PromptLock Ransomware

Cybanetix

V1.0 27/06/2025

Cybanetix Limited
cybanetix.com
Registered at:
The Coade
Level 9
98 Vauxhall Walk
London
SE11 5EL
Company Registration: 10558582
VAT Number: GB 262502430

CYBANETIX

Table of Contents

PromptLock Ransomware Technical Analysis | OPTIX Briefing

Timeline of Discovery and Development

- July 2025 – Emergence of AI-Assisted Malware: The concept of using AI in malware gained traction when CERT-UA (Ukraine) reported *LameHug*, an LLM-powered malware used by Russian APT28, which generated malicious Windows commands on the fly. This demonstrated that nation-state actors were experimenting with AI-driven attacks prior to PromptLock's appearance.
- August 2025 – PromptLock Discovered by ESET: ESET researchers hunting through VirusTotal uncovered a new ransomware sample that didn't match known malware signatures. On August 27, 2025, ESET announced this find as *PromptLock*, heralded as the first *"AI-powered ransomware"* observed. Early variants of PromptLock (both Windows and Linux Golang binaries) had been uploaded to VirusTotal by a U.S.-based user. ESET noted that it had not been seen in any in-the-wild attacks, implying it was likely a proof-of-concept or developmental malware.
- Late August 2025 – Public Disclosure: ESET's public report highlighted PromptLock's novel use of a generative AI model to autonomously decide whether to exfiltrate or encrypt files during an infection. The news raised industry-wide concern that a *"turning point"* had been reached where AI could supercharge ransomware capabilities. Media coverage (e.g. DarkReading and BleepingComputer) amplified the alert, though experts noted the implementation appeared unfinished and not yet actively deployed by criminals.
- Early September 2025 – Academic Origins Revealed: On September 5, 2025, researchers from NYU's Tandon School of Engineering came forward to claim authorship of PromptLock as part of an academic project dubbed "Ransomware 3.0". They explained that PromptLock's code was an experimental prototype uploaded to VirusTotal during testing, unbeknownst to ESET. The academic team's goal was to illustrate the potential harms of AI-orchestrated malware in a closed-loop attack without human intervention. ESET updated its reports to acknowledge the malware's origin as a *proof-of-concept* from this research, while affirming that PromptLock *"represents the first known case of AI-powered ransomware"*.
- Current Status: As of late 2025, PromptLock remains non-attributed to any criminal group or APT and has not been seen in active cyberattacks. It is best

understood as a concept demonstration. However, its emergence is a warning sign: it closely followed real-world LLM-based malware (LameHug by APT28) and exemplifies the rising trend of threat actors exploring AI for more adaptive, cross-platform attacks. Security experts anticipate that PromptLock's capabilities could be adopted or refined by malicious actors in the near future, marking a new era of "ransomware 3.0" threats.

Technical Breakdown of PromptLock

PromptLock is a ransomware toolkit with an unprecedented architecture: it offloads much of its malicious logic to an AI language model rather than relying on solely pre-written code. Below is a detailed breakdown of how PromptLock functions:

- Instead of embedding fixed ransomware routines, PromptLock contains *hard-coded natural-language prompts* that it feeds to a generative large language model (LLM) at runtime. The malware leverages OpenAI's gpt-oss:20b model (an open-source 20-billion-parameter language model) via the Ollama API to produce malicious Lua scripts on the fly. In essence, PromptLock conducts a prompt-injection attack on the LLM: it instructs the AI to act as a code generator and produce specific attack scripts (written in Lua) which PromptLock then executes in memory.
- PromptLock's wrapper malware is written in Go, allowing compilation for Windows, Linux, and macOS systems from a single codebase. ESET discovered both Windows (WinGo/Filecoder.PromptLock.A) and Linux (Linux/Filecoder.PromptLock.A) variants on VirusTotal. The Golang binary includes the logic to interface with the LLM and carry out the AI's outputs. This cross-platform design dramatically broadens the potential attack surface, as the same malware build can adapt to multiple operating systems.
- Once executed on a victim machine, PromptLock uses AI-generated Lua scripts to scan and enumerate the local filesystem in search of valuable or sensitive data. One of the static prompts instructs the LLM to generate Lua code that recursively searches files and directories, looking for indicators of sensitive content (e.g. keywords, file types) and logs matching filenames to a list (e.g., target_file_list.log). This represents the Reconnaissance/Discovery phase of the attack: the malware dynamically maps out target data without needing a pre-programmed file list. Notably, the LLM's *reasoning* capabilities allow it to decide which files are likely important (for example, documents with financial terms, databases, source code, etc.) – a task traditionally requiring human planning.
- If high-value files are found, PromptLock can turn into a data thief. Based on predefined prompt logic, the AI autonomously decides whether to exfiltrate or

encrypt each file it finds. For files flagged for exfiltration, the malware's next step is to have the LLM generate a script to upload those files to a remote server. In the prototype, the Lua payload uses the system's command-line (via curl commands) to send each targeted file to an attacker-controlled server address with an API key. This means PromptLock can perform data theft (double extortion) by exfiltrating sensitive information before or instead of encryption. The exfiltration mechanism is executed locally (the malware calling curl for each file), so it does not rely on any particular C2 protocol beyond normal HTTPS outbound traffic.

- For files designated for ransom, PromptLock instructs the LLM to generate an encryption routine. Uniquely, it uses the SPECK 128-bit cipher (a lightweight block cipher) implemented in Lua. The choice of SPECK is unusual for ransomware (more commonly AES or ChaCha20 are used) and is likely due to its simplicity and small code size, making it easier for an AI to implement on the fly. The AI-generated script reads each target file and encrypts it in place using the SPECK algorithm, leveraging a provided key. The encryption is fast and cross-platform (pure Lua code), albeit *cryptographically weaker* than modern standards. This step corresponds to the Impact/Encryption phase of the attack – PromptLock can lock the victim's data to demand a ransom.
- PromptLock even uses the LLM to compose ransom demand messages. The malware's prompt includes instructions to the model to act as a "cybersecurity expert" and draft a plausible extortion note for the victim. The AI can incorporate specific details such as the filenames of stolen data or the victim's organization name into the note, making it personalized and more convincing. Notably, the ransom note generated in the discovered sample contained a *hard-coded Bitcoin address* for payment – specifically, the first Bitcoin address ever created (associated with Satoshi Nakamoto) was used as a placeholder. This infamous address was likely inserted as an Easter egg or to signal the prototype status (no real ransom intended). The ransom notes are saved to a file (e.g., note.txt) on the victim system. In a real attack, this note would instruct the victim to pay a ransom (though in PromptLock's case, any demanded payment would have gone to the dummy Satoshi address).
- PromptLock's code also hints at a possible data destruction feature, though it was inactive in the samples we have handled. ESET observed that the malware includes a *"destructive function"* which was not enabled. The academic authors describe that after exfiltration and encryption, the malware could optionally wipe the files – the AI can generate a script to securely overwrite and delete each file listed in target_file_list.log. This would function as a "kill switch" or punitive action if the ransom isn't paid (or simply to complicate recovery). In the

PromptLock proof-of-concept, the data destruction phase was likely disabled to avoid accidental harm during testing. However, the presence of this code underscores how easily an AI-driven malware could adapt its payload to include sabotage (MITRE T1485).

- One of PromptLock's defining features is its use of a locally accessible AI model. The malware does not query a cloud AI service during execution; instead, it leverages the Ollama platform to run the LLM in a contained environment. In the prototype, the OpenAI gpt-oss:20b model was accessed through an API, with the model's outputs *"served directly to the infected device."*. This was achieved by running an instance of the LLM (20-billion parameter model) on a server and having the malware connect to it over the network. To avoid downloading the large model onto each victim machine, the attackers can tunnel the victim's connection to a remote server running the LLM. In practice, the PromptLock binary establishes a proxy or SSH tunnel from the compromised host out to the attacker's LLM server. The model (hosted on the attacker side) then receives the prompts and returns Lua code to the malware in real time. This design effectively makes the AI model part of the command-and-control (C2) infrastructure: the malware queries an external AI service for instructions (scripts), akin to receiving commands from a C2 server. Because the LLM is self-hosted by the attacker and accessed via API calls, network defenders might see unusual traffic to an unknown server or on an uncommon port (Ollama's API runs on port 11434 by default).
- The PromptLock samples analyzed did not exhibit persistence mechanisms – there was no code to survive reboot or maintain long-term foothold. It appears the malware was designed to run as a one-shot payload (perform recon, exfiltrate/encrypt data, drop a note, optionally destroy data, then exit). This is consistent with its proof-of-concept nature and the fact that it was not used in a campaign that required stealth or longevity. In an operational scenario, attackers could easily add common persistence techniques (e.g. registry Run keys, scheduled tasks, launch agents) to PromptLock. The absence of persistence in the discovered samples suggests the authors were focused on demonstrating the LLM-driven attack loop rather than stealth or resilience. Defense note: The lack of persistence actually makes the malware easier to contain – if detected and removed, it will not reboot itself – but a more sophisticated variant could be persistently lurking if combined with typical droppers or loaders.
- A core implication of PromptLock's design is that *the malware's behavior can change with every execution*. The Lua scripts generated by the AI will not be identical on each run – the LLM might produce functionally similar code with different syntax, or handle different files depending on context. This polymorphic

nature means traditional Indicators of Compromise (IoCs) like specific file hashes or static strings in memory are far less useful, since the *malicious script content varies between infections*. As ESET noted, *"indicators of compromise may vary between executions"* because the AI-generated code adapts to the environment. In effect, PromptLock behaves somewhat like fileless malware or a living-off-the-land script: the actual malicious logic is synthesized at runtime, making signature-based detection extremely difficult. However, the PromptLock binary itself (the Golang container and its embedded prompts) remains constant across runs. This gives defenders a single static artifact to focus on (see Detection section below). The use of an AI model also helps PromptLock evade certain defenses by *outsourcing complex logic to the LLM*: for example, there is no large encryption routine in the binary to analyze – it's generated on the fly – potentially confusing static analysis. Additionally, by using an encrypted tunnel to communicate with its AI model, PromptLock can hide its C2 communications within seemingly benign traffic (e.g. an HTTPS connection to a non-domain endpoint).

Associated Threat Actors and Attribution

PromptLock has not been attributed to any known cybercriminal or APT group. In fact, it was created by academic researchers as a demonstration, rather than by a threat actor. ESET initially found the code uploaded from the U.S., but there was no indication it had been used in attacks or linked to an existing hacking group. After the NYU team claimed credit, it became clear PromptLock was essentially an *academic proof-of-concept (PoC)*, not a weapon actively deployed by criminals.

- The multi-platform ransomware functionality is reminiscent of advanced ransomware groups, but to date no ransomware gang has been publicly tied to PromptLock. It's conceivable that organized cybercriminal groups (ransomware-as-a-service gangs) will draw inspiration from PromptLock's design in the future, given its potential to evade defenses.
- Notably, a month before PromptLock's disclosure, APT28 (a.k.a. Fancy Bear) – a Russian state-backed group – *deployed an LLM-powered malware named "LameHug"* against Ukrainian targets. LameHug used an AI model (via the HuggingFace API) to generate malicious Windows shell commands dynamically, and it was delivered through phishing emails to government agencies. This real-world attack by an APT group underscores that nation-state adversaries are already experimenting with AI to augment malware. While LameHug and PromptLock were developed independently, they both validate the feasibility of AI-orchestrated cyberattacks. (CERT-UA attributed LameHug to APT28 with medium confidence.)

While PromptLock itself is not known to be in the hands of cybercriminals or spy agencies at this time. However, the techniques it showcases are already surfacing in related forms. Security researchers warn that both financially motivated gangs and APT actors are likely to incorporate AI/LLM components into their malware in the coming months and years. PromptLock has effectively given a blueprint of "what's possible," and it has raised alarms within the security community about the next generation of threats.

Tactics, Techniques, and Procedures (TTPs) – MITRE ATT&CK Mapping

PromptLock's behavior spans multiple phases of the attack lifecycle, enabled by its AI-driven adaptability. The key TTPs observed or inferred in PromptLock (and their corresponding MITRE ATT&CK techniques) include:

- Initial Access (TA0001) – *Delivery & Execution:* Since PromptLock was not deployed in the wild, its initial access vector was not observed. In concept, the malware could be delivered via typical ransomware vectors. For example, Spearphishing Attachment (T1566.001) is a likely method (similar AI malware LameHug arrived via phishing emails with malicious attachments). Another possibility is Supply Chain or Trojaned Software (T1195), as the academic authors noted one could hide such AI prompts in software that promises LLM features. User Execution (T1204) is required – the victim must run the PromptLock binary (e.g. by opening a fake installer or document) to trigger the attack.
- Execution (TA0002) – *Dynamic Scripting:* PromptLock executes its malicious behavior via Command and Scripting Interpreter (T1059). Specifically, it generates Lua scripts in memory and runs them to perform its tasks. This is akin to fileless script execution orchestrated by the malware. The Golang dropper itself calls the Ollama API and then likely invokes a Lua engine or system commands to execute the returned Lua code. Each script run is tailored to the victim's system (AI-driven), making PromptLock's execution flow highly dynamic. Additionally, the malware spawning system commands (like curl for exfiltration) is part of its execution technique (overlaps with OS Command Execution – T1059.003 using the shell via AI-generated instructions).
- Persistence (TA0003) – *No Persistence:* PromptLock did not implement explicit persistence (e.g., no autoruns, services, or startup items observed). In a real scenario, attackers could add Boot or Logon Autostart (T1547) techniques such as Registry Run keys (on Windows) or Launch Daemons (on macOS) to ensure

the ransomware persists. The lack of persistence in PromptLock's sample indicates the malware runs in-memory during one session, which can limit its footprint but also means if the system reboots before it finishes, the malware would not continue running.

- Privilege Escalation (TA0004) – *No Elevation:* The available information doesn't indicate that PromptLock performs privilege escalation. It likely runs with user-level permissions. If higher privileges were needed (e.g., to access certain files or terminate security software), an attacker might need to include an exploit or prompt the AI to attempt some privilege escalation (though that was not part of the PoC). In current form, PromptLock operates at whatever privilege level it starts with; thus, if executed by a user, it might only impact user-accessible files. (Future AI malware could integrate privilege escalation via LLM – e.g., having the AI generate code to exploit known vulnerabilities – but this was beyond PromptLock's demonstrated scope.)
- Defense Evasion (TA0005) – *Obfuscation & Polymorphism:* PromptLock largely evades defenses through Obfuscated Files or Information (T1027) – not by packing or encryption, but by *virtue of its AI-driven polymorphism*. The actual malicious payload (Lua code) exists only ephemerally and can be different on each run. This defeats traditional file-signature or hash-based detection. Additionally, because the heavy-lifting is done by the LLM, the static binary may look relatively benign (no obvious ransomware loops or large suspicious imports). The use of common tooling (like curl) and a legitimate model API can help it blend in. One could also consider Non-Standard Interpretation (T1059) as it uses Lua – a scripting language that is less commonly monitored on endpoints. The variability of IoCs and the indirect way code is executed pose a significant evasion challenge for static analysis and even some behavior-based detection.
- Discovery (TA0007) – *System and File Discovery:* PromptLock performs extensive File and Directory Discovery (T1083) and possibly System Information Discovery (T1082). The AI-generated reconnaissance script traverses the filesystem, identifies files of interest (documents, databases, etc.), and may glean system info (like user directories, OS type to tailor commands). Essentially, the malware conducts automated recon to understand its environment and choose actions (exfiltrate or encrypt) based on what it finds. This is a step beyond typical ransomware, which often encrypts everything blindly; PromptLock attempts a targeted approach.
- Collection (TA0009) – *Data Staging:* After discovering sensitive files, PromptLock collects them for exfiltration. It uses Data from Local System (T1005) – reading the contents of files deemed valuable. The files to steal are listed in target_file_list.log as a staging mechanism. Then, using an AI-scripted routine, it

prepares these files for upload (ensuring full file paths, etc.). This corresponds to Data Staging (T1074), where the malware organizes data prior to exfiltration (in this case, creating a list and then uploading each file). PromptLock's design shows the AI can make decisions about which files to collect, adding a layer of "intelligence" to the Collection phase that is typically static in other malware.

- Exfiltration (TA0010) – *Automated Exfiltration & C2:* For data exfiltration, PromptLock utilizes Exfiltration Over C2 Channel (T1041) or more specifically Automated Exfiltration (T1020). The AI-generated Lua code invokes outbound network calls (via curl) to send files to an attacker-controlled server. Because the malware already has a network channel open to the LLM server (its proxy to the Ollama API), it could use that channel for exfiltration or a separate HTTPS POST to a hardcoded endpoint. The academic case study indicates the prompt included concrete server details for file uploads. In practice, these could be attacker infrastructure (e.g., a cloud storage or web server) acting as the C2/exfiltration point. PromptLock likely uses standard ports (80/443 via curl) for exfiltration traffic, which helps it blend with normal traffic. The data exfil is scripted to use full file paths and an API key, ensuring the attacker can authenticate and receive the files properly.

- Command and Control (TA0011) – *LLM as C2:* PromptLock blurs the line between payload and C2. Its primary "command" mechanism is the LLM. We can map this to Application Layer Protocol – Web APIs (T1071.001) since it communicates with the AI model using HTTP/JSON calls via the Ollama REST API. The model essentially sends back commands (in the form of Lua code). Unlike typical malware that reaches out to a C2 for instructions, PromptLock reaches out to an AI service. When the LLM is hosted remotely, this traffic can be considered C2 traffic. The use of a localhost or remote port 11434 (default for Ollama) is notable – if the attacker's LLM is remote, the malware might connect to that port through a tunnel. This is an unusual C2 method that might be classified as Non-Standard Port (T1571) or even Domain Fronting or Payload (if the AI service were on a cloud domain). Additionally, after initial infection, PromptLock no longer requires human operator input – the *LLM orchestrator takes over, autonomously driving the attack lifecycle*. This closed-loop operation is novel: the "C2" (the AI) not only delivers static commands but can adaptively generate new ones based on the environment (which falls under Automated Workflow that isn't yet a defined ATT&CK technique, but is an emerging concept in autonomous malware).

- Impact (TA0040) – *Data Encryption & Destruction:* PromptLock squarely targets data integrity and availability in the Impact phase. It performs Data Encrypted for Impact (T1486) by encrypting files using SPECK-128 via the AI-generated routine. This renders the victim's data inaccessible without the decryption key (which the

attacker presumably holds externally). The inclusion of a ransom note and payment instructions is characteristic of ransomware's Impact: *Indicator of Compromise* for this is the note.txt file and the extension/marker of encrypted files (PromptLock might not change filenames but encrypted content and the note are giveaways). Moreover, although not active in the discovered sample, PromptLock was capable of Data Destruction (T1485) – wiping files after exfiltration. Had this feature been enabled, it would permanently destroy data (beyond just encryption), increasing the impact on the victim (and removing the attacker's leverage if ransom was not paid, which suggests it might be used as a punitive action or if the goal is pure sabotage). The combination of theft and encryption aligns with double-extortion ransomware tactics, and the potential destruction adds a *triple* extortion or nation-state sabotage element.

- Another impact aspect is Financial Extortion (T1486 – Impact: Resource Hijacking/Threat) as the attacker demands a ransom in cryptocurrency. PromptLock's AI writes a personalized ransom note threatening to expose stolen data and keep files locked. This psychological impact (pressure via personalized threats) is a tactic to increase the likelihood of payment. While MITRE categorizes the encryption itself under T1486, the act of extortion and threatening release of data is part of the modern ransomware playbook.

The table below maps PromptLock's key behaviors to corresponding MITRE ATT&CK techniques:

Tactic	Technique	PromptLock Behavior
Initial Access	T1566.001 – Spearphishing Attachment (likely)	(Not observed; conceptually delivered via phishing emails similar to LameHug or trojanized apps)
Execution	T1059 – Command and Scripting Interpreter	Runs AI-generated Lua scripts in memory to execute payload
	T1059.003 – Windows Command Shell	Uses shell commands (e.g., curl) generated by LLM for exfiltration
Persistence	— *None observed* —	No persistence in PoC (no autorun). Could be added in real attacks (T1547, etc.)
Discovery	T1083 – File and Directory Discovery	Enumerates filesystem for sensitive files via Lua script

	T1082 – System Info Discovery (likely)	Gathers info about system/user to tailor attack (implied by adaptive behavior)
Collection	T1005 – Data from Local System	Reads content of files deemed sensitive
	T1074 – Data Staging	Writes list of target files to target_file_list.log for processing
Exfiltration	T1041 – Exfiltration Over C2 Channel	Uploads files to remote server via scripted curl calls (over HTTP/HTTPS)
	T1020 – Automated Exfiltration	Automatically exfiltrates selected files without human step-by-step control
C2	T1071.001 – Application Layer Protocol (Web API)	Communicates with LLM over HTTP API (Ollama on port 11434)
	T1571 – Non-Standard Port	Tunnels traffic to LLM server using port 11434 (not a typical web port)
	— Closed-loop AI Orchestration —	(Novel) LLM acts as automated C2, planning attack steps autonomously
Defense Evasion	T1027 – Obfuscated/Packed Code	Polymorphic code generation – attack scripts vary each execution
	T1218 – Signed Binary Proxy Execution (curl)	Abuse of legitimate tool (curl) for data transfer to avoid detection
	T1497 – Virtualization/Sandbox Evasion	(Potential) AI could adapt if it detects sandbox artifacts (not documented in PoC)
Impact	T1486 – Data Encrypted for Impact	Encrypts files on victim system using SPECK algorithm
	T1485 – Data Destruction	Can wipe files (code present but not activated)
	T1491 – Defacement / Ransom Note	Drops ransom note (note.txt) with threats and payment info
	(Impact - N/A) Personalized Extortion	AI crafts ransom message referencing victim's own data (to maximize pressure)

Indicators of Compromise (IOCs)

Because PromptLock is a proof-of-concept and not widespread, IOCs are relatively limited to the sample artifacts and a few unique behaviors. Security teams should watch for the following PromptLock indicators:

- Malware Sample Hashes: ESET published SHA-1 hashes of PromptLock files found on VirusTotal. These include both Windows and Linux variants. Known hashes (SHA-1) include:
    - 24bf7b72f54aa5b93c6681b4f69e579a47d7c102 – PromptLock sample (Linux)
    - ad223fe2bb4563446aee5227357bbfdc8ada3797 – PromptLock sample (Linux)
    - bb8fb75285bcd151132a3287f2786d4d91da58b8 – PromptLock sample (Linux)
    - f3f4c40c344695388e10cbf29ddb18ef3b61f7ef – PromptLock sample (Linux)
    - 639dbc9b365096d6347142fcae64725bd9f73270 – PromptLock sample (Windows)
    - 161cdcdb46fb8a348aec609a86ff5823752065d2 – PromptLock sample (Windows)
    - 8c7bcafce90f5fb121131ecb27346ecfc6e961c5 – PromptLock sample (Windows)

(All above were detected by ESET as Filecoder.PromptLock.A.) Any file with these hashes (or detected under that name) is malicious.

The PromptLock binaries contain distinctive plaintext strings because of the embedded prompts. For example, references to the model name "gpt-oss:20b" and the Ollama API URL or localhost port 11434 appear in the code. Also, fragments of the prompt text (English instructions for Lua code generation) are hard-coded. These strings can be used in YARA rules. Example suspicious strings in PromptLock:

- "You are a Lua code generator" (or similar AI system prompt phrasing) – instructing the LLM.
- "gpt-oss:20b" – the specific model identifier.
- "Ollama" – the API/engine name.
- "SPECK" – since the Lua script prompt explicitly mentions the SPECK algorithm.
- Filenames like "target_file_list.log" or "note.txt" – artifacts the malware uses.

If such strings are found in an unknown executable, it is a strong indicator of PromptLock or a similar AI-driven malware. EDRs and scanners should flag binaries containing these markers, especially the unique model reference.

- The ransom note content (if generated) contains a specific Bitcoin address – notably the Satoshi Nakamoto Genesis address was used as a placeholder. That address (commonly known as *"1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa"*) is obviously a false payment destination (no attacker can spend from it). Its presence in any ransom note or communication is a telltale sign of the PromptLock PoC. Real attacks would use a different, attacker-controlled address; however, if a ransom note references that exact address or claims Satoshi's identity, it's likely the work of a copycat or a test rather than a serious intrusion.
- PromptLock in its default configuration may cause the infected host to initiate connections on TCP port 11434 – this is the default port for Ollama's API. In a typical enterprise environment, port 11434 is rarely used. Monitoring network logs for outbound or lateral traffic on 11434 could reveal a PromptLock malware attempting to reach an LLM server. Specifically, if a host that doesn't run Ollama locally is trying to connect to another host on 11434, it's suspicious. (Attackers might tunnel this, but any direct use is a clear sign.)
- PromptLock's use of Lua might leave traces. If the malware writes the AI-generated Lua script to disk (even temporarily) or spawns a lua interpreter process, that's an anomaly on most endpoints. For example, security logs might catch a process dropping a .lua file or executing a Lua interpreter (which is uncommon on typical user systems). Additionally, if a security solution monitors in-memory code, detecting large chunks of Lua code or invocation of Lua runtime libraries inside a process could be an indicator.
- During its operation, PromptLock is expected to create or modify a few specific files:

- target_file_list.log – as mentioned, a log of files to act on. Presence of this file (especially in a temp or user folder) with paths of many files listed inside is a strong indicator.
- If the malware stages files for exfil, you might find copies of sensitive files in a temp directory or a large archive being created (depending on implementation). The PoC directly uploads files via curl, so staging on disk might be minimal.
- After encryption, a plaintext ransom note file is typically left in directories. For PromptLock, it specifically writes to note.txt per the research paper. The note will contain a message generated by AI (which may have a certain tone or even odd phrasing). Any ransom note referencing an AI ("I am an automated intelligence…" etc.) or containing the Satoshi address would be a clear sign.
- Since PromptLock uses its own script and SPECK, it might not add a new file extension for encrypted files (unlike many ransomware that append something like *.locked*). Instead, files might simply become garbled. One IOC could be many files suddenly failing to open and containing high entropy data, while their names remain the same. However, this is generic; one might specifically look for references to SPECK in memory or crash dumps.

- The following behavioral IOCs could indicate PromptLock:
  - A process (especially a Golang binary with no prior reputation) performing intensive file read operations across the system (enumerating user directories, etc.), then launching multiple curl processes connecting to external hosts. This combination – mass file access followed by network exfil – is suspicious for ransomware and data theft.
  - Unexplained outbound connections to hosts that are not common for the environment, possibly correlating with the timing of file access. Because the LLM C2 and exfil traffic might go to non-standard or new domains/IPs, any spike of unusual DNS queries or IP connections from a user workstation or server could be a clue.
  - The binary name itself might vary (since it's a PoC it had no fixed name). But if someone finds a Golang binary of ~10-30 MB size with no digital signature, created in 2025, that alone is suspect in many environments.

In summary, defenders should combine file-based IOCs (hashes, YARA strings) with behavioral IOCs (Lua script execution, port 11434 connections, mass file access + curl

network usage, ransom note artifacts) to detect PromptLock. Given the malware's polymorphism, reliance on behavior-based detection is crucial (see next section).

Detection and Mitigation Strategies

Detecting and stopping a threat like PromptLock requires a blend of traditional signature-based methods and advanced behavioral analysis. Likewise, mitigation involves both preventative measures (to reduce the chance of infection) and response strategies (to limit damage if malware runs). Below are recommended detection and mitigation approaches, including use of YARA, Sigma, and heuristics:

Detection Strategies

Security teams can craft YARA rules to identify PromptLock binaries by matching unique strings and characteristics embedded in the file. As noted, the presence of the model identifier gpt-oss:20b or the phrase ollama in a PE/ELF file is a high-confidence indicator.

Also, fragments of the AI prompts (English sentences about file searching, Lua code generation, or ransom note phrasing) can be matched. For example, a YARA rule could trigger on strings like "function encrypt_file(" and "SPECK" and "rb+" (from the encryption script logic) occurring together in a binary. Another string from the ransom note prompt – e.g. "You are a cybersecurity expert" – is unlikely to appear in normal software and would be a strong indicator.

Because the core PromptLock executables remain consistent across runs, such YARA rules can reliably flag the malware before it executes. ESET's Anton Cherepanov noted that *"robust security solutions could flag these executables as malicious"* given their consistent footprint. This implies that endpoint security products can incorporate YARA-like signatures for PromptLock. As a precaution, organizations should update AV/EDR signatures (many vendors by now detect PromptLock as some variant of *Filecoder.AI* ransomware).

Creating custom YARA rules for additional defense in depth (e.g., to scan incoming files or routinely sweep file shares for any PromptLock artifacts) is highly recommended.

To complement static scanning, detection engineers should deploy Sigma rules or SIEM alerts for behaviors associated with PromptLock:

- Alert on a single process reading a large number of user files or scanning directories in quick succession. Many EDRs can trigger if an unusual process opens hundreds of files (especially if it's not an indexer or backup process). A Sigma rule could look for event patterns like: Process = unknown EXE, File accesses > X in Y minutes.

- : If your environment can monitor scripting engine usage, any invocation of a Lua interpreter or the creation of .lua files should be rare. A Sigma rule might detect a process spawning lua.exe on Windows or processes loading lua.dll/liblua.so modules unexpectedly. HivePro specifically recommends tracking Lua script execution as a behavior to catch PromptLock. While PromptLock's Golang binary might have an embedded Lua runtime (no separate process), any attempt by a non-developer tool to run Lua code is unusual in enterprise logs.
- Many Linux and macOS logging solutions can catch command-line usage. A rule could flag a process (that is not a software updater or known service) launching curl with http:// or https:// arguments, especially if it happens repeatedly in a short span. On Windows, if the malware uses something like curl.exe (Windows 10+ include curl) or an equivalent web request, that can be logged via command-line auditing. Detecting a series of HTTP POSTs to an unfamiliar server by a user process should raise an alert as well.
- Monitor network logs for any internal host listening on or connecting to port 11434. If an admin has not explicitly installed Ollama (an AI service) in your environment, this port should normally see no traffic. A Sigma rule can watch firewall or proxy logs for outbound connections to any IP on port 11434, or internal east-west traffic on that port. This can catch both scenarios: malware trying to reach an external AI host, or a compromised machine running an illicit local Ollama server.
- The PromptLock sample might not have a consistent name, but if you observe a binary with a name trying to masquerade as something legitimate but running from the wrong path (e.g., notepad.exe in a Temp folder, or svchost.exe in a user directory), that's a generic red flag. Coupling that with the above behaviors (file access, network) can improve fidelity of detection.
- Because any single behavior of PromptLock might mimic benign activity (e.g., curl is legitimate, file access could be an indexing service), look for combinations: The *same process* doing file enumeration, then network exfil, then writing a note file. A well-tuned SIEM detection could say: alert if a process touches >100 files and within an hour opens a network connection to an IP never seen before, and writes a .log or .txt file with certain keywords. Such multi-factor correlation is key to catching smart malware.

- Deploying *canary files* (decoy files with monitoring) can help detect ransomware. If PromptLock touches certain bait files (like a fake passwords.xlsx placed in a folder, monitored for reads), that could trigger an alert. Since PromptLock's AI might likely pick up juicy filenames, well-crafted lures named "HR_Salaries.csv" or "TopSecret.txt" could be attractive and lead the malware to reveal itself. Honeypot SMB shares or dummy data that should never be accessed could similarly catch its enumeration or exfil attempts.

Mitigation Strategies

- Since the likely entry vector for malware like PromptLock is phishing or user execution, reinforce user awareness. Educate users about not running unknown programs or opening suspicious email attachments – especially tools claiming AI capabilities from untrusted sources. Deploy strong email filtering to block malicious attachments and use sandboxing for file downloads. PromptLock could be hidden in something that lures users with AI (e.g., "AI assistant tool"). Ensuring users treat unsolicited "new AI tool" executables with skepticism is important.
- Limit the damage potential by running users with standard (non-admin) accounts whenever possible. If PromptLock executes with user-level rights, it might be unable to encrypt files outside that user's profile or access network shares (if those require elevated access). Use of application whitelisting or restricting execution in certain directories (e.g., only allow programs from Program Files to run) can prevent a random downloaded PromptLock binary from launching. Tools like Microsoft AppLocker or Windows Defender Application Control can be configured to disallow execution from Temp or Downloads folders, which would thwart many malicious droppers.
- Ensure that *OS and software are up to date*, to reduce the risk of exploits that could be paired with an AI malware (even though PromptLock didn't use one, future variants might). Disable or restrict utilities that PromptLock abuses: for instance, if curl is not needed on Windows endpoints, consider removing or locking it down. On Linux, you can alias curl to a monitoring script in sensitive environments to catch misuse. Likewise, monitor use of tools like PowerShell, which an AI malware might leverage similarly.
- Limit the ability of malware to exfiltrate data or reach out to AI hosts. Implement strict egress filtering – only allow necessary external connections from servers/workstations. Block or require proxy for non-standard ports like 11434; so even if PromptLock runs, it cannot directly connect to the attacker's LLM API without detection. Use network segmentation to prevent an infected machine

from scanning and accessing wide file shares or critical servers. In one scenario, PromptLock could compromise a low-privilege machine but then the AI might instruct it to move laterally. Strong internal segmentation and firewall rules can contain such spread.

- As organizations adopt AI platforms (e.g., local LLMs, Jupyter notebooks, etc.), treat them as potential *dual-use tools*. Implement monitoring and access control for internal AI infrastructure. If a server running Ollama or similar is set up, secure it with authentication and network restrictions so that a malware can't just connect and use it. Conversely, block outbound access to known public AI APIs from sensitive systems – some malware might try to abuse cloud AI (though PromptLock used a local model).
- Use EDR solutions that can automatically *block* or isolate a host when ransomware behavior is detected. Many EDRs have ransomware heuristics (like rapid file modifications) – tune these to include detecting exfiltration patterns. If a host begins enumerating and exfiltrating data, automated playbooks could cut off its network access (to stop exfil) and alert security staff. Given PromptLock's speed (AI-driven decisions can accelerate the kill chain), having automation to respond quickly is crucial.
- Despite best detection efforts, some ransomware may slip through. Mitigation of impact then relies on maintaining comprehensive data backups and a practiced incident response (IR) plan. Follow the 3-2-1 backup rule: keep three copies of data, on two different media, with one offline/offsite copy. Ensure backups are protected from tampering (immutable backups or at least credentials separate so ransomware can't encrypt the backups too). Test your backup restore process regularly so you can recover data if PromptLock (or any ransomware) strikes. In addition, an IR plan should include steps for ransomware: isolating infected machines, preserving forensic data (like memory dumps that might contain the AI prompts or keys), and notification/escalation procedures.
- If using any AI frameworks internally (Ollama, etc.), keep them updated. While PromptLock didn't exploit a vulnerability in Ollama, one can imagine future malware might target vulnerabilities in local AI services. Also, monitor the research community for any defensive tools or patches. For example, the academic team behind PromptLock also researched behavioral signals of such AI malware. Future endpoint solutions might incorporate those findings (like detecting certain AI model load patterns).
- As AI becomes part of the IT environment, consider dedicated monitoring for AI workloads. For instance, if high-end GPUs are present, sudden usage by an unknown process could be a flag. In PromptLock's case, the LLM was run

remotely, but if an attacker tries to package a smaller model locally, you might catch unusual processes consuming GPU/TPU resources on a workstation. Similarly, keep an eye on processes that normally wouldn't do heavy computation suddenly using lots of CPU – that could indicate an LLM is running or being queried.

- Share any new IoCs or behaviors related to PromptLock with the community (through ISACs, CERTs, etc.). As this is an emerging threat area, collective intelligence will help. The academic authors withheld some details to prevent misuse, but if any organization finds PromptLock in the wild, reporting it will be vital to update defensive measures. Stay tuned to reputable sources (ESET research updates, threat intel blogs, etc.) for any evolution of this threat.
- Explore using AI defensively – e.g., employing machine learning models to detect the kind of polymorphic tactics used by PromptLock. Some security teams use generative AI to *hypothesize malware variants* or assist in writing detection rules (even ChatGPT could help formulate YARA/Sigma rules given descriptions of behavior). Just as attackers leverage AI, defenders can too – to simulate attacks, bolster analysis, and automate response. However, remain cautious of over-reliance on AI without human verification, especially since attackers might attempt to trick defensive AI (the "AI vs AI" scenario).

Conclusion

PromptLock represents a pivotal moment in malware evolution – a shift from hand-crafted code to malware that *"writes itself"* using artificial intelligence. Historically, we have seen polymorphic and metamorphic malware that modify their code to evade detection, but PromptLock takes this to the next level by leveraging an external AI model to generate context-aware malicious code in real time. This approach can yield highly adaptive malware that is difficult to detect using traditional signatures, as each infection can look different at the code level.

For cybersecurity professionals, PromptLock is both a warning and an opportunity. It warns that threat actors (from APTs to ransomware gangs) *will likely integrate AI* to streamline and supercharge their attacks. This could lead to ransomware that spreads faster, adapts to each victim automatically, and evades many existing defenses. But it also provides a concrete example that defenders can study to get ahead of the curve. By analyzing PromptLock's TTPs, the community has devised new detection techniques and begun adjusting best practices (as outlined above).

In the cat-and-mouse game of cybersecurity, the arrival of AI-driven malware means defenders must also up their game, potentially using AI for anomaly detection, automating incident response, and focusing on the fundamental behaviors of attacks

rather than static indicators. Ransomware fortified with AI could target a broader range of victims – not just large enterprises but also individuals and small businesses – because the AI lowers the skill barrier for deploying sophisticated attacks. This makes basic cyber hygiene (patching, least privilege, backups, user education) even more crucial for everyone.

Finally, while PromptLock in its current form is a *proof-of-concept*, the threat it signifies is very real. Security teams should treat this moment as a chance to anticipate the "next era" of cyber threats. PromptLock may be the first of its kind, but it will certainly not be the last. By implementing the detection and mitigation strategies discussed – and fostering collaboration across the infosec community – we can prepare for and protect against AI-powered malware, keeping one step ahead of adversaries as they venture into artificial intelligence.