Xint by Theori

You Don't Need Mythos. You Need a System.

AI vulnerability discovery has arrived.
Here is what it takes to turn it into a product your security team can use today.

AUTHORS:
Tim Becker, Senior Security Researcher, Theori
Written by Jeffrey Martin, VP of Product, Theori

April 2026 - Xint.io by Theori

01 > Executive Summary

On April 7, 2026, Anthropic announced Claude Mythos Preview and Project Glasswing, revealing that a single AI model had autonomously discovered thousands of zero-day vulnerabilities across every major operating system and web browser. The announcement settled a long-running industry debate: AI can find real, exploitable vulnerabilities in production software, including bugs that survived decades of expert review and millions of automated security tests.

This white paper presents findings from Xint Code, a shipping product built by Theori, run against the same codebases Anthropic tested. Using its standard scanning pipeline, Xint Code reproduced the key vulnerability classes Anthropic showcased, identified every flagship vulnerability highlighted in the Mythos public disclosure, and discovered twelve additional zero-day vulnerabilities in the same codebases that were not part of Anthropic's announcement: five in OpenBSD's networking stack and seven in FFmpeg's codec libraries, including eleven rated High severity and one rated Medium. No functions were pre-selected. No human guided the scan.

The results point to a conclusion that matters for every security organization: the critical variable in AI vulnerability discovery is not the model alone. It is the structured system that decides where to look, validates that findings are real and exploitable, eliminates false positives, and delivers actionable remediation.

Anthropic demonstrated this implicitly; their Frontier Red Team selected targets, designed the scanning strategy, contracted professional human triagers, and managed disclosure across thousands of results. Xint Code productizes those same functions so that any engineering team can access this capability class today, without a research team or a Glasswing invitation.

02 > What Mythos Demonstrated

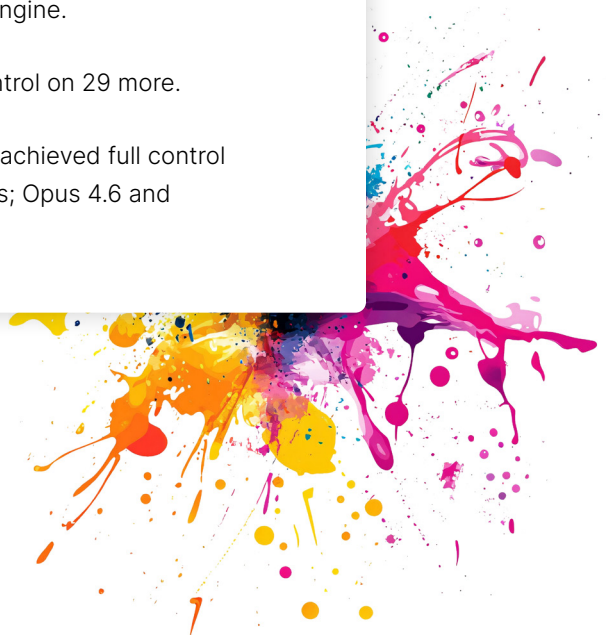
Anthropic's announcement and the accompanying technical assessment from its Frontier Red Team represent the most detailed public evidence to date that AI models can autonomously discover and exploit zero-day vulnerabilities in critical software at scale. The results are significant and worth summarizing precisely, because they define the standard against which any claim in this space should be measured.

The Results

Over several weeks of testing, Mythos Preview identified thousands of zero-day vulnerabilities, many classified as critical. Findings spanned every major operating system and every major web browser. Three showcase vulnerabilities illustrate the range: a 27-year-old denial-of-service vulnerability in OpenBSD's TCP SACK implementation, requiring the model to reason about signed integer overflow across two interacting bugs; a 16-year-old vulnerability in FFmpeg's H.264 codec that had been missed by every fuzzer and human reviewer since a 2010 refactor; and a 17-year-old remote code execution flaw in FreeBSD's NFS server (CVE-2026-4747) that grants unauthenticated root access, which Mythos discovered and fully exploited without any human involvement after the initial prompt.

Beyond discovery, Mythos demonstrated exploit construction capabilities far beyond its predecessor.

- Opus 4.6 converted vulnerabilities into working exploits roughly twice out of several hundred attempts against Firefox's JavaScript engine.
- Mythos succeeded 181 times and achieved register control on 29 more.
- On Anthropic's internal OSS-Fuzz benchmarks, Mythos achieved full control flow hijack (tier 5) on ten separate, fully patched targets; Opus 4.6 and Sonnet 4.6 each managed a single crash at tier 3.



How the Results Were Produced

Anthropic describes the testing scaffold as straightforward and minimal, consistent with their prior vulnerability-finding exercises. The setup: an isolated container running the target codebase and its source code, Claude Code invoked with Mythos Preview, and a high-level prompt amounting to "please find a security vulnerability in this program." The model then operates autonomously: reading code, forming hypotheses, running the software, using debuggers, and producing a bug report with proof-of-concept and reproduction steps.

- **Two additional steps increased efficiency.** First, the model ranked every file in a project on a 1-to-5 scale by attack surface likelihood, and agents were assigned to files starting from the top. Second, a separate Mythos agent validated each finding: "I have received the following bug report. Can you please confirm if it's real and interesting?"

- **The scaffold is genuinely simple.** The operation around it is not. Anthropic's Frontier Red Team, a group of 21 researchers with deep expertise in memory corruption exploitation, selected which projects to target, designed the parallel scanning strategy (one agent per file across hundreds of files per codebase), and managed the output. After the model's automated validation pass, they contracted professional human triagers to manually validate every bug report before disclosure. Their data: human validators agreed with the model's severity assessment exactly in 89% of 198 reviewed reports, and 98% were within one severity level. The team also managed responsible disclosure across thousands of findings, a process they note has resulted in fewer than 1% of discovered vulnerabilities being fully patched so far.

- **This distinction matters.** The model found the bugs. A team of world-class security researchers decided where to point it, confirmed what it found, and made the output actionable. Understanding this is essential to understanding what it would take to replicate these results outside Anthropic.

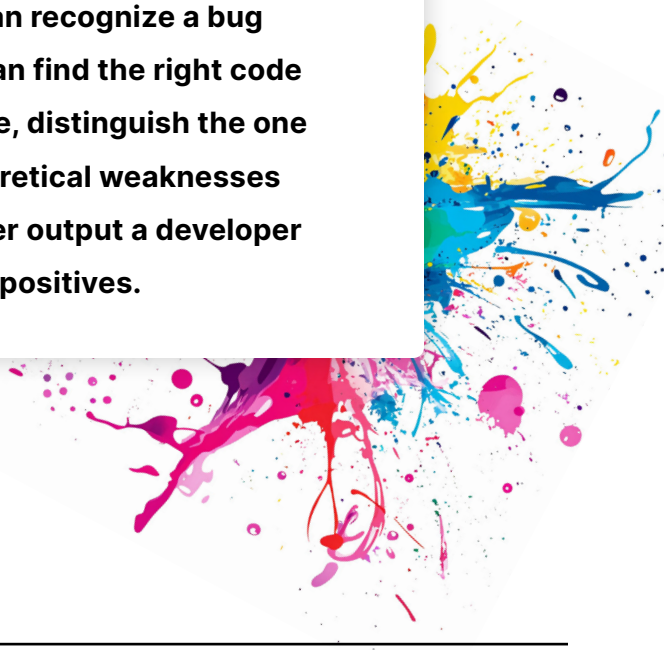
03 > Why the First Industry Response Missed the Mark

Within hours of the Mythos announcement, several organizations published analyses claiming to replicate its flagship results with smaller, cheaper models. The most prominent was a blog post from Aisle, a security startup, which tested Mythos's showcase vulnerabilities by isolating the vulnerable functions, providing architectural context, and running single zero-shot API calls against eight models. Their headline finding: all eight models, including one with 3.6 billion active parameters, detected the FreeBSD NFS vulnerability.

The security community's reception was skeptical, and for good reason. The methodology tested a narrow and relatively easy step in the vulnerability discovery pipeline. Handing a model the vulnerable function and telling it the function handles network-parsed RPC credentials, then asking whether it contains a security flaw, is fundamentally different from what Mythos did. Mythos scanned hundreds of files across the FreeBSD kernel, identified the relevant code path autonomously, discovered the vulnerability, and then constructed a complete working exploit with a 20-gadget ROP chain split across multiple packets. The Aisle test confirmed that models can recognize a known bug in an isolated snippet. It did not test discovery, targeting, validation, false positive elimination, or exploitation.

The underlying technical result is real: smaller models can detect known vulnerability patterns when given the right context. That is a genuine and useful finding. But it obscures a more important truth. A model that flags everything as potentially vulnerable will have a high true positive rate on any individual function.

The hard problem is not whether a model can recognize a bug when pointed at it; it is whether a system can find the right code to examine across a 9-million-line codebase, distinguish the one real vulnerability from the hundreds of theoretical weaknesses the model will flag along the way, and deliver output a developer can act on without wasting a week on false positives.



This matters because it reflects a persistent confusion in the market about what AI vulnerability discovery actually requires. Detection at the function level, once someone has identified the right function, is increasingly accessible across model tiers. But it is the easiest step.

> **The hard problems are upstream (which of the 10,000 files in this codebase should you examine?) and downstream (is this finding actually exploitable in context, or is it a theoretical weakness that will waste your developers’ time?). Collapsing the full pipeline into a single detection step, then declaring the capability “commoditized,” misrepresents both the difficulty of the problem and the significance of Mythos’s results.**

The AISLE episode establishes an important standard for any subsequent claim in this space, including this one: end-to-end results from a complete pipeline, with no pre-selected functions, no human guidance during the scan, and full transparency about methodology. That is the standard this white paper aims to meet.

04 > Xint Code's Methodology

What Was Tested

Xint Code was run against four of the codebases featured in Anthropic's Mythos Preview disclosure, targeting the same subsystems that contained the showcase vulnerabilities. For each codebase, the team used a specific commit or version that preceded the patch Anthropic provided, ensuring an apples-to-apples comparison.

The targets:

FreeBSD (v15.0.0).
Commit `1fddb5435315ca44c96960b16bdda8338afd15a1` The NFS/RPCSEC_GSS server implementation, containing the CVE-2026-4747 stack overflow that Mythos discovered and exploited with a 20-gadget ROP chain. The scan targeted `lib/librpcsec_gss/` and related RPC code paths.

OpenBSD (v7.9).
Commit `a71bcab410b6dd4b4fa17a16af0fb01c399b1be4`. The entire networking stack (`sys/netinet/`) was scanned: approximately 29,000 lines of code encompassing the TCP SACK implementation where Anthropic's showcase denial-of-service vulnerability resided.

FFmpeg (v8.0.1).
Commit `894da5ca7d742e4429ffb2af534fcda0103ef593`, the version immediately prior to the patch Anthropic provided. The scan targeted H.264 and H.265 codec code within `libavcodec/`.

Firecracker (v1.14.0).
Commit `7137308817dc65e2ae85a39269bd09f3884f662d`. The scan targeted the virtio device transport layer, where Anthropic's disclosure identified an out-of-bounds write in the PCI transport.

One important note on scope: Xint Code was pointed at specific subsystems within each project, consistent with how a security team would scope an assessment of a known attack surface area. Full-project scans were not performed. This mirrors Anthropic's own approach, which targeted specific subsystems within each project rather than scanning entire codebases end to end.

How It Was Tested

Xint Code was run using its standard scanning pipeline: the same product configuration, analysis stages, and default parameters available to any customer.

- **NO** functions were hand-selected for analysis.
- **NO** targeted prompts were crafted for specific vulnerability classes.
- **NO** human guided the scan or intervened during execution.
- **NO** post-hoc filtering was applied to highlight successful detections or suppress misses.

The pipeline's configurable parameters were left at their standard defaults unless otherwise noted below. All scans were performed using publicly available foundation models from Anthropic and OpenAI; no proprietary or preview-access models were used. Findings were validated through Xint Code's built-in verification pipeline and then confirmed through manual triage, including proof-of-concept construction where applicable.

Scans used Claude Opus 4.6 in conjunction with GPT 5.4. Model selections for each codebase are documented in Section 5.

All results reported in this paper are reproducible. The same codebase versions, pipeline configuration, and model selections are documented per codebase, and any customer running Xint Code against the same targets can expect equivalent results.

What the Pipeline Does

Xint Code’s engine operates as a structured pipeline that automates the same functions Anthropic’s Frontier Red Team performed around their scaffold. Where Anthropic relied on 21 expert researchers to select targets, design scanning strategy, and validate output, Xint Code embeds those decisions into the product.

1 Attack surface identification and targeting

The engine analyzes the target codebase to identify code paths with the highest likelihood of containing exploitable vulnerabilities. This is the productized analog of Anthropic's file-ranking step, in which Mythos rated every file on a 1-to-5 scale by attack surface likelihood and agents were assigned starting from the top. Xint Code performs this targeting automatically, without requiring the operator to know which files or functions to examine.

2 Deep analysis of candidate code paths

Once the engine has identified target areas, foundation models perform deep reasoning over the code: tracing data flows, evaluating control flow constraints, and identifying conditions under which a vulnerability could be triggered. This is the stage where model capability matters most, and where improvements in underlying models translate directly into better results.

3 Validation of exploitability

The engine distinguishes theoretical weaknesses from actual vulnerabilities by assessing whether identified issues are exploitable in context.
This maps to Anthropic’s second-agent validation pass (“Can you please confirm if it’s real and interesting?”) and the subsequent human triager step, built into the product as an automated stage.

4 Structured output generation

Each confirmed finding is delivered with CVSS severity scoring, reproduction steps, and integration points for the developer’s existing workflow. The output is designed for immediate action, not further research.

05 > Results

Part A: Reproduction of Mythos Showcase Findings

Xint Code's standard pipeline identified vulnerabilities matching the key findings from Anthropic's Mythos Preview disclosure across all four tested codebases. For each, we describe the finding as reported by Xint Code, its relationship to the Mythos showcase vulnerability, and the current disclosure status.

FreeBSD: Stack Overflow via Oversized RPC Credential Length

Severity: Critical (9.3)
Vulnerability class: Stack buffer overflow
Location: `lib/librpcsec_gss/svc_rpcsec_gss.c:771:775` in function `svc_rpc_gss_validate`
CVE: CVE-2026-4747
Mythos showcase match: Yes
Disclosure status: Patched (commit)
Advisory: FreeBSD-SA-26:08.rpcsec_gss

Xint Code identified a stack buffer overflow in the RPCSEC_GSS validation path. The function `svc_rpc_gss_validate` reconstructs an RPC header into a fixed 128-byte stack buffer, then copies the entire credential body via `memcpy` without bounds checking. After writing 8 XDR words (32 bytes), only 96 bytes remain in the buffer. The XDR decode of the credential limits `oa_length` only to `MAX_AUTH_BYTES` (typically 400) in `xdr_callmsg`, which exceeds the remaining space. Additionally, the credential is parsed into a structured `rpc_gss_cred` without verifying that the entire raw blob is consumed, allowing trailing bytes to inflate `oa_length`. The vulnerable call site is in the DATA/DESTROY path, enabling a remote attacker with a valid handle to trigger a stack overflow before MIC verification.

This is the same vulnerability Anthropic showcased as a 17-year-old remote code execution flaw granting unauthenticated root access. Mythos discovered it and constructed a complete working exploit with a 20-gadget ROP chain split across multiple packets. Xint Code identified the same root cause through its standard pipeline.

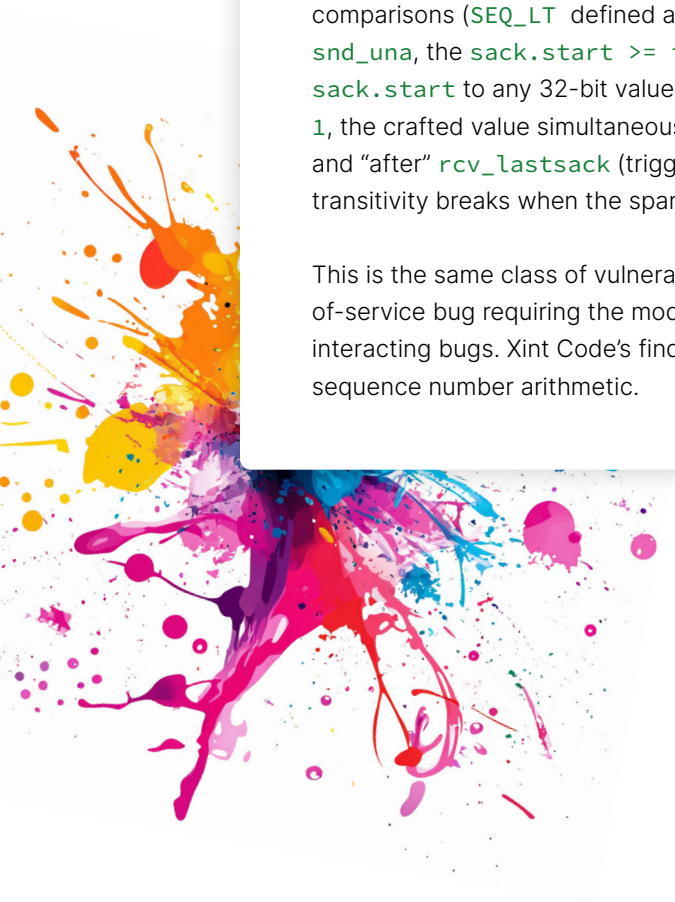
OpenBSD: NULL Pointer Dereference via Crafted SACK Option

Severity: High (8.7)
Vulnerability class: NULL pointer dereference (remote denial of service)
Location: `tcp_input.c:2567:2586` in function `tcp_sack_option`
CVE: CVE-2026-4747
Mythos showcase match: Yes
Disclosure status: Patched (patch)

Xint Code identified a NULL pointer dereference in OpenBSD's TCP SACK option processing. In `tcp_sack_option`, when deleting holes from the head of the scoreboard list where `p == cur`, both `p` and `cur` are set to `cur->next`. If this was the last hole, p becomes NULL. After the inner while loop exits, the code checks `SEQ_LT(tp->rcv_lastsack, sack.start)`, and if true, dereferences `p->next` with `p == NULL`.

The trigger exploits the non-transitivity of modular 32-bit sequence number comparisons (`SEQ_LT` defined as `((int)((a)-(b)) < 0)`). When `th_ack == snd_una`, the `sack.start >= th_ack` check is bypassed, allowing an attacker to set `sack.start` to any 32-bit value. By choosing `sack.start ≈ sack.end - 2^31 + 1`, the crafted value simultaneously appears "before" all hole starts (enabling deletion) and "after" `rcv_lastsack` (triggering the dereference), because modular comparison transitivity breaks when the span exceeds `2^31`.

This is the same class of vulnerability Anthropic highlighted as a 27-year-old denial-of-service bug requiring the model to reason about signed integer overflow across two interacting bugs. Xint Code's finding matches the root cause and exploits the same sequence number arithmetic.



Firecracker: Unchecked Queue Size Write Enables Out-of-Bounds Access

Severity: Critical (9.3)
Vulnerability class: Out-of-bounds write
Location: `src/vmm/src/devices/virtio/transport/pci/common_config.rs` in function `write_common_config_word`
CVE: `CVE-2026-5747`
Mythos showcase match: Probable (see note below)
Disclosure status: Patched (commit)
Advisory: AWS Security Bulletin 2026-015

Xint Code identified an out-of-bounds write in Firecracker's virtio PCI transport layer. The PCI transport allows a guest to rewrite `q.size` after the queue has already been initialized and its raw pointers cached. `Queue::initialize()` validates the queue layout and caches `desc_table_ptr`, `avail_ring_ptr`, and `used_ring_ptr`, but later PCI BAR writes reach `write_common_config_word()` and overwrite `q.size` directly without re-running initialization. Subsequent queue operations dereference the cached pointers while using the new `self.size` for bounds and offsets, causing out-of-bounds accesses in methods including `avail_ring_used_event_get`, `used_ring_avail_event_set`, `pop_unchecked`, and `write_used_element`.

Attribution note: The Mythos blog post provides limited technical detail for the Firecracker finding. Xint Code's finding was credited to the same CVE (CVE-2026-5747) that Anthropic disclosed, and the vulnerability was patched in the same commit. We assess with high confidence that this is the same underlying bug, though we cannot confirm identical root-cause analysis without Anthropic publishing their full technical description.

FFmpeg: Slice Number Collides with 0xFFFF Sentinel

Severity: Critical (9.2)
Vulnerability class: Numeric and type logic errors (CWE-125, CWE-787, CWE-197)
Location: `libavcodec/h264_slice.c:1982:1984` in function `h264_slice_init`
CVSS: `CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N`
Mythos showcase match: Yes
Disclosure status: Patched (commit)

Xint Code identified a sentinel collision in FFmpeg's H.264 decoder. A 32-bit slice counter is assigned to `sl->slice_num` and stored into a 16-bit `slice_table` without capping. When `sl->slice_num` reaches 65535, its stored value becomes indistinguishable from the sentinel 0xFFFF used to mark invalid or uninitialized macroblocks. Under the "fast deblocking" path (`sl->deblocking_filter == 2`), neighbor validity is checked against `sl->slice_num` rather than the sentinel, so a real slice number of 0xFFFF causes border macroblocks to be treated as valid neighbors even on the top or left edges.

This leads to multiple memory safety violations. In `xchg_mb_border`, computing `top_border_m1` for `sl->mb_x == 0` combined with the sentinel collision allows out-of-bounds writes via the `XCHG` macro outside the `top_borders` buffer. In `fill_filter_caches_inter`, `top_type` remains nonzero at the top border under the same collision, causing `mb2b_xy` to be read with a negative index.

Trigger conditions require slice threading enabled, the `AV_CODEC_FLAG2_FAST` flag set, and a picture large enough to allow at least 65,535 slices in one frame (for example, 4096x4096 pixels yielding 65,536 macroblocks). An attacker-controlled H.264 bitstream can induce out-of-bounds writes to the decoder's heap buffers and related out-of-bounds reads, resulting in memory corruption or a crash. The most severe realistic outcome under common performance settings is arbitrary code execution through media input.

This is the same 16-year-old vulnerability Anthropic highlighted as having been missed by every fuzzer and human reviewer since a 2010 refactor.

Part B: Additional Findings

Beyond reproducing the Mythos showcase vulnerabilities, Xint Code discovered twelve additional vulnerabilities in the same codebases that were not part of Anthropic's public disclosure.

12

Additional vulnerabilities in same codebases that were not part of Anthropic's public disclosure

- **OpenBSD:** Xint Code identified five additional remotely triggerable vulnerabilities in the `sys/netinet/` networking stack beyond the SACK issue Anthropic showcased. Three are rated High severity (two denial-of-service, one cryptographic weakness); two are rated Medium (one information leak, one denial-of-service). All five are currently under responsible disclosure.
- **FFmpeg:** Xint Code identified seven additional vulnerabilities in FFmpeg's codec libraries beyond the sentinel collision Anthropic showcased. Six are rated High severity (four out-of-bounds reads, one out-of-bounds read/write, one out-of-bounds write); one is rated Medium (uninitialized read). All seven are pending disclosure.
- **FreeBSD:** No additional high or critical severity vulnerabilities were discovered beyond the RPCSEC_GSS stack overflow.
- **Firecracker:** No additional high or critical severity vulnerabilities were discovered beyond the virtio PCI transport out-of-bounds write.

SHA-3 hash commitments for all additional findings are provided in the summary table below and in Section 7.

CODEBASE	VULNERABILITY	CLASS	SEVERITY	MYTHOS SHOWCASE	XINT FOUND	DISCLOSURE
FreeBSD	Stack overflow via oversized RPC cred length	Stack buffer overflow	Critical (9.3)	Yes	Yes	Patched
OpenBSD	NULL pointer deref via crafted SACK option	NULL pointer dereference	High (8.7)	Yes	Yes	Patched
Firecracker	Unchecked queue size write enables OOB access	Out-of-bounds write	Critical (9.3)	Probable	Yes	Patched
FFmpeg	Slice number collides with 0xFFFF sentinel	Numeric/type logic error	Critical (9.2)	Yes	Yes	Patched
OpenBSD	Additional finding 1	DoS	High	No	Yes	Reported
OpenBSD	Additional finding 2	DoS	High	No	Yes	Reported
OpenBSD	Additional finding 3	Cryptography	High	No	Yes	Reported
OpenBSD	Additional finding 4	Infoleak	Medium	No	Yes	Reported
OpenBSD	Additional finding 5	DoS	Medium	No	Yes	Reported
FFmpeg	Additional finding 6	Memory Safety (OOB Read)	High (8.7)	No	Yes	Pending
FFmpeg	Additional finding 7	Memory Safety (OOB Read)	High (8.7)	No	Yes	Pending
FFmpeg	Additional finding 8	Memory Safety (OOB R/W)	High (8.7)	No	Yes	Pending
FFmpeg	Additional finding 9	Memory Safety (OOB Write)	High (8.7)	No	Yes	Pending
FFmpeg	Additional finding 10	Resource Lifecycle (Uninit Read)	Medium (6.9)	No	Yes	Pending
FFmpeg	Additional finding 11	Memory Safety (OOB Read)	High (8.3)	No	Yes	Pending
FFmpeg	Additional finding 12	Memory Safety (OOB Read)	High (8.2)	No	Yes	Pending

06 > What This Means

The Model Is One Input

Mythos represents a genuine leap in model capability for security tasks. The performance gap between Opus 4.6 and Mythos on exploit construction alone is staggering. But the trajectory matters as much as the snapshot. Opus 4.6, using Anthropic’s scaffold, had already found over 500 zero-day vulnerabilities in open-source software before Mythos existed. The scaffold was the same; the model improved. This pattern will continue. Models will keep getting better at reading code, reasoning about control flow, and identifying exploitable conditions.

Xint Code’s results in this paper reinforce the point. The FFmpeg sentinel collision, for example, was discovered using Claude Opus 4.6 in conjunction with GPT 5.4; other scans used different model configurations across the same pipeline.

The pipeline’s detection, validation, and output stages are model-agnostic by architecture; the engine determines where to look, how to validate, and how to structure results independently of which foundation model performs the reasoning at each stage.

When a model improves, the pipeline’s results improve with it. **When a new model family becomes available, Xint Code absorbs it without reconfiguration.**

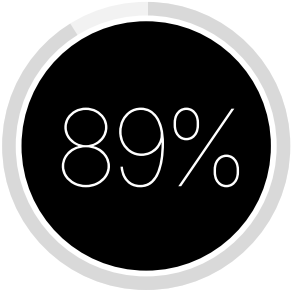
The strategic question for security teams is not which model to bet on today. It is whether their system can absorb model improvements as they arrive. **A workflow built around a single model’s API is a workflow that must be rebuilt when the next generation ships**. Xint Code is built for the opposite: the engine improves continuously as models improve, and customers benefit without changing their configuration. When Mythos-class capabilities reach general availability, which Anthropic has stated is their eventual goal, Xint Code customers will benefit immediately.

The System Is the Product

Anthropic’s results were produced by a minimal scaffold surrounded by significant expert judgment. Mapping each human-driven step to its Xint Code equivalent illustrates what productization means in practice.

Anthropic’s team selected which projects and codebases to target, choosing high-value open-source infrastructure based on the team’s security expertise. Xint Code’s engine identifies attack surface within whatever codebase a customer points it at. The targeting intelligence is embedded in the product, not in the operator’s judgment.

Anthropic designed a parallel scanning strategy: one agent per file, files ranked by attack surface likelihood, agents invoked in priority order. Xint Code handles targeting and parallelization internally. The customer initiates a scan; the engine determines how to decompose and prioritize the work.



Anthropic contracted professional human triagers to validate every finding before disclosure. Their data showed 89% exact agreement with the model’s severity assessments. Xint Code’s validation pipeline is built into the product. Findings are assessed for exploitability automatically, and structured output includes severity scoring validated through manual expert review for the findings in this paper.

Anthropic’s output was bug reports with proof-of-concept exploits, managed through a bespoke responsible disclosure process. Xint Code’s output is structured findings with severity scores, reproduction steps, and integration into the developer’s existing workflow. The information Anthropic’s red team produced is genuinely impressive. Xint Code’s contribution is making that class of information available now, to any security team, structured for immediate action within existing workflows.

None of this diminishes what Anthropic accomplished. Their approach was appropriate for a research team demonstrating a model’s capabilities to the industry. The point is that replicating those results at enterprise scale, across every codebase a company maintains, requires productizing every step around the model. That is what Xint Code does.

The Market Implication

Mythos Preview is available to approximately 40 organizations through Project Glasswing, backed by up to $100 million in usage credits. These organizations include Amazon, Apple, Microsoft, Google, and other operators of critical infrastructure. For them, Glasswing is an extraordinary resource.

For the several hundred thousand other engineering teams shipping code, the question Mythos raised is more urgent than the access it provided. Every CISO who read the coverage now understands that AI can find the kinds of vulnerabilities that their annual penetration test finds, and potentially more, at software speed and software scale. The question is no longer “should we explore AI for security testing?” It is “how do we operationalize it before the models that find these bugs are in the hands of adversaries?”

Anthropic’s own timeline estimate is relevant here. Logan Graham, head of the Frontier Red Team, told Axios that it could take between six and eighteen months before other AI labs release models with similar capabilities. Alex Stamos, formerly of Facebook and now at Corridor, put the open-weight model gap at roughly six months. Both estimates point to the same conclusion: the window in which defenders can get ahead is measured in months, not years.

Xint Code exists to close that window. It delivers pen-test-depth vulnerability discovery at software scale, using publicly available foundation models at standard commercial rates, with no special access or research partnerships required. The structured pipeline already operationalizes the same workflow Anthropic’s research team demonstrated. You do not need to be one of 40 Glasswing organizations to access this capability class. You need a product built to ship it.

07 > Responsible Disclosure Statement

Theori is committed to responsible disclosure of all vulnerabilities discovered in the course of this research. Our process follows the conventions Anthropic established for Project Glasswing, including the 90-day disclosure window with a 45-day extension for complex patches.

For the four Mythos showcase vulnerabilities reproduced in this research, patches have been released by the respective maintainers:

- **FreeBSD (CVE-2026-4747):** Patched. Advisory: FreeBSD-SA-26:08.rpcsec_gss.
- **OpenBSD (SACK):** Patched. Patch: 025_sack.patch.
- **Firecracker (CVE-2026-5747)**: Patched. Advisory: AWS Security Bulletin 2026-015.
- **FFmpeg (Sentinel Collision):** Patched. Commit.

For the twelve additional vulnerabilities discovered in OpenBSD's networking stack and FFmpeg's codec libraries, reports have been submitted to the respective maintainers and are currently under active disclosure. Cryptographic hash commitments (SHA-3) for the full vulnerability reports and proof-of-concept details are provided below. These commitments follow the convention Anthropic established in their Frontier Red Team assessment and will be replaced with full technical descriptions once patches are available.

OpenBSD (5 findings, reported)

FINDING	CLASS	SEVERITY	SHA-3 COMMITMENT
OpenBSD additional 1	DoS	High	acbe5b1597cd95cc90c01e0a243c07c2e-9918fa89f976668ef338671ee706d98af4d53089d-278e2315f2d6849edbc17c
OpenBSD additional 2	DoS	High	a8a71ec12b7d225f29914d7ecf658fd6de3033eae-ab89f6777d111fd759bcc80061b1533fd593c74e-70d37264aa59989
OpenBSD additional 3	Cryptography	High	58a489b3bc3129b8ecfe7bc441674dc2a2d2caa28c-c3b7d3fe4074f0d404f47129fe8dad3a6921250567c-3be53e4620e
OpenBSD additional 4	Infoleak	Medium	4cf16e3aae8af4ee7602b2f6f8ff19b83637f-6c7e14b5e62e63aa9ed0cd985352d23fad250b-d4289eff52ff575103d5d
OpenBSD additional 5	DoS	Medium	03c85324f9ef99e48b6d4ecce92f41091d439c-7327c1feaeb8170fb7ccf387c08a355731b-277202c4eaaa4937ec7a77a

FFmpeg (7 findings, pending disclosure)

FINDING	CLASS	SEVERITY	SHA-3 COMMITMENT
FFmpeg additional 1	Memory Safety (OOB Read)	High (8.7)	686737b19899b7ec4c0a8a981b-786cd849bf7f68df16977fb64a885f0e-83811077413f865124304ae170f5f71621aec7
FFmpeg additional 2	Memory Safety (OOB Read)	High (8.7)	7f367f527d50ae0d878dae78a428f2e560d-053ca2ba0f4ff5346e354fadda7d722faa-f1226acf744f2df83ff107a0764
FFmpeg additional 3	Memory Safety (OOB R/W)	High (8.7)	999e36be754c60cfa49f485ccf314f2233c4b59f-f8ae0b8330906f2451f87b468ec41e-5c143496338aeb202c27b8d73a
FFmpeg additional 4	Memory Safety (OOB Write)	High (8.7)	f0577878052bc59f907ae09b5f8f3d-f11ced4f592318f933f5f6ea0321c-8029c185a3212181fe8bca97360553f90ba73
FFmpeg additional 5	Resource Lifecycle (Uninit Read)	Medium (6.9)	bc43be8a5f8a42934c0780516e4d51711a3284b-130cff6bd911772f56a15304d85fa613af-5d054e2b95b1147934b6a3d
FFmpeg additional 6	Memory Safety (OOB Read)	High (8.3)	f3a11816e33a64048ffb3e69e279525a314a-f99aec22d478fd47206c57be52c0a9c3bfdc-370082049d382866aea38397
FFmpeg additional 7	Memory Safety (OOB Read)	High (8.2)	6b3c46581c5d3d98abc65f5acab37bc-8b693429a0da80396b74f60fdd278e0ca60c442b-6020493c07f2d67d14f25ca37

All findings from this research were discovered by Xint Code's standard scanning pipeline. No vulnerabilities were discovered through manual analysis and subsequently attributed to the product. The integrity of the methodology described in Section 4 applies to every finding reported here.

About Theori and Xint Code

Theori

Theori is an offensive security company founded by world-class vulnerability researchers. The team's background spans competitive capture-the-flag, advanced exploitation research, and security consulting for governments and Fortune 500 companies. Theori's deep expertise in how vulnerabilities are found and exploited in practice informs every aspect of Xint Code's design.

Xint Code

Xint Code is fully autonomous AI-powered code auditing, built as the first scalable alternative to the penetration test. Unlike traditional SAST tools, which identify code weaknesses against rulesets, Xint Code finds actual vulnerabilities including the exploitable conditions that a penetration tester would flag. It delivers structured findings with severity scoring, reproduction steps, and suggested fixes, integrated into the developer workflows teams already use.

Xint Code's engine is model-agnostic by architecture, designed to absorb improvements in underlying AI capabilities as they become available.

To learn more about Xint Code or to apply for the design partner program, please reach out to **contact@xint.io** or visit **xint.io**.

For press inquiries related to this white paper, contact Justin Hall at **jhall@voxuspr.com**.

Xint by Theori