. Comparison with attacker cost uses the declared resource conventions.

Let N_0 be the initial usable inventory and $G(t)$ the number of additional usable variants prepared by time t. If $C(t)$ counts consumption, feasibility requires

$$C(t) \le N_0 + G(t) \qquad \text{throughout the mission.} \tag{22}$$

Fleet-specific diversity changes consumption. Reusing an old artifact may reduce supply requirements but can restore attack transfer, so its effect on attack transfer must be included in the model.

For modular renewal, an accepted variant can be a validated configuration update. Generation and switching averages then charge the changed group, compatibility checks and any shared-state migration. The declared inventory model must cover the groups required by the policy.

The engineering problem is to choose a construction and cadence that meet the behavioral, risk, supply and resource constraints together. In the game formulation, these constraints determine which policies belong to $\mathcal{D}_{\text{feas}}(\mathcal{B}_D)$, and a validated cadence bound can supply $U(D_\tau)$. Defensive change must be affordable under the defender's budget while the residual attack opportunity remains inadequate under the attacker's budget. The computational gap becomes an operational advantage when these constraints produce lower mission risk.

G Attack measurements and inference

An empirical evaluation should retain the attacker's knowledge across variants and include family-level attacks, pre-activation artifact access, parallelism and multiple reachable versions. Report time to first useful compromise, full-mission success, aggregate compute and tool cost, and the fraction of analysis reused. Characterize lower-tail successful attack times as well as average effort.

The defender-side account includes construction and proof-checking cost, rejected variants, runtime and tail latency, handover duration, availability and recovery. Where the model admits deceptive observations, also measure discrimination work and the cost of unnecessary responses. Keep aggregate computation distinct from elapsed time and monetary cost. Establish compromise through target-side evidence of the defined violation. Record unsuccessful attempts and timeouts with their elapsed effort and stopping rule.

For $n > 0$ independent, identically distributed mission trials with zero observed successes, a one-sided binomial upper confidence bound is

$$p_{\text{upper}} = 1 - \alpha^{1/n} \tag{23}$$

at confidence $1 - \alpha$, for $0 < \alpha < 1$. It follows by solving $(1 - p)^n = \alpha$. This bounds an empirical mission-success parameter under the sampling assumptions and the tested attack policy. Adaptively changing or dependent trials require a different analysis. A defense selected after inspecting attack results needs validation that accounts for that selection. Theorem 2 requires separately justified bounds uniform over admitted attacker strategies and surviving histories.

An application fails its stated premises if a cheap invariant attack bypasses the varied realizations, retained learning defeats the residual-work premise, actual exposure exceeds the modeled window, or the required contract and service limits cannot be maintained. Such findings require revising the mechanism or its stated applicability while retaining the declared attack coverage.

H Implemented mechanisms and reference components

This catalogue records what each mechanism does, which source and tools are available, and what evidence supports its use. Repository documentation, license information and the commercial pages below were checked on November 23, 2026. The review did not include component installation, builds, proof execution or efficacy tests. Dependencies may have separate licenses.

A deployment using these components needs its own correctness, attack-cost and replacement analysis.

H.1 Formal specifications, refinement and checking

Community Z Tools (CZT). CZT supplies Java tools for parsing, typechecking, transforming and printing Z specifications, plus animation of a restricted subset [7]. It is a concrete open-source starting point for working with formal state and operation specifications. The repository reports GPL-2.0 licensing and a last push in November 2024.

Specware. Specware combines a formal specification language, composition, stepwise refinement and code generation. Its project overview explicitly reports diverse implementations of one specification [12]. The public implementation is BSD-2-Clause licensed. Its README specifies Isabelle2016 for proof support, and the repository last-push date was February 2022. Reuse requires assessing compatibility with that legacy environment.

Fiat. Fiat is a Coq abstract-data-type synthesis library, with specification-to-implementation refinement and Coq-checkable proof trails [13]. Its source LICENSE contains the MIT permission terms. The README reports that most of the library is unmaintained, the SQL-like query-structures target no longer builds, and only the core and parser targets are maintained for Coq's continuous integration.

Software Analysis Workbench (SAW). SAW reasons about formal models of C, Java, Rust and Cryptol programs, supporting implementation/property and equivalence verification [14]. Its public source is BSD-3-Clause licensed, with repository activity in November 2026 and documented binary releases. Each use must state the modeled code, assumptions and properties actually checked.

H.2 AST and IR transformation infrastructure

Abstract syntax trees (ASTs) and compiler intermediate representations (IRs) provide places to express transformations. Different tree or graph structures can implement the same behavior; correctness is assessed through the semantic relation to the specification.

Clang LibTooling. LibTooling provides an open-source framework for standalone Clang frontend tools, with AST access and compilation-database support [15]. LLVM uses Apache-2.0 with LLVM exceptions; its repository was active at the review date.

Alive2. Alive2 provides bounded LLVM transformation/refinement checking, a symbolic executor, SMT abstractions and translation-validation tools [16]. Its source is MIT licensed and had November 2026 activity. Its primary paper explains that bounded loop unrolling can miss bugs; success within the checked bounds is not an unbounded preservation proof. The README explicitly excludes interprocedural transformations and warns of spurious counterexamples with such passes. Validation must respect the tool's supported semantics and scope.

CROW. CROW implements enumerative synthesis of selected LLVM computation fragments, SMT equivalence checks and generation of WebAssembly variants [19]. The fragment model has a narrower memory scope than a complete program specification. The public implementation

is in the SLUMPs repository, now under ASSERT-KTH; its README describes a modified Souper toolchain and a Python 3.7-era environment. The package metadata declares MIT, but complete licensing coverage of this multi-tool composition was not established in this review. The repository last-push date was May 2023. The paper measures binary and execution-trace diversity.

DivCon. DivCon generates assembly variants by solving a mathematical model of compiler choices, including register allocation, instruction selection and scheduling [20]. Its public implementation is based on Unison. The license file specifies BSD-3-Clause for DivCon and identifies separately licensed included code; the repository last push was November 2024. Its evaluation measures gadget survival for code-reuse attacks.

Verified structural transformations. Coq-verified control-flow flattening and mixed Boolean-arithmetic rewriting establish that nontrivial obfuscation can carry preservation proofs [23, 24]. Lu and colleagues also formalize selected transformations while distinguishing correctness from other protection goals [25]. Distorted-interpreter specialization provides semantic generation with potency relative to specified analyses [18].

H.3 Evolution and live-renewal precedents

Program evolution. Cohen's 1993 paper, available in an author-hosted version dated 1992, connects equivalent-program evolution, structural complexity and computational advantage [22]. It describes changes at installation, startup and during operation, and an experimental source-template evolver. It distinguishes the number of possible variants from the difficulty of a successful attack.

ASPIRE. ASPIRE combines a priori same-semantics variants, structural obfuscation, diversified code blocks and server-driven renewal [26]. Its white-box cryptography evaluation includes renewal schedules of 1–5 seconds. Compatible block sets provide an implemented modular-renewal precedent.

The central framework repository uses BSD-3-Clause terms for its own files and explicitly gives submodules their separate licenses. Its last push was April 2021. Some proprietary components were not released, and present-day toolchain compatibility requires assessment. The reported correctness checks use testing. A formal proof of the composed pipeline is not reported. Professional penetration tests covered non-renewed protections; later internal mitigation checks have limited disclosure.

Shuffler. Shuffler demonstrates continuous code re-randomization with a reported 50 ms period and 14.9% SPEC CPU overhead, while imposing an attacker deadline [31]. The prototype targets code-reuse attacks and offloads shuffling to another CPU core. Attacks on address-space-randomization schemes further show why attack strategy and resources must be included in the protection argument [32].

H.4 Commercial mechanisms

Table 2 summarizes current vendor descriptions. Product efficacy was not independently tested in this review.

Offering	Documented mechanism	When changes occur
RunSafe Protect [27]	Granular function relocation to frustrate memory-based code reuse.	Load-time Function Randomization on each software run.
Morphisec [28]	Automated Moving Target Defense through application-memory morphing.	The homepage FAQ specifies load-time morphing and gives no recurring renewal interval.
Promon Code Protect [29]	Post-compile structural obfuscation, encryption and integrity controls.	Build/release protection with runtime checks.
Digital.ai Application Security [30]	Post-build hardening, tamper resistance and Runtime Application Self-Protection.	Hardening after the build; configured responses during execution.

Table 2: Commercial reference mechanisms, as described by their vendors on November 23, 2026.

ASPIRE's authors also report historical commercial live-video systems renewing decompression and decryption code multiple times per second [26]. This is the authors' account of a historical deployment; it does not include a formal correctness proof.

Related patent disclosure. Cisco's patent on LLM-generated polymorphic malware samples describes behavior-guided variant generation for detection, including changed sequence and implementation [21]. The author is a named co-inventor; Cisco Technology Inc. is the recorded assignee.

I Evidence on agentic coding and repair

The studies below measure agent performance and resource use. The ML-engineering study also features in Kiraa's October 18, 2026 video *Agentic Coding Scam* [5]. That video provides commentary; the empirical findings presented here come from the primary papers.

ML-engineering harness comparison. Brilliantov and colleagues compare a minimal autonomous agent, Malena, with elaborate harnesses under matched model, hardware and time conditions [4]. The main comparison covers 30 MLE-bench tasks, with additional NatureBench evaluation. The headline GLM 5.2 result is a 62.5% medal rate versus 47.1% for the best external harness. The authors report limited statistical power, possible benchmark contamination, limited coordination experiments and possible tuning asymmetry.

The paper reports modeled GLM 5.2 costs of $12.12 for Malena and $1.95 for AiScientist, driven by accumulated context reads and a uniform assumed cache discount. The experiments on additional ML validation guidance and coordination measure effects on benchmark performance. They do not evaluate authorization enforcement, vulnerability closure or safe production deployment.

Longitudinal software-repair study. The ML paper cites Ben Sghaier and colleagues' controlled study of 35 Qwen Code releases [3]. It uses Qwen3-Next-80B-A3B-Instruct, 50 sampled SWE-bench Verified tasks, two runs per task and a 600-second task limit. Across releases, 52–94% of tasks receive a non-empty patch, while 23–39% meet the test-based resolution criterion. No statistically significant monotonic improvement in resolution is detected. Reported mean token use rises from approximately 391,000 for the first nine releases to 668,000 for the latest releases.

The controlled results concern one harness/model combination and a benchmark subset. Two runs per task provide limited evidence about run-to-run variation. The trend test leaves uncertainty about changes too small to detect and changes that are not monotonic. Neither study measures production-security failure rates or compares agent repairs with human repairs. Applying the checks in Section 2 requires evidence from the system being changed.

References

[1] A. Blum, K. Vanderlee, J. Xia, and D. Mainor. *GTIG AI Threat Tracker: Agentic Threat Activity Crosses a Significant Threshold*. Google Threat Intelligence Group, November 16, 2026. https://cloud.google.com/blog/topics/threat-intelligence/agentic-threat-activity-crosses-a-significant-threshold.

[2] C. J. Moses. *Chinese-nexus actor's Cairn AI agents found a zero-day on their own*. AWS Security Blog, November 20, 2026. https://aws.amazon.com/blogs/security/chinese-nexus-actors-cairn-ai-agents-found-a-zero-day-on-their-own/.

[3] O. Ben Sghaier, H. Li, B. Adams, and A. E. Hassan. *Don't Blame the Large Language Model: How Agent Harness Evolution Shapes Coding Agent Quality*. arXiv:2607.03691, version 2, July 20, 2026. https://arxiv.org/abs/2607.03691v2.

[4] K. Brilliantov, A. Hernández-Cano, and E. Abbé. *How Much of a Harness Does a Strong Agent Need for Autonomous ML Engineering?* arXiv:2609.40303, version 1, submitted September 30, 2026; PDF title page dated October 1, 2026. https://arxiv.org/abs/2609.40303v1.

[5] Kiraa (`@kiraa_ai`). *Agentic Coding Scam*. YouTube, October 18, 2026. Automatic transcript inspected November 23, 2026. https://www.youtube.com/watch?v=dwD3Gd0Cq1w.

[6] J. Jacky. *The Way of Z: Practical Programming with Formal Methods*. Cambridge University Press, 1997. Author's book page, Chapter 4 excerpt, "What is Z?", and Appendix A glossary, accessed November 23, 2026: https://staff.washington.edu/jon/z-book/. https://staff.washington.edu/jon/z-book/what-is-z.html. https://staff.washington.edu/jon/z/glossary.html.

[7] Community Z Tools project. *Community Z Tools*: repository, README and license metadata. Accessed November 23, 2026. https://github.com/community-z-users/czt.

[8] M. Stuart. *Airmen supply munitions, ensure mission success*. 86th Airlift Wing Public Affairs, Ramstein Air Base, October 10, 2014. https://www.ramstein.af.mil/News/Features/Display/Article/727084/airmen-supply-munitions-ensure-mission-success/.

[9] N. Immerman. *Computability and Complexity*. Stanford Encyclopedia of Philosophy, substantive revision October 18, 2021; accessed November 23, 2026. https://plato.stanford.edu/entries/computability/.

[10] R. Baldoni, E. Coppa, D. C. D'Elia, C. Demetrescu, and I. Finocchi. *A Survey of Symbolic Execution Techniques*. ACM Computing Surveys 51(3), 2018. Author preprint, version 3: https://arxiv.org/abs/1610.00502v3.

[11] CompCert project. *The CompCert C Compiler User's Manual*, Chapter 1, especially Section 1.2. Accessed November 20, 2026. https://compcert.org/man/manual001.html.

[12] Y. V. Srinivas and J. L. McDonald. *The Architecture of Specware, a Formal Software Development System*. Kestrel Institute technical report KES.U.96.7, August 1996. https://www.kestrel.edu/research/specware/papers/specware/specware-arch.pdf. See also the project overview and source repository, accessed November 23, 2026: https://www.kestrel.edu/research/specware/. https://github.com/KestrelInstitute/Specware.

[13] B. Delaware, C. Pit-Claudel, J. Gross, and A. Chlipala. *Fiat: Deductive Synthesis of Abstract Data Types in a Proof Assistant*. Proceedings of ACM POPL, 2015. https://doi.org/10.1145/2676726.2677006. Author-hosted full paper: https://adam.chlipala.net/papers/FiatPOPL15/FiatPOPL15.pdf. Source, maintenance notice and LICENSE, accessed November 23, 2026: https://github.com/mit-plv/fiat.

[14] Galois and SAW contributors. *Software Analysis Workbench*: source repository, README, verification documentation and license metadata. Accessed November 23, 2026. https://github.com/GaloisInc/saw-script.

[15] LLVM project. *LibTooling*, LLVM source repository and `llvm/LICENSE.TXT`. Accessed November 23, 2026. https://clang.llvm.org/docs/LibTooling.html. https://github.com/llvm/llvm-project.

[16] N. P. Lopes, J. Lee, C.-K. Hur, Z. Liu, and J. Regehr. *Alive2: Bounded Translation Validation for LLVM*. Proceedings of ACM PLDI, 2021. https://web.ist.utl.pt/nuno.lopes/pubs/alive2-pldi21.pdf. See also the source repository, README and license metadata, accessed November 23, 2026: https://github.com/AliveToolkit/alive2.

[17] M. Dalla Preda and R. Giacobazzi. *Semantics-based Code Obfuscation by Abstract Interpretation*. Journal of Computer Security 17(6), pp. 855–908, 2009. https://profs.scienze.univr.it/~giaco/download/JCS09.pdf.

[18] R. Giacobazzi, N. D. Jones, and I. Mastroeni. *Obfuscation by Partial Evaluation of Distorted Interpreters*. Proceedings of ACM PEPM, pp. 63–72, 2012. https://profs.scienze.univr.it/~giaco/download/pepm12-giacobazzi.pdf.

[19] J. Cabrera Arteaga, O. Floros, O. Vera Perez, B. Baudry, and M. Monperrus. *CROW: Code Diversification for WebAssembly*. Workshop on Measurements, Attacks, and Defenses for the Web, 2021. https://doi.org/10.14722/madweb.2021.23004. https://arxiv.org/abs/2008.07185. Implementation README and package metadata, accessed November 23, 2026: https://github.com/ASSERT-KTH/slumps/tree/master/crow.

[20] R. M. Tsoupidi, R. Castañeda Lozano, and B. Baudry. *Constraint-Based Software Diversification for Efficient Mitigation of Code-Reuse Attacks*. Preprint, 2020. https://arxiv.org/abs/2007.08955. Implementation, README and license file, accessed November 23, 2026: https://github.com/romits800/divCon.

[21] V. Parla, A. Zawadowskiy, B. Anderson, H. M. Latapie, and O. Bessonov. *LLM Technology for Polymorphic Generation of Samples of Malware for Future Malware Detection*. US Patent 12,682,060 B2, July 14, 2026. Recorded assignee: Cisco Technology Inc. https://patents.google.com/patent/US12682060B2/en. Corresponding application US20250363215A1, published November 27, 2025.

[22] F. B. Cohen. *Operating System Protection Through Program Evolution*. Computers & Security 12(6), pp. 565–584, 1993. Author-hosted version copyright 1992: https://all.net/books/IP/evolve.html.

[23] S. Blazy and A. Trieu. *Formal Verification of Control-Flow Graph Flattening*. Proceedings of ACM CPP, pp. 176–187, 2016. https://doi.org/10.1145/2854065.2854082.

[24] S. Blazy and R. Hutin. *Formal Verification of a Program Obfuscation Based on Mixed Boolean-Arithmetic Expressions*. Proceedings of ACM CPP, pp. 196–208, 2019. https://doi.org/10.1145/3293880.3294103.

[25] W. Lu, B. Sistany, A. Felty, and P. Scott. *Towards Formal Verification of Program Obfuscation*. IEEE European Symposium on Security and Privacy Workshops, pp. 635–644, 2020. https://doi.org/10.1109/EuroSPW51379.2020.00091. https://www.site.uottawa.ca/~afelty/dist/sad20.pdf.

[26] B. Abrath, B. Coppens, J. Van den Broeck, B. Wyseur, A. Cabutto, P. Falcarin, and B. De Sutter. *Code Renewability for Native Software Protection*. ACM Transactions on Privacy and Security 23(4), article 20, 2020. https://doi.org/10.1145/3404891. https://arxiv.org/abs/2003.00916. Public framework, README and COPYRIGHT, accessed November 23, 2026: https://github.com/aspire-fp7/framework.

[27] RunSafe Security. *RunSafe Protect*. Product mechanism description, accessed November 23, 2026. https://runsafesecurity.com/platform/protect/.

[28] Morphisec. *Morphisec*: platform overview and homepage AMTD FAQ. Accessed November 23, 2026. https://www.morphisec.com/.

[29] Promon. *Promon Code Protect*. Product mechanism description, accessed November 23, 2026. https://promon.io/products/promon-code-protect.

[30] Digital.ai. *Application Security*. Product mechanism description, accessed November 23, 2026. https://digital.ai/products/application-security/.

[31] D. Williams-King et al. *Shuffler: Fast and Deployable Continuous Code Re-Randomization*. Proceedings of USENIX OSDI, pp. 367–382, 2016. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/williams-king.

[32] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh. *On the Effectiveness of Address-Space Randomization*. Proceedings of ACM CCS, pp. 298–307, 2004. https://hovav.net/ucsd/papers/sppgmb04.html.

[33] M. van Dijk, A. Juels, A. Oprea, and R. L. Rivest. *FlipIt: The Game of "Stealthy Takeover"*. Cryptology ePrint Archive, Paper 2012/103, 2012. https://eprint.iacr.org/2012/103.

[34] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. *On the (Im)possibility of Obfuscating Programs*. Extended abstract, 2001; revised full version, 2010. https://www.wisdom.weizmann.ac.il/~oded/p_obfuscate.html.