Cyber Agency Is Stack-Level, Not Model-Level

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

Draft: 2026-05-11 **Author:** Migel Tissera **Product:** Drost **Status:** v1.4 technical note

Abstract

Frontier AI models are becoming materially cyber-capable. The UK AI Security Institute's evaluation of an early GPT-5.5 checkpoint reported a 71.4% average pass rate on expert-level cyber tasks, a custom-VM reverse-engineering solve in 10 minutes and 22 seconds, and end-to-end completion of "The Last Ones," a 32-step corporate-network range, in 2 of 10 attempts.

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

This note 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 objective derivation x proof validation x safety boundary x cost model
```

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. Representative Evaluation: Proof Efficiency Across Target Classes

To make the argument concrete, Drost ran fixed-target, fixed-stack representative evaluations across four different application-security target classes:

1. **RolePlay:** a synthetic multi-role web application designed to exercise authenticated application testing, browser evidence, role-specific behavior, and replay-backed authorization proof.
2. **crAPI:** a modern intentionally vulnerable API application designed to exercise authenticated API exploitation, object-level authorization, token handling, business logic, and server-side request behavior.
3. **OWASP Juice Shop:** a full-stack intentionally vulnerable web application designed to exercise broad web and SPA-style application-surface conversion.
4. **DrostBench:** a private full-stack benchmark built for this evaluation, with a harness-side oracle of expected vulnerabilities and negative controls. DrostBench is intended to reduce prior-exposure effects from public vulnerable applications and to separate raw finding count from unique oracle coverage.

These are not global cyber benchmarks. They are representative application-layer evaluations of one Drost stack under one budget profile:

- fixed target per run;
- same Drost runtime and proof standard;
- same 5M-token and 60-minute cap per run;
- same candidate-versus-confirmed policy;
- same compact scorecard schema;
- no raw secrets in published artifacts.

The question was not "which model is globally best at cyber?" The question was:

> Under the same Drost stack, how efficiently does each model lane convert early evidence into confirmed, proof-backed outcomes?

The target classes are intentionally different. RolePlay is the cleaner proof-lane target: it measures whether browser-observed role and request evidence becomes replay-backed proof. crAPI is the authenticated API target: it measures whether models use Drost's state, auth context, and tools to convert API behavior into confirmed findings. Juice Shop is

the broad full-stack web target: it measures whether models can traverse a larger known-vulnerable application surface and turn many application behaviors into confirmed outcomes. DrostBench is the private benchmark target: it measures oracle-normalized coverage on an application not intended to be part of common model pretraining or vulnerable-app memorization.

Juice Shop is also the target with the strongest caveat. It is a well-known public vulnerable application and many frontier models may have prior exposure to its challenge patterns. Drost treats it as a broad app-surface benchmark, not as evidence of unseen generalization.

8.1 RolePlay: Replay-Proof Conversion

Model lane	Confirmed findings	Replay proofs	Auth contexts	First confirmed	First replay	Tool calls	Useful-tool ratio
GLM-5.1	19	10	8	140.8s	392.5s	225	0.83
Sonnet-4.6	11	2	4	106.2s	533.8s	144	0.78
Opus-4.7	9	4	9	66.2s	184.2s	191	0.59
Kimi-K2.6	6	2	4	591.9s	1578.3s	97	0.95
GPT-5.5 xhigh	4	2	4	523.8s	769.8s	276	0.58
DeepSeek-V4-Pro	4	2	5	1231.1s	702.6s	112	0.58
Qwen-3.6-Plus	4	0	3	815.5s	n/a	69	0.90

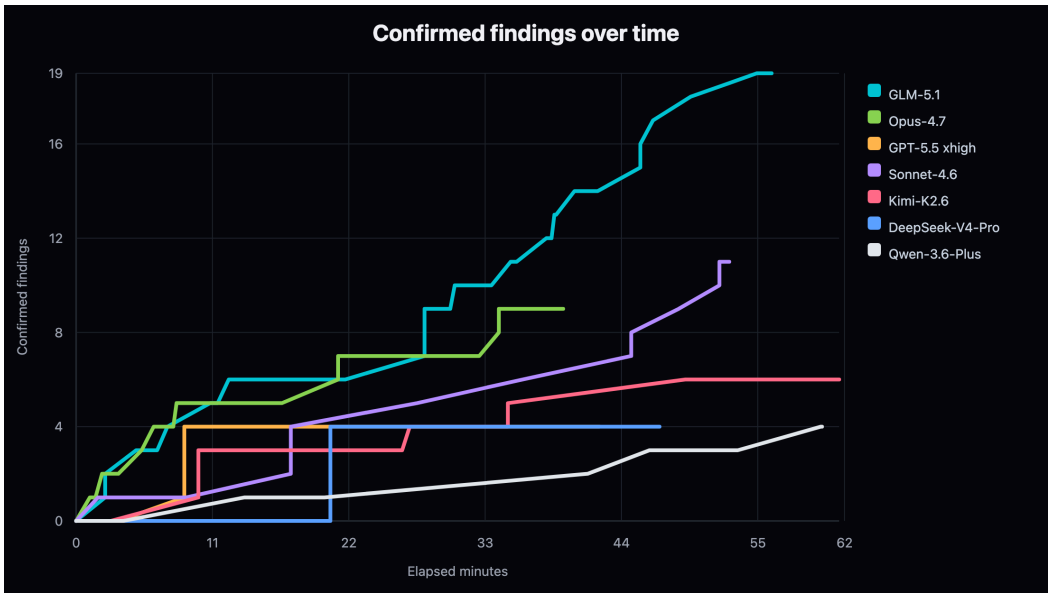

Figure 1. Confirmed findings over time in a representative single-target evaluation. Each model ran under the same Drost stack, budget cap, target reset policy, and proof standard.

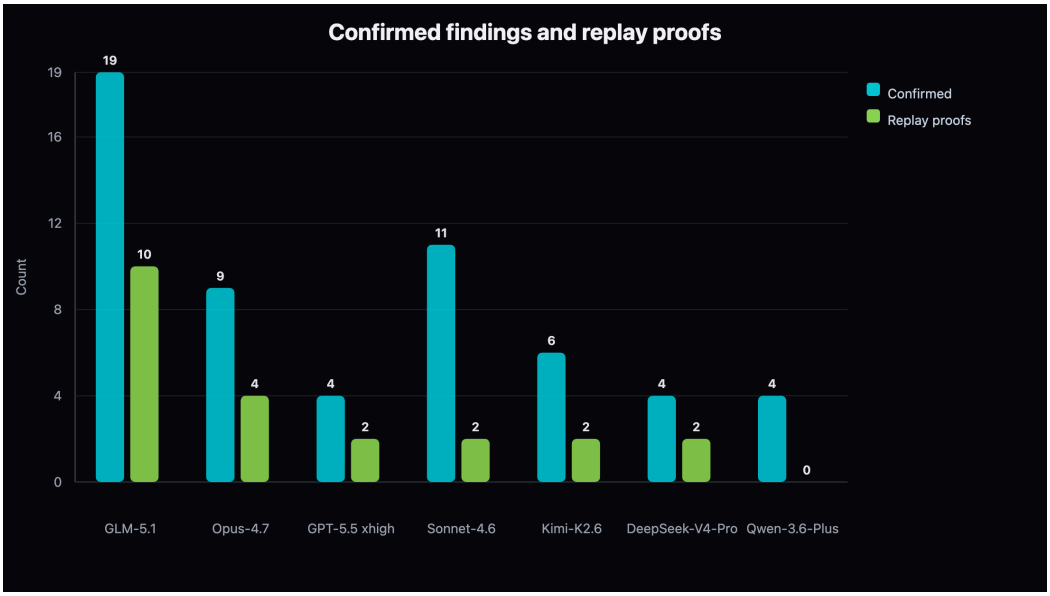

Figure 2. Final confirmed-finding yield and replay-proof yield. Finding count and proof-source mix are intentionally separated.

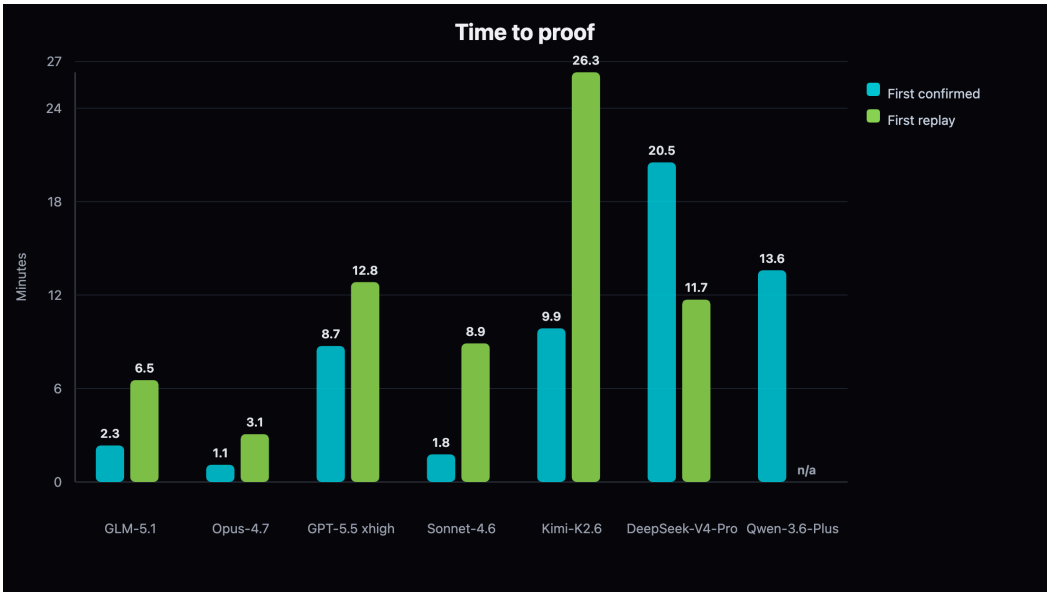

Figure 3. Time to first confirmed finding and first replay proof. Time-to-proof measures when the first verified outcome was produced, not when a model first mentioned a possible issue.

RolePlay shows that replay-proof conversion is model-sensitive even when the runtime and proof path are fixed.

GLM-5.1 produced the highest confirmed yield and replay-proof yield. Opus-4.7 produced the fastest first confirmed result and fastest first replay proof, but did not match GLM-5.1 on total conversion over the hour. Sonnet-4.6 showed a strong middle ground: high confirmed yield, good useful-tool ratio, and clean invalid-auth behavior.

Finding count alone would hide product quality. Qwen-3.6-Plus converted four findings with high tool efficiency, but produced no replay proofs in this run. GPT-5.5 xhigh and DeepSeek-V4-Pro each exercised replay machinery but under-converted relative to GLM-5.1, Opus-4.7, and Sonnet-4.6. A leaderboard that only counts "findings" would miss the proof-source distinction.

Time-to-proof is a better product metric than final count alone. In customer work, an agent that converts high-quality evidence early is more useful than one that spends most of the budget gathering observations without proof. This is especially important when budgets vary across engagements.

8.2 crAPI: Authenticated API Exploit Conversion

RolePlay is not enough by itself. It is proof-path rich by design. Drost also needs to perform on modern API surfaces where the dominant proof source is not replay proof but HTTP behavior under controlled authenticated context.

The crAPI run used the same 5M-token / 60-minute profile and isolated target reset policy. Six model lanes were attempted in sequence: Opus-4.7, GLM-5.1, GPT-5.5 xhigh, DeepSeek-V4-Pro, Kimi-K2.6, and Sonnet-4.6.

Model lane	Wave status	Confirmed findings	Candidate observations	Auth contexts	First confirmed	Tool calls	Useful-tool ratio
GLM-5.1	completed	23	6	1	375.1s	199	0.85
Opus-4.7	completed	19	0	1	225.4s	167	1.03
Sonnet-4.6	completed	11	5	2	494.1s	115	0.83
GPT-5.5 xhigh	completed	8	2	1	226.2s	246	0.48
Kimi-K2.6	completed with shutdown timeout warning	0	0	1	n/a	182	0.30
DeepSeek-V4-Pro	failed before engagement summary	n/a	n/a	n/a	n/a	n/a	n/a

Figure 4. Confirmed findings over time on crAPI. This target measured authenticated API exploit conversion rather than replay-proof conversion.

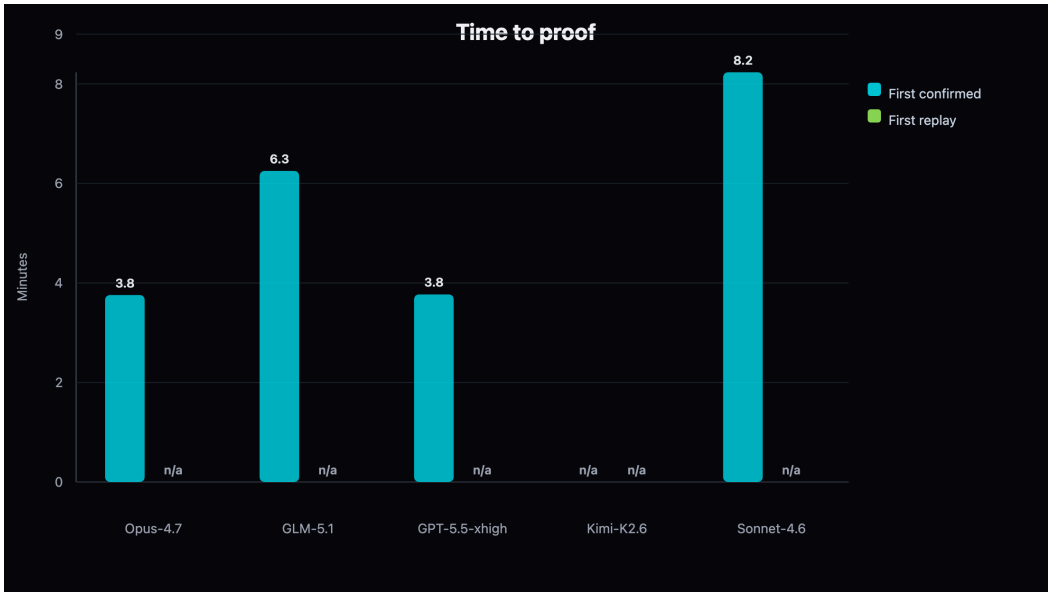

Figure 5. Time to first confirmed finding on crAPI. Opus-4.7 reached first confirmation fastest, while GLM-5.1 produced the highest total confirmed yield.

The crAPI result is useful precisely because it differs from RolePlay.

On crAPI, no replay proofs were recorded. That is not a failure of the replay system. It reflects the target class. crAPI's findings were confirmed through HTTP behavior: authentication bypass, exposed configuration, server-side request behavior, object-level authorization, information disclosure, injection-adjacent behavior, and business-logic weaknesses.

GLM-5.1 again produced the highest confirmed yield, but the shape changed: 23 confirmed findings, 6 candidates, and no replay proofs. Opus-4.7 again produced the fastest first confirmed finding among productive lanes and reached 19 confirmed findings with no candidate backlog. Sonnet-4.6 produced a lower raw count than GLM and Opus, but had the cleanest safety/noise profile in the run. GPT-5.5 xhigh produced early confirmation but under-converted across the hour and generated a higher invalid-auth skip count. Kimi-K2.6 captured an HTTP auth context but exhausted its budget without confirmed or candidate findings. DeepSeek-V4-Pro showed activity in observability, but hit the hard timeout before producing an engagement summary and is excluded from proof-efficiency plots.

8.3 Juice Shop: Broad Full-Stack Web Conversion

The third run used OWASP Juice Shop as a broad full-stack web target. The run used the same 5M-token / 60-minute profile and isolated target reset policy. Six model lanes were attempted in sequence: Opus-4.7, GLM-5.1, GPT-5.5 xhigh, Kimi-K2.6, DeepSeek-V4-Pro, and Sonnet-4.6.

Juice Shop is useful because it changes the evaluation shape again. RolePlay is designed around replay proof. crAPI is API-heavy. Juice Shop is a large, known vulnerable application with a wider mix of web, API, auth, injection, access-control, and configuration behaviors. It is not a clean unseen benchmark, but it does test broad-surface conversion under a fixed Drost stack.

Model lane	Wave status	Confirmed findings	Candidate observations	Auth contexts	First confirmed	Tool calls	Useful-tool ratio
Opus-4.7	completed	70	2	5	127.0s	329	1.05
GLM-5.1	completed	37	4	5	171.7s	169	0.98
GPT-5.5 xhigh	completed	27	2	2	400.9s	200	0.60
Sonnet-4.6	completed	22	1	4	679.3s	132	0.93
DeepSeek-V4-Pro	completed with shutdown timeout warning	10	1	1	731.1s	92	0.91
Kimi-K2.6	failed before engagement summary	n/a	n/a	n/a	n/a	n/a	n/a

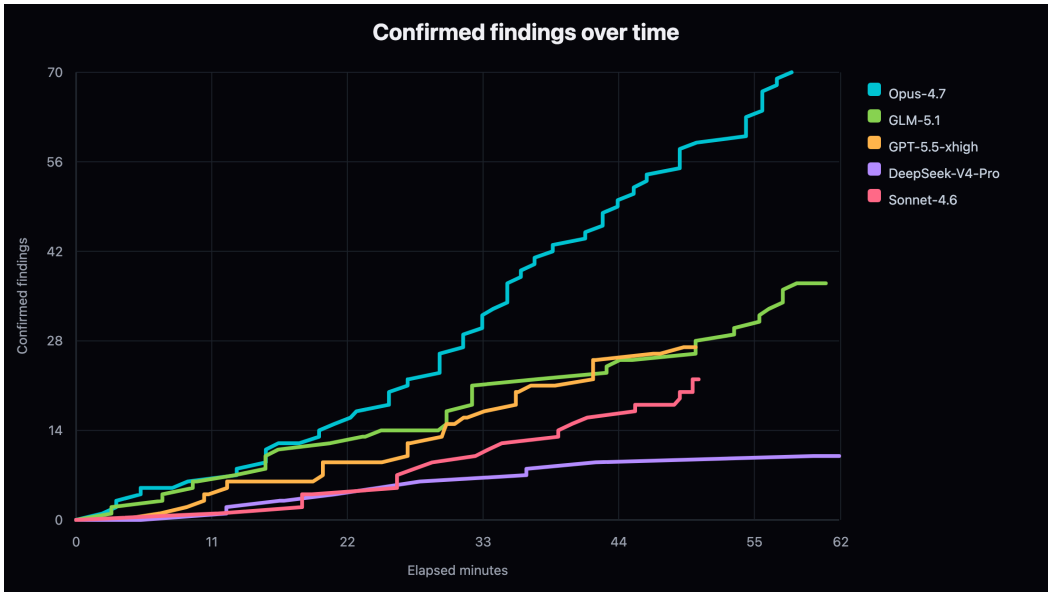

Figure 6. Confirmed findings over time on Juice Shop. This target measured broad full-stack web conversion on a known vulnerable application.

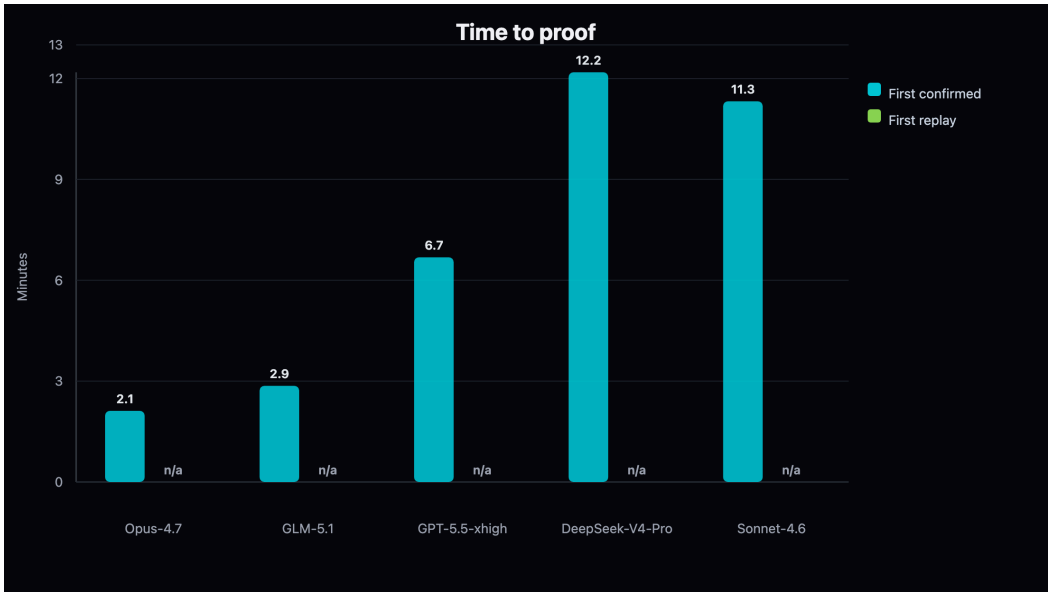

Figure 7. Time to first confirmed finding on Juice Shop. Opus-4.7 led both first-confirmed time and total confirmed yield.

The Juice Shop result is the first target where Opus-4.7 clearly led on both speed and total conversion. Opus reached first confirmation in 127.0 seconds and produced 70 confirmed findings. GLM-5.1 remained strong and completed cleanly with 37 confirmed findings, but it did not lead this target. GPT-5.5 xhigh completed cleanly with 27 confirmed findings, though it generated more invalid-auth and destructive-action skips than the strongest lanes. Sonnet-4.6 completed cleanly with 22 confirmed findings and a high useful-tool ratio. DeepSeek-V4-Pro produced a usable scorecard before shutdown timeout. Kimi-K2.6 hit the hard timeout before writing an engagement summary and is excluded from proof-efficiency plots.

The important point is not that Juice Shop proves one model is globally better. It does not. The important point is that target shape changes the practical model-routing question. On this broad, known app-surface target, Opus-4.7's exploration and conversion behavior produced a materially different result from the RolePlay and crAPI rankings.

8.4 DrostBench: Private Oracle-Normalized Coverage

Public vulnerable applications are useful for development, but they carry a prior-exposure caveat. A model may already know common challenge names, routes, payloads, or expected exploit patterns.

To reduce that effect, DrostBench was built as a private full-stack benchmark for this evaluation. It includes multi-role authentication, cookie and JWT behavior, API workflows, business-logic tampering, access-control flaws, injection behavior, SSRF, file access, and negative controls. The scoring oracle is harness-side and is not served by the application.

The DrostBench run used the same 5M-token / 60-minute profile for six completed model lanes: Opus-4.7, GLM-5.1, GPT-5.5 xhigh, Sonnet-4.6, Kimi-K2.6, and DeepSeek-V4-Pro. A Qwen lane was started and stopped before completion, then excluded. Unlike the raw proof-efficiency scorecard, the DrostBench oracle score de-duplicates repeated reports of the same root issue and only credits findings that map to one of the 12 oracle vulnerabilities.

Model lane	Wave status	Oracle coverage	Raw confirmed findings	Matched confirmed events	Duplicate oracle events	Unmapped confirmed events	First oracle proof	Tool calls	Negative-control hits
Opus-4.7	completed	8/12	18	11	3	7	420.2s	176	0
GLM-5.1	completed	7/12	21	12	5	9	447.5s	200	0
GPT-5.5 xhigh	completed	7/12	11	8	1	3	774.2s	299	0
Sonnet-4.6	completed	7/12	16	10	3	6	565.7s	178	0
DeepSeek-V4-Pro	completed	6/12	12	8	2	4	1605.6s	104	0
Kimi-K2.6	completed	2/12	5	2	0	3	1516.0s	111	0

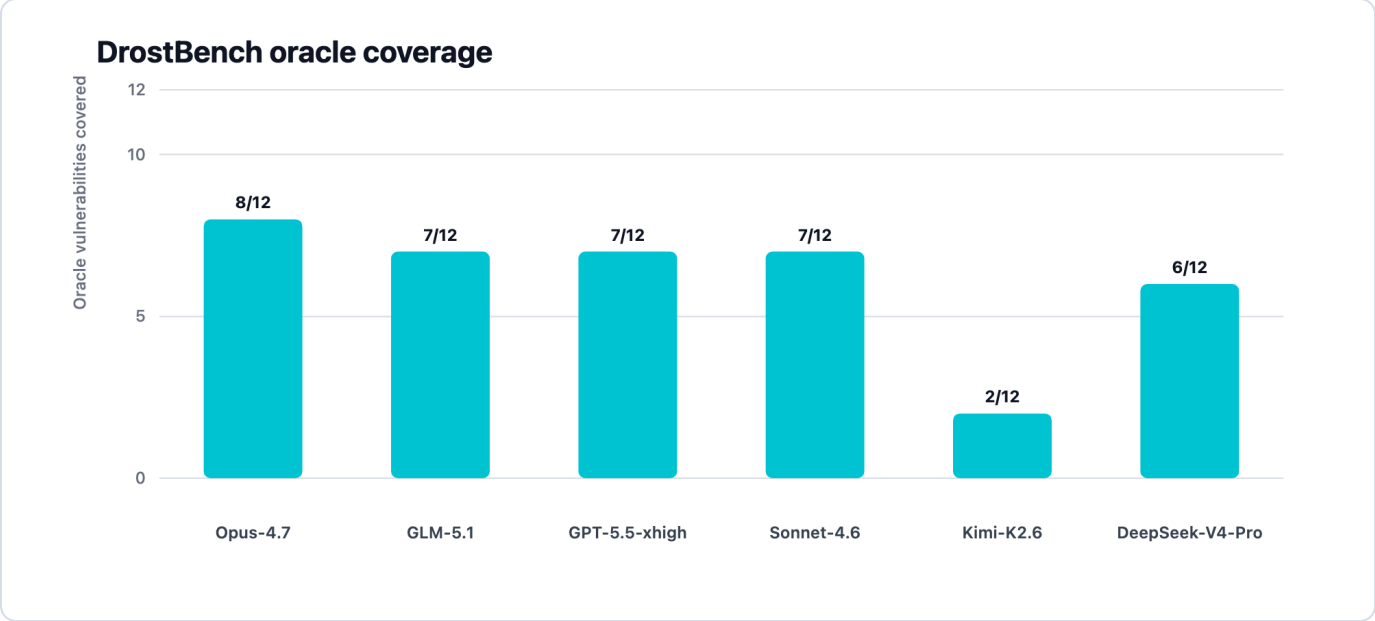

Figure 8. DrostBench oracle-normalized coverage. This de-duplicates repeated root-cause reports and credits only expected vulnerabilities in the harness-side oracle.

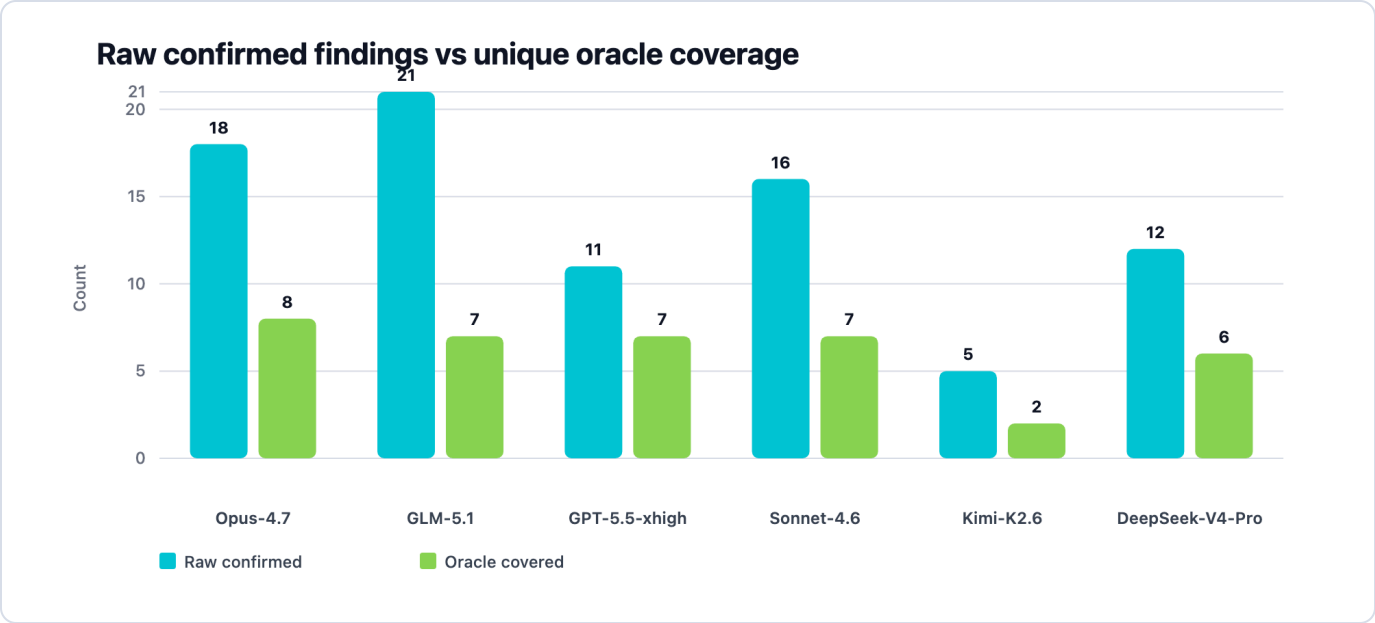

Figure 9. Raw confirmed findings versus matched oracle events. GLM-5.1 produced the highest raw count, while Opus-4.7 covered the most distinct oracle vulnerabilities.

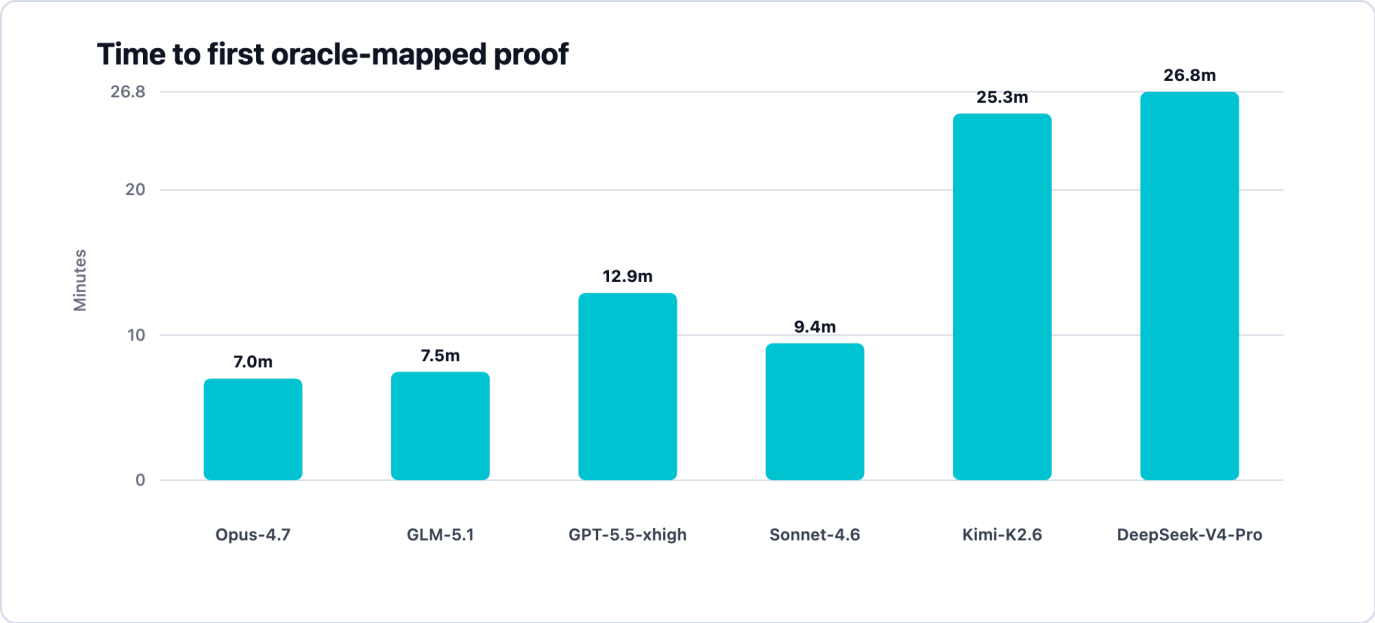

Figure 10. Time to first oracle-mapped proof. Opus-4.7 reached the first oracle-mapped proof fastest, with GLM-5.1 close behind.

DrostBench changes the interpretation of the same run data. By raw confirmed count, GLM-5.1 produced the highest output. By oracle-normalized coverage, Opus-4.7 covered the most distinct expected vulnerabilities. Sonnet-4.6 and GPT-5.5 xhigh tied GLM-5.1 on unique oracle coverage, but with different raw output, duplicate-event, and first-proof profiles.

This is why oracle-normalized coverage is a better private-benchmark metric than raw finding count. Raw count can reward duplicate reports, legitimate-but-out-of-oracle issues, or decomposition of one root cause into many findings. Oracle coverage asks the stricter question:

> Which expected vulnerabilities did the system actually prove?

No replay proofs were recorded on this DrostBench run. That means the result should be treated as private full-stack exploit-conversion evidence, not as additional replay-proof evidence. RolePlay remains the cleaner replay-proof target.

The DrostBench result is still preliminary because it is one private target, not a benchmark family. Its value is not final ranking. Its value is methodological: it shows why private oracle scoring is needed before Drost makes stronger public claims about model routing or cross-model comparison.

8.5 Cross-Target Readout

The four-target evidence supports a more useful claim than a model leaderboard.

Question	RolePlay signal	crAPI signal	Juice Shop signal	DrostBench signal
Which model produced the highest total confirmed yield?	GLM-5.1: 19 confirmed	GLM-5.1: 23 confirmed	Opus-4.7: 70 confirmed	Raw count: GLM-5.1, 21 confirmed; oracle coverage: Opus-4.7, 8/12
Which model reached confirmed proof fastest?	Opus-4.7: 66.2s	Opus-4.7: 225.4s	Opus-4.7: 127.0s	Opus-4.7: 420.2s to first oracle-mapped proof
Which target exercised replay proof?	Yes: 10 GLM replay proofs, 4 Opus, 2 Sonnet/GPT/Kimi/DeepSeek	No replay proofs recorded	No replay proofs recorded	No replay proofs recorded
Which target exercised API exploit conversion?	Some, but role/replay is dominant	Yes: all confirmed findings in scorecards were HTTP-behavior proof	Yes: broad web/API HTTP-behavior confirmation dominated	Yes: private full-stack app/API conversion with oracle scoring
What changed by target class?	replay conversion and multi-role auth context mattered most	authenticated API exploration, endpoint selection, and exploit commitment mattered most	breadth, known challenge surface, and web app traversal mattered most	private oracle scoring separated raw output from unique vulnerability coverage

Four patterns matter.

First, target class changes the conversion problem. RolePlay rewards models that can turn browser/auth/role evidence into replay proof. crAPI rewards models that commit to authenticated API exploit hypotheses and confirm them through HTTP behavior. Juice Shop rewards broad full-stack web exploration and rapid conversion across a larger known vulnerable surface.

Second, model rankings are not universal. GLM-5.1 produced the highest confirmed yield on RolePlay and crAPI. Opus-4.7 produced the fastest first confirmed result across RolePlay, crAPI, and Juice Shop, dominated Juice Shop on total yield, and led DrostBench on oracle-normalized coverage. Sonnet-4.6 was not the highest-yield lane, but it remained stable and tied GLM-5.1 and GPT-5.5 xhigh on DrostBench oracle coverage. GPT-5.5 xhigh produced fast early crAPI proof and strong DrostBench coverage, but under-converted relative to GLM and Opus on other targets.

Third, proof source matters as much as count. RolePlay's strongest evidence is replay proof. crAPI and Juice Shop were dominated by HTTP-behavior proof. Treating those as the same kind of "finding count" would blur the product signal.

Fourth, oracle normalization matters. DrostBench shows that raw confirmed-finding count can disagree with unique vulnerability coverage. That is a useful discipline for future public evaluations: count proof-backed findings, but also map them to a fixed oracle where the target permits it.

8.6 What These Evaluations Support

These evaluations support a bounded claim:

> Across four fixed-target Drost evaluations under the same 5M-token / 60-minute budget profile, models differed materially in how efficiently they converted evidence into confirmed outcomes. RolePlay measured replay-proof conversion. crAPI measured authenticated API exploit conversion. Juice Shop measured broad full-stack web conversion on a known vulnerable application. DrostBench measured private full-stack coverage with oracle normalization.

They do not support broader claims that:

- one model is globally better at cyber;
- any target represents all production applications;
- Drost has been independently benchmarked by a third party;
- vulnerable or synthetic targets fully predict customer environments;
- confirmed finding counts can be compared directly to external cyber benchmarks.

The results are still useful because they measure the product layer: not just whether a model can reason about cyber, but whether the full system converts that reasoning into scoped, proof-backed outcomes under different application-security shapes.

---

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 reveal practical differences between models that broad cyber capability scores do not capture. 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.** The representative evaluation needs a public-safe protocol before it can be treated as external evidence.
2. **Target representativeness.** Synthetic and intentionally vulnerable 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.
6. **Cost accounting.** Some provider cost data was unavailable in the compact scorecards, so cost-per-confirmed-finding is not yet publication-ready.
7. **Private benchmark breadth.** DrostBench currently represents one private full-stack target with six completed model lanes. It should become a family of public and private scenarios before being treated as a broad benchmark.

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 note 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. Are the four fixed-target evaluations framed with the right claim boundaries?
5. What additional 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. **Split DrostBench into a public/private benchmark project.** Publish a community benchmark with a public oracle while preserving private holdout scenarios for model routing and release validation.
5. **Add production-shaped applications.** Evaluate against realistic multi-role, API-heavy, workflow-heavy applications under authorization.
6. **Add cost-complete scorecards.** Provider cost data was unavailable in some compact scorecards; v1 evidence should include cost-per-confirmed-finding where terms permit disclosure.
7. **Improve customer-ready reporting.** Turn structured proof into concise, redacted, reviewable finding narratives.
8. **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

Representative Drost evaluations:

- Fixed-target RolePlay proof-efficiency evaluation, May 2026.
- Seven completed model lanes: GLM-5.1, Opus-4.7, GPT-5.5 xhigh, Sonnet-4.6, Kimi-K2.6, DeepSeek-V4-Pro, and Qwen-3.6-Plus.
- Fixed-target crAPI proof-efficiency evaluation, May 2026.
- Six model lanes attempted: Opus-4.7, GLM-5.1, GPT-5.5 xhigh, DeepSeek-V4-Pro, Kimi-K2.6, and Sonnet-4.6.
- crAPI produced five compact proof-efficiency scorecards. DeepSeek-V4-Pro hit the hard timeout before writing an engagement summary and is retained only as a failed wave artifact.
- Fixed-target OWASP Juice Shop proof-efficiency evaluation, May 2026.
- Six model lanes attempted: Opus-4.7, GLM-5.1, GPT-5.5 xhigh, Kimi-K2.6, DeepSeek-V4-Pro, and Sonnet-4.6.
- Juice Shop produced five compact proof-efficiency scorecards. Kimi-K2.6 hit the hard timeout before writing an engagement summary. DeepSeek-V4-Pro produced a scorecard before shutdown timeout and is retained with the warning.
- Fixed-target DrostBench private oracle-normalized evaluation, May 2026.
- Six model lanes completed: Opus-4.7, GLM-5.1, GPT-5.5 xhigh, Sonnet-4.6, Kimi-K2.6, and DeepSeek-V4-Pro. Qwen-3.6-Plus was started and stopped before completion and is excluded from the DrostBench aggregate.
- DrostBench used 12 harness-side oracle vulnerabilities and three negative controls. Compact scorecards were supplemented with oracle-normalized coverage to de-duplicate repeated root-cause reports and separate raw findings from unique expected-vulnerability coverage.
- Same Drost runtime, fixed target per lane, 5M-token cap, and 60-minute wall-clock profile.
- Compact scorecards were used for the public-safe summary. Raw engagement logs, credentials, and proof payloads are not included in this note.

Model IDs used in the representative evaluation:

Public label	Model ID
GLM-5.1	`accounts/fireworks/models/glm-5p1`
Opus-4.7	`claude-opus-4-7`
GPT-5.5 xhigh	`gpt-5.5`
Sonnet-4.6	`claude-sonnet-4-6`
Kimi-K2.6	`kimi-k2.6`
DeepSeek-V4-Pro	`deepseek-v4-pro`
Qwen-3.6-Plus	`accounts/fireworks/models/qwen3p6-plus`