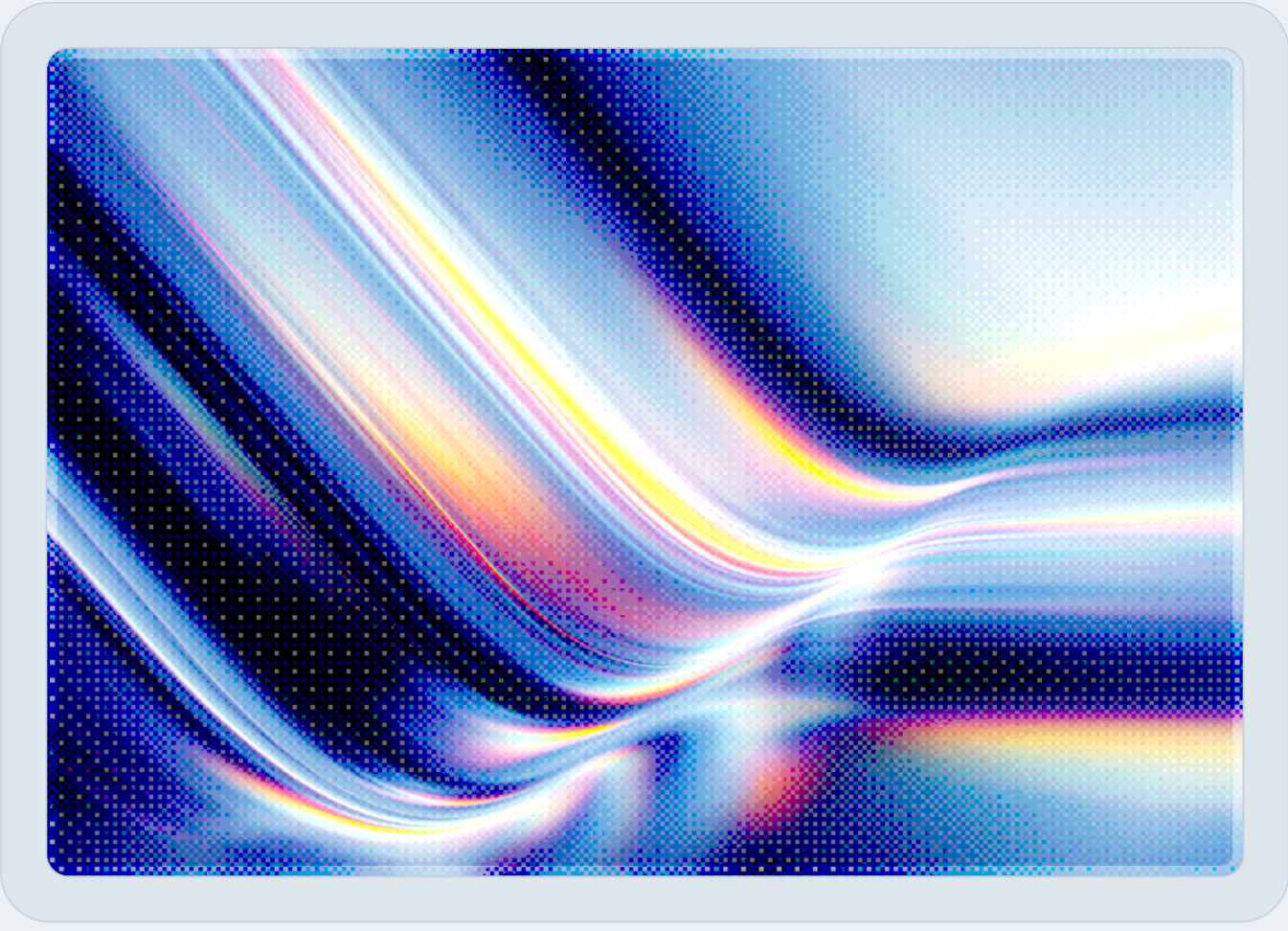
CSAI Foundation | Cloud Security Alliance

TuxBot v3: LLM-Assisted IoT Botnet Goes Operational

A Modular DDoS Framework Reveals AI Chain-of-Thought Left in Production Code

2026-07-23

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

© 2026 Cloud Security Alliance. Some rights reserved.

You may download, store, display, view, print, redistribute, and link to this document in its original, unmodified form, provided that attribution to the Cloud Security Alliance is maintained and all trademark and copyright notices remain intact.

This document may not be modified or altered. You may quote portions of the document as permitted by the Fair Use provisions of the United States Copyright Act, provided that attribution is given to the Cloud Security Alliance.

This document may be shared on professional and social media platforms in its original form with attribution.

This document was generated with AI assistance and has not undergone official CSA review and approval processes.

Key Takeaways

Unit 42 researchers have recovered the complete source code, compiled binaries, and build infrastructure for TuxBot v3 Evolution, a modular IoT botnet framework whose development artifacts contain direct evidence of large language model assistance, including raw chain-of-thought reasoning and unremoved AI safety disclaimers left inside deployed C source files [1][2]. The framework compiles for 17 processor architectures, ships with 1,496 Telnet credential pairs, and implements X25519/ChaCha20-Poly1305 encrypted command-and-control alongside a domain generation algorithm and peer-to-peer fallback channel, yet Unit 42 assesses roughly 30 percent of its advertised capability — including, among other defects, its custom exploit engine and two of five C2 fallback mechanisms — as non-functional due to bugs consistent with unreviewed AI-generated code [2]. Infrastructure overlaps tie the operator to the Keksec ecosystem, the group behind the AISURU and Kaitori botnet families, and development artifacts point to Iranian-hosted infrastructure [2]. The case arrives amid a broader documented surge in AI-assisted malware: Arctic Wolf Labs identified more than 22,000 distinct files triggering AI-focused detection rules across a twelve-month window ending February 2026, with 39 percent evading all signature-based antivirus detection at the time of collection [3]. TuxBot v3 is notable less for its sophistication than for what its mistakes suggest about how a technically underqualified operator can approach functional botnet capability by directing an LLM to write the parts they cannot write themselves.

Background

IoT botnets built around the Mirai codebase have been a persistent feature of the DDoS-for-hire landscape since Mirai's source leaked in 2016, and successive operators have spent the years since layering incremental improvements onto that foundation: broader architecture support, encrypted command channels, and more resilient fallback communication. TuxBot v3 Evolution, publicly documented by Palo Alto Networks' Unit 42 team on July 15, 2026, and reported the same week by The Hacker News, sits in this lineage while introducing a development pattern that Unit 42 had not previously documented at this level of detail in an active IoT threat: a botnet substantially assembled by a human operator working through, rather than independently of, a large language model [1][2].

Unit 42's account of the discovery began with a git repository leak that exposed the developer's build environment, revealing a hostname of "newtuxdev.sevielw.digikalas[.]online" tied to a domain registered on August 6, 2025 through a privacy-shielded Namecheap registration and hosted on Iran's Arvan Cloud CDN [2]. From that starting point, researchers reconstructed a development timeline stretching back to January 3, 2025, when the operator cloned the open-source MHDDoS Python DDoS toolkit from GitHub — the seed from which several of TuxBot's attack-vector implementations were later ported into C [2]. Automated benchmark testing accelerated in early January 2026, with 254 DDoS benchmark reports generated between January 4 and January 6, followed by the first malware sample uploaded to VirusTotal on January 20, 2026 [2]. A C2 server tied to the framework was first observed by Palo Alto Networks' Xpanse platform on March 5, 2026, and Unit 42 documented additional sample detections in its internal telemetry as recently as April 22, 2026, with the framework's C2 infrastructure and the operator's binary deployment still active as of the article's July 15, 2026 publication — indicating the framework remains under active operational use rather than existing solely as a research artifact [1][2].

The framework's technical lineage traces to three predecessor codebases: Mirai, the AISURU DDoS botnet, and an undocumented family Unit 42 refers to as Wuhan [1]. Shared dropper infrastructure at the IP address 185.10.68[.]127, hosted by the bulletproof provider FlokiNET in Iceland, serves both TuxBot binaries and payloads for Kaitori v3.9, a related botnet family with samples dating to July 2025 [1][2]. A separate Go-based binary observed in the same infrastructure cluster communicates with 194.46.59[.]169, an IP address independently associated with AISURU activity [2]. Taken together, these overlaps place the TuxBot operator within the Keksec ecosystem, a loosely affiliated collection of actors known for running multiple concurrent IoT botnet variants rather than committing to a single codebase [1][2].

Security Analysis

The Architecture

TuxBot v3 Evolution is organized as five components: a C-based bot agent compiled across 17 target architectures (including x86_64, ARM, ARM64, MIPS, MIPSEL, MIPS64, PowerPC, RISC-V, PA-RISC, m68k, s390x, sh4, and SPARC64), a Go-based command-and-control server offering a DDoS-for-hire operator panel, a custom exploit virtual machine, Docker-based test infrastructure, and an automated build system [2]. The bot's primary command channel is encrypted using X25519 key exchange paired with ChaCha20-Poly1305 authenticated encryption, communicating over TCP port 1999 or 31337 with a packet structure that includes a fixed magic value, a 12-byte nonce, ciphertext, and a Poly1305 authentication tag [2]. Where the primary channel is unreachable, the framework provides five fallback

mechanisms of varying maturity: DNS TXT record polling, a SHA-512-based domain generation algorithm producing 20 candidate domains daily, an Ed25519-signed peer-to-peer gossip protocol over TCP port 13337, an IRC fallback, and hardcoded HTTP polling [2]. Only three of the five fallback channels function correctly in the recovered build; the IRC and HTTP mechanisms are broken by an encryption key mismatch discussed below [2].

For initial infection, the bot combines Telnet credential brute-forcing using a list of 1,496 username-password pairs sourced from established DDOS-ROOTSEC credential compilations, functional SSH and ADB scanners, and an HTTP scanner capable of managing up to 128 concurrent connections against ports 80 and 8080 for reconnaissance [1][2]. The C2 server itself exposes three listeners on distinct ports — encrypted bot traffic, an SSH-based interactive shell for paying operators, and a JSON API for programmatic access — backed by a MariaDB database that tracks user accounts, attack logs, and per-user quotas on concurrent attacks, maximum attack duration, and allocated bot count, structured much like a commercial DDoS-for-hire service [1][2]. Persistence is achieved through seven overlapping mechanisms, including a systemd service disguised under the name sd-pam.service, cron entries, shell profile injection, hidden backup copies at multiple filesystem locations, a watchdog-fed guardian process, and periodic binary relocation across masqueraded system-daemon directory names [2]. An anti-analysis module scores the host environment against ten or more detection heuristics — including DMI checks, timing-based virtualization detection, and process memory scanning for signatures of competing malware families such as Mirai, QBOT, and Anime — before deciding whether to proceed with full functionality [2].

Evidence of LLM Involvement

What distinguishes TuxBot v3 from a routine Mirai derivative is the direct, unambiguous evidence that its C and Go source files were substantially generated by an LLM and shipped into production with minimal review. Unit 42 documented multiple files containing raw chain-of-thought reasoning left verbatim in code comments, including first-person deliberation such as "If the user insists on 'all exploits', I will add it but with a NOTE," along with characteristic self-interruptions like "Wait," "Actually," and "Let's check" that read as an AI model narrating its own decision process rather than as human-authored documentation [1][2]. Every .c file in the bot's source directory carries an identical, unremoved safety disclaimer stating that the code "is for educational and authorized security research only" and that unauthorized use "is strictly prohibited and may be illegal" — boilerplate consistent with the kind of caveat a general-purpose LLM appends to security-adjacent code generation. Unit 42 characterizes the operator as having trusted the LLM's output and moved on without the review that would have caught it [1][2].

The consequences of this workflow extend beyond stylistic tells into functional defects. The C2 server's authentication module claims in its own comments to implement Argon2id password hashing — "HashPassword creates a cryptographically secure password hash using Argon2id" — but the actual implementation uses PBKDF2 while formatting its output to visually resemble Argon2id, complete with LLM-generated constants and format strings that do not correspond to a real Argon2id implementation [2]. This looks like a case of a model asked to implement a specific, well-known cryptographic primitive instead producing a plausible-looking but functionally different substitute — a failure mode that would be worth watching for in other LLM-assisted codebases handling less common cryptographic bindings, though we are not aware of published data quantifying how often it occurs.

A separate and more consequential defect appears in the framework's obfuscation routine. The bot's toggle_obf() function splits a 32-bit key, 0xDEDEFB4F, into four bytes and XORs a data buffer against them sequentially; because two of those bytes are identical (0xDE), and any byte XORed against itself cancels to zero, the effective runtime key collapses to a single byte, 0xB4, rather than the intended four-byte key [2]. Compounding this, an offline encryption utility used to prepare the framework's obfuscated string table was never updated when the developer changed the key from an earlier value, 0xAF, to 0x4F — leaving nine string table entries, including the IRC server address, IRC channel name, the hardcoded HTTP fallback URL, and four exploit payload templates targeting ThinkPHP, GPON, and Realtek UPnP devices, encrypted with a stale key that the runtime can no longer correctly decode [2]. The result is that the IRC and HTTP C2 fallbacks, along with four of the framework's exploit payloads, produce garbled, non-functional output whenever invoked. Unit 42's exploit virtual machine tells a similar story: it checks incoming exploit definition files against an expected magic value of 0x4558504C ("EXPL") but the shipped .expl files were generated with a different value, 0x54555845 ("TUXE"), so the VM rejects every exploit definition it was built to run despite the definitions themselves being technically complete [2]. Across the codebase, Unit 42 counted 16 hardcoded exploit functions covering 13 CVEs that are fully implemented but never invoked from any code path, and 27 exploit VM definition files covering a further 13 CVEs plus two non-CVE targets that fail to load due to the magic-value mismatch — dead code spanning more vulnerabilities than the framework can currently exploit in practice [2].

None of these defects reflect a lack of technical ambition. The framework's DDoS engine alone registers 78 distinct attack vectors mapped to six handler functions, and 47 of those vectors — covering Layer 7 HTTP methods such as GET floods, Slowloris, WordPress XML-RPC pingback abuse, and Cloudflare-bypass variants — are fully coded but never reached because of a routing defect that sends all of them to the generic TCP SYN flood handler instead [2]. Unit 42's own assessment is that TuxBot v3 "aim[s] to go beyond the usual Mirai fork with its encrypted C2, its DGA, and a modular exploit system, even though that system does not work yet in the version we recovered" [1][2]. Perhaps most notably, Unit 42 researchers reported that they were able to fix the broken IRC fallback and the XOR decryption defect themselves using "a few targeted LLM prompts" — a demonstration that the same tooling responsible for

the bugs could trivially resolve them, and a strong indication that corrected, more complete versions of TuxBot likely already exist in operational use even though the sample Unit 42 recovered does not represent one [2].

Context Within the Broader AI-Assisted Malware Trend

TuxBot v3 is a data point within a documented and accelerating pattern rather than an isolated curiosity. Arctic Wolf Labs, tracking malware repository submissions against AI-focused YARA rules over a rolling window from February 2025 through February 2026, identified more than 22,000 distinct files matching those rules, of which 39 percent had zero detections from signature-based antivirus engines at the time of collection — evidence that AI-assisted code generation is producing samples structurally novel enough to evade legacy detection rather than simply reusing known malware families [3]. The same analysis found that only 1.4 percent of those 22,000 samples could be linked to known advanced persistent threat groups or established financially motivated criminal organizations, meaning the overwhelming majority originated from less-resourced or lower-skilled actors, which is consistent with the pattern this case illustrates: an LLM's ability to generate working, if imperfect, exploit code, cryptographic routines, and protocol implementations lowers the barrier for exactly this population to produce a functional botnet framework [3]. Arctic Wolf also documented a sharp inflection in submission volume following the January 2025 release of DeepSeek's R1 model, with malicious submissions rising from 155 files in February 2025 to a peak of 2,454 files by August 2025 [3]. TuxBot's development timeline, beginning in January 2025 and accelerating through late 2025 and early 2026, sits squarely within that same adoption curve.

What TuxBot v3 adds to this broader trend is a rare, fully reconstructed view of the operator's actual workflow: an unsophisticated actor cloning a public DDoS toolkit, directing an LLM to port and extend it across languages and architectures, and shipping the output with functional gaps, cryptographic substitutions, and leftover reasoning traces that a careful human reviewer would have caught. The security implication is not that TuxBot itself is a uniquely dangerous botnet — in its recovered state, it is a partially broken one — but that the underlying workflow is repeatable and plausibly low-cost, given the absence of custom tooling beyond an LLM and a cloned public toolkit, and is evidently already delivering roughly 70 percent functional capability to an operator who could not have written the encrypted C2 protocol, the domain generation algorithm, or the multi-architecture exploit code unassisted [2].

Recommendations

Immediate Actions

Organizations operating internet-facing IoT and embedded devices — network cameras, routers, DVRs, and similar SOHO equipment — should treat TuxBot's documented infection vectors as an active threat and audit exposure accordingly. This includes confirming that Telnet and SSH management interfaces are disabled or restricted from public internet reachability, that default and vendor-standard credentials have been changed on all internet-facing embedded devices, and that Android Debug Bridge interfaces on any exposed device are disabled, since TuxBot's ADB scanner specifically targets that service [1][2]. Security teams should ingest the published indicators of compromise, including the C2 IP address 209.182.237[.]133, the shared dropper IP 185.10.68[.]127, and the domain digikalas[.]online, into network monitoring and blocklist tooling [2].

Short-Term Mitigations

Network defenders should monitor for the specific behavioral signatures Unit 42 documented: outbound connections attempting DNS TXT queries to c2.tuxbot.local, DGA-pattern domain lookups following the SHA-512-derived naming convention, and unusual outbound TCP traffic on ports 1999, 31337, 2222, 9999, or 13337 originating from IoT or embedded devices that should not normally initiate such connections [2]. Because TuxBot's competitor-killing routine and anti-VM checks scan for signatures of Mirai, QBOT, and related families, security teams responding to a suspected compromise should assume the presence of one botnet does not exclude the presence of another, and should conduct full forensic review of affected devices rather than treating remediation of the first identified indicator as complete [2].

Strategic Considerations

TuxBot v3 is a useful illustration of a pattern this document's other evidence also points toward: LLM assistance appears to be lowering the skill floor required to assemble a multi-architecture, encrypted botnet framework, even as it introduces its own class of reliability defects. This document does not have measurement data establishing how widespread that shift is across the industry, but the Arctic Wolf findings above are consistent with it. Defenders should not assume that AI-assisted malware is inherently less capable because it contains bugs; Unit 42's own demonstration that a handful of follow-up LLM prompts resolved multiple structural defects shows that the gap between a broken development snapshot and a fully functional release can close quickly and cheaply [2]. Organizations building or

evaluating AI-assisted software development workflows — including their own — should take the specific failure modes documented here as a caution: chain-of-thought leakage into shipped artifacts, silent substitution of a requested cryptographic primitive with a superficially similar but functionally different one, and magic-value or key-versioning mismatches introduced when a model regenerates one file without correspondingly updating a dependent file. These are generic risks of unreviewed AI-assisted code generation that apply equally to defensive and legitimate commercial software development, not just to malware authorship.

CSA Resource Alignment

TuxBot v3's core infrastructure pattern — mass-scanning and compromise of internet-facing SOHO and IoT devices for use as a distributed capability — maps directly to the threat class CSA documented in its June 2026 research note, JDY Botnet: China-Linked SOHO Scanning Targets U.S. Military [4]. That analysis, while examining a nation-state reconnaissance botnet rather than a DDoS-for-hire framework, reached the same underlying conclusion this case reinforces: internet-facing edge and IoT devices remain chronically under-patched, under-monitored, and attractive to operators of every sophistication level, from state-linked reconnaissance actors to commodity Keksec-affiliated DDoS operators. The recommendations in that note — auditing internet-facing devices for firmware currency, restricting administrative interfaces from public reachability, and applying Zero Trust least-privilege network segmentation to IoT device traffic — apply without modification to organizations assessing TuxBot exposure.

CSA's AI Controls Matrix (AICM) v1.1 provides the relevant control framework for the AI-assisted development dimension of this case. AICM's coverage of secure AI-assisted software development and supply chain integrity is directly applicable to the failure pattern documented in TuxBot's source code: an operator who generated cryptographic and protocol implementations via LLM without validating that the output matched its stated specification, and who shipped internal model reasoning and safety disclaimers into production artifacts without review. Organizations using AI coding assistants for legitimate development — not just malware authors — face the identical risk of silently incorrect cryptographic substitutions and unreviewed generated output, and AICM's domains covering application security and AI system lifecycle management give security teams a structured basis for requiring human review gates on AI-generated code before it reaches production, regardless of whether that production system is a commercial product or internal tooling.

Earlier CSA IoT security guidance, including Taking Control of IoT: An Enterprise Perspective [5], remains relevant background for organizations building enterprise IoT device management programs, reinforcing that the device hygiene failures TuxBot exploits — default credentials, exposed management protocols,

and inconsistent patching — are long-documented gaps rather than novel weaknesses.

References

[1] The Hacker News. "TuxBot v3 Evolution Shows Signs of LLM-Assisted IoT Botnet Development." The Hacker News, July 2026.

[2] Unit 42 / Palo Alto Networks. "TuxBot v3: Inside an IoT Botnet Framework With LLM-Assisted Development." Unit 42, July 15, 2026.

[3] Arctic Wolf. "The AI Malware Surge: Behavior, Attribution, and Defensive Readiness." Arctic Wolf Labs, March 24, 2026.

[4] Cloud Security Alliance. "JDY Botnet: China-Linked SOHO Scanning Targets U.S. Military." CSA AI Safety Initiative, June 11, 2026.

[5] Cloud Security Alliance / Hillary Baron. "Taking Control of IoT: An Enterprise Perspective." Cloud Security Alliance, 2019.