Cyber Agency Is Stack-Level, Not Model-Level

From Cyber-Capable Models to Scoped, Verified Autonomous Security Outcomes

Draft: 2026-05-08

Author: Migel Tissera

Product: Drost

Status: v0.9 technical-peer review draft

Abstract

Frontier AI models are becoming materially cyber-capable. The UK AI Security Institute's evaluation of an early GPT-5.5 checkpoint showed a second frontier model reaching the same general tier as Claude Mythos Preview on advanced cyber tasks and long-horizon cyber ranges. GPT-5.5 achieved a 71.4% average pass rate on expert-level cyber tasks, solved a custom-VM reverse-engineering challenge in 10 minutes and 22 seconds using a basic ReAct scaffold with Bash and Python in Kali, and completed the 32-step "The Last Ones" corporate network range end-to-end in 2 of 10 attempts.

Those results matter, but they do not by themselves define the product layer for autonomous security.

This paper argues that autonomous cyber performance is determined by the full agent stack, not by the model alone. Model capability is necessary. It is not sufficient. Practical cyber agency depends on the harness around the model: scoped execution, durable state, authenticated context, tool boundaries, proof validation, redaction, safety policy, model routing, and cost control.

Drost is built around that claim. It converts cyber-capable models into authorized, auditable, proof-backed security outcomes. The product does not rely on an unbounded model improvising against a target. It constrains the model inside a runtime that owns scope, policy, tool execution, state, proof, and reporting boundaries.

The core claim is simple:

> **Cyber capability is becoming model-distributed. Cyber agency is stack-level.**

---

1. Scope, Authorization, and Safety Model

Drost is designed for authorized security testing under explicit scope and rules of engagement. The system is intended for defenders, security teams, and authorized operators evaluating applications, APIs, and exposed attack surface.

That constraint is architectural, not cosmetic. Drost is organized around:

- explicit engagement scope;
- rules-of-engagement gates before active testing;
- runtime authorization for tool actions;
- managed authentication contexts rather than raw secret exposure;
- durable action logs and proof artifacts;
- negative-control validation for false-positive restraint;
- destructive-action blocking unless explicitly permitted;
- redaction at the proof boundary;
- separation between candidate observations and confirmed findings.

This matters because autonomous offensive capability without scope, proof, and safety boundaries is not a product. It is a liability. A useful cyber agent must answer more than "can this be exploited?"

It must also answer:

- Was this action authorized?
- Which identity or session was used?
- Which evidence produced the hypothesis?
- Which tool action confirmed or rejected it?
- Is the proof reproducible and safe to share?
- Did the system avoid reporting vulnerabilities where none were present?

The rest of this note should be read in that frame: authorized testing, explicit boundaries, auditable evidence, and conservative confirmation.

---

2. Why Model Capability Is Not Cyber Agency

A model evaluation asks a bounded question:

> **Given a model, a scaffold, a task, and a token budget, how far can the model get?**

A production autonomous security system has to answer a harder set of questions:

- Can it operate inside a real scope boundary?
- Can it distinguish a lead from a confirmed vulnerability?
- Can it preserve useful state over a long engagement?
- Can it use authenticated context safely?

- Can it avoid hallucinating findings on clean systems?
- Can it produce proof artifacts that a human reviewer can trust?
- Can it route work across model lanes without losing accountability?
- Can it produce customer-useful output at viable unit economics?

Those questions are not solved by model weights alone. They require an agent stack.

The Drost thesis is that practical autonomous security outcomes come from the interaction of:

```
model x harness x tools x state x auth x proof x safety x cost
```

A stronger model in a thin harness can underperform a weaker or cheaper model in a stronger harness. A model that performs well on cyber tasks may still fail to convert authenticated application state into customer-grade proof. A model that follows tool-use protocol cleanly may still be a weak autonomous tester if it over-reads state, delays action, or reports unproven candidates as findings.

The product layer is therefore not "a better prompt around a model." It is an architecture for converting model capability into scoped, verified cyber outcomes.

---

3. What Drost Does

Drost is a security-agent system for turning authorized scope into verified findings.

At a high level, the workflow is:

```
authorized organization or target
  -> scope and rules-of-engagement policy
  -> attack-surface discovery
  -> stateful planning and tasking
  -> scoped tool execution
  -> proof validation
  -> confirmed finding or rejected candidate
  -> redacted customer-facing evidence
```

The important partition is:

> **Runtime owns mechanics. The model owns judgment.**

The model helps prioritize, interpret, and choose between plausible paths. The runtime owns the parts that should not be left to free-form generation: scope checks, action policy, tool invocation, evidence storage, auth-context handling, replay constraints, redaction, and proof status.

This partition is the difference between an autonomous system that can be evaluated and a chatbot that happens to have security tools.

---

4. Capability Surface

Drost's public product capabilities can be described without exposing internal implementation details.

Capability	What it does	What the customer gets
Scope-gated execution	Every active action is checked against engagement scope and rules of engagement	Confidence that testing remains inside authorized boundaries
Organization-level discovery	Domain and organization seeds can become scoped attack-surface inventory	A structured view of externally visible assets worth testing
Application and API testing	Web, API, and authenticated workflows are tested through controlled tool lanes	Candidate and confirmed findings tied to affected surfaces
Browser-assisted evidence	Browser automation captures UI/session evidence where HTTP-only testing is insufficient	Screenshots, request traces, storage-state evidence, and auth-flow context
Replay-backed proof	Security leads are confirmed only when deterministic proof or equivalent validation establishes impact	Reproducible evidence rather than scanner-style suspicion
Auth-context lifecycle	Sessions and credentials are referenced, validated, and protected by the runtime	Safer authenticated testing with less raw-secret exposure
Negative-control discipline	Clean or low-surface controls are used to measure false-positive restraint	A quality signal beyond raw finding count
Redacted reporting	Proof artifacts are shaped for review without leaking credentials or unnecessary sensitive data	Evidence a security team can share and act on

The key product distinction is not that Drost "uses a browser" or "uses tools." Many systems can do that. The distinction is that Drost treats observations, actions, and proof as separate states with different trust levels.

---

5. Candidate Observations vs. Confirmed Findings

Most automated security tools overproduce observations. They can report missing headers, reflected parameters, version strings, exposed routes, and suspicious request shapes. Some of those observations matter. Many do not.

Drost separates observations from findings.

A **candidate observation** is a plausible lead. It may come from discovery, browser evidence, HTTP behavior, API structure, identity/session context, or model interpretation.

A **confirmed finding** requires proof. Confirmation may come from deterministic replay, safe exploitation output, authenticated role-differential behavior, validated component impact, or equivalent artifact-backed evidence.

That distinction is load-bearing:

```
observation -> candidate -> proof attempt -> confirmed finding or rejected lead
```

Without that boundary, autonomous systems can appear productive by inflating low-signal output. With that boundary, the system can be measured by the harder question:

> **Which claims survived proof?**

That is the standard Drost is designed around.

---

6. Browser Observes. Replay Proves.

Modern application security often depends on browser-shaped evidence:

- UI controls that appear only for certain roles;
- hidden or readonly fields;
- client-side state;
- JavaScript-generated requests;
- session storage and token handling;
- multi-step workflows;
- browser-set headers and cookies.

Browser automation can observe those things. Observation is valuable, but it is not proof.

Drost treats browser evidence as a source of leads. A browser artifact becomes security-relevant when it can be tied to a safe proof path. For authorization and parameter-tampering classes, that proof often requires replaying a request or typed mutation under a valid, in-scope auth context and comparing the result.

The principle is:

> **Browser observes. Replay proves.**

A browser-only artifact such as "admin saw this control" or "this hidden field appeared in a request" should not automatically become a confirmed vulnerability. It becomes a lead. It becomes a finding only when proof establishes impact under the engagement's rules.

Customer-facing evidence should look like:

> **A lower-privileged session replayed a higher-privileged action and received an unauthorized success response, with redacted request/response proof.**

That is materially different from:

> **The model thinks this endpoint might have an access-control issue.**

---

7. Why App-Layer Security Differs From CTFs and Cyber Ranges

Cyber tasks and ranges are valuable because they test real skills: reverse engineering, exploit development, tool use, chaining, and long-horizon autonomy. But application-layer security has a different operational shape.

Modern web and API testing often depends on:

- authenticated sessions;
- role-specific behavior;
- object-level authorization;
- hidden or readonly parameters;
- token reuse and expiry;
- browser-driven workflows;
- API schemas;
- business-logic invariants;
- replay under multiple identities;
- safe proof suitable for client reporting.

A model may solve a difficult reverse-engineering challenge and still underperform on application-layer objective conversion if the harness does not manage auth state, roles, request context, replay, and proof boundaries.

Conversely, a less expensive model may perform well when the harness externalizes enough method and constrains the action space.

This is why Drost treats application-layer testing as a system problem. The model is the reasoning engine. The system around it supplies memory, tools, scope, state, auth, proof, and cost discipline.

---

8. What Internal Evaluation Has Shown

Drost has been evaluated internally across intentionally vulnerable applications, API-heavy targets, role-based synthetic fixtures, known-vulnerability fixtures, and clean negative controls. Those evaluations are not yet an independently reproduced public benchmark, and this note does not present them as one.

The useful lessons are qualitative and architectural:

1. **Harness design changes model performance.** The same model behaves differently when it has durable state, explicit tasking, structured evidence, and proof gates.
2. **Raw finding count is a weak metric.** Systems can inflate counts with low-value observations. Confirmed impact, severity, proof quality, and false-positive restraint matter more.
3. **Authenticated context changes the work.** Many high-value application findings appear only after the system captures, validates, and safely reuses session context.
4. **Browser evidence is useful but insufficient alone.** UI/session artifacts become valuable when they feed a proof path.
5. **Negative controls are essential.** A system that finds more issues but invents findings on clean targets is not better.
6. **Model routing is a product lever.** Different model lanes can be useful for different work types, but routing only matters when the runtime preserves accountability and proof.

The conclusion from internal evaluation is not "one model wins." It is that a stronger agent stack can change the practical usefulness, cost, and reliability of cyber-capable models.

---

9. Product Implications

9.1 Evaluation Should Measure Systems, Not Just Models

Public model evaluations remain essential. They show what models can do under controlled conditions. But security buyers and defenders also need system evaluations:

- model plus harness;
- model plus tool substrate;
- model plus auth context;
- model plus proof pipeline;
- model plus safety boundary;
- model plus cost model.

The same model can behave differently across harnesses. The same harness can change the practical ranking of models. Evaluating only the model misses the layer where product outcomes are created.

9.2 Proof Quality Will Matter More Than Volume

As model capability diffuses, the strongest security-agent products will not be the ones that produce the longest report. They will be the ones that produce trustworthy evidence.

The key question becomes:

> **Can the system show exactly what it did, why it did it, under whose authority, with which**

> **identity, and what proof confirmed the result?**

This is why Drost emphasizes proof source, redaction, negative controls, and candidate-versus-confirmed separation.

9.3 Unit Economics Shape the Product

Autonomous security products cannot ignore cost. Some model lanes may be appropriate for high-value validation, while lower-cost lanes may be better for frequent testing, triage, and regression. A practical system needs routing, budgets, and proof-aware escalation.

The goal is not to use the most expensive model everywhere. The goal is to use the right capability at the right point in the workflow while preserving evidence quality.

9.4 Safety Is a Runtime Property

Safety cannot depend only on the model "choosing well." Scope enforcement, destructive-action gates, auth-context handling, redaction, and proof policy must be runtime constraints.

That is especially important for cyber systems. The model should reason about what to try. The runtime should decide what is allowed, what is recorded, and what counts as proof.

---

10. Limitations and Non-Claims

This note should be read with clear limitations.

Drost is not claiming that:

- internal evaluations are independently validated public benchmarks;
- model evaluations directly predict customer-environment performance;
- vulnerable-app tests fully represent production applications;
- browser evidence alone proves exploitability;
- autonomous testing should run outside explicit authorization;
- every security class can be safely automated today;
- model routing removes the need for human review.

Current limitations include:

1. **Evaluation reproducibility.** Internal evaluations need a public-safe protocol before they can be treated as external evidence.
2. **Target representativeness.** Intentionally vulnerable and synthetic targets are useful for development but do not fully capture production variability.
3. **Real customer validation.** Customer-authorized engagements are the next important validation layer.
4. **Reporting maturity.** Structured proof exists before polished, customer-ready narrative reporting.

5. **Governance.** Preview access, authorization workflow, and deployment controls must remain conservative until the product is hardened.

These limitations do not weaken the central claim. They sharpen it. The right next step is not broader rhetoric. It is more rigorous system-level evaluation under authorization.

---

11. What We Want Technical Peers To Review

This draft is intended for technical peers, not broad marketing distribution. The requested review is specific:

1. Does the stack-level thesis explain a real gap between model capability and useful cyber agents?
2. Is the proof standard credible: observations stay candidates until proof exists?
3. Are the authorization, scope, redaction, destructive-action, and negative-control boundaries strong enough for public discussion?
4. Which product claims should be softened before public release?
5. What evidence would make this argument reproducible enough for third-party review?
6. What should be in a public evaluation protocol?

The paper should be judged as an architecture-and-evaluation argument. It is not a claim that Drost is ready for unsupervised production use, and it is not a universal ranking of cyber-capable models.

---

12. Next Work

The next public-facing research and product steps are:

1. **Publish a public-safe evaluation protocol.** Define target setup, budget, model configuration, scoring, false-positive policy, and artifact requirements without exposing implementation internals.
2. **Separate candidate and confirmed metrics.** Report candidate leads, confirmed findings, severity, and proof source independently.
3. **Expand negative controls.** Use more clean and patched targets to measure false-positive restraint.
4. **Add production-shaped applications.** Evaluate against realistic multi-role, API-heavy, workflow-heavy applications under authorization.
5. **Improve customer-ready reporting.** Turn structured proof into concise, redacted, reviewable finding narratives.
6. **Invite external methodology review.** Technical peer review should focus on scope, safety, proof quality, and reproducibility.

For technical peer review, the highest-value next artifact is not more internal implementation detail. It is a reproducible, public-safe evaluation appendix for one representative authorized run.

Conclusion

AISI's GPT-5.5 evaluation reinforces a larger point: frontier models are becoming meaningfully cyber-capable. Long-horizon reasoning, coding, tool use, and self-correction are producing serious cyber performance across multiple model families.

But cyber-capable models are not the same thing as cyber agents.

The product problem is conversion: turning model capability into scoped, authorized, verified, cost-efficient security outcomes. That requires a full agent stack: harness, tools, state, auth context, proof validation, safety boundaries, reporting, and model routing.

Drost is built around that conversion layer. It is not a better prompt around a model. It is an agent architecture for converting cyber-capable models into scoped, verified, customer-trustworthy security outcomes.

The central claim is therefore:

> **The next frontier in autonomous security is not only better cyber-capable models. It is better cyber-agent architecture.**

Source Notes

Primary external reference:

- UK AI Security Institute, "Our evaluation of OpenAI's GPT-5.5 cyber capabilities," 2026. https://www.aisi.gov.uk/blog/our-evaluation-of-openais-gpt-5-5-cyber-capabilities

Internal Drost evaluation artifacts informed this draft but are intentionally not cited by path or release sequence in the public review version.