information

MDPI

Article

Social Engineering Attacks Using Technical Job Interviews: Real-Life Case Analysis and AI-Assisted Mitigation Proposals

Tomás de J. Mateo Sanguino

Escuela Técnica Superior de Ingeniería, Universidad de Huelva, Av. de las Artes, s/n, 21007 Huelva, Spain; tomas.mateo@diesia.uhu.es; Tel.: +34-959-217665

Abstract

Technical job interviews have become a vulnerable environment for social engineering attacks, particularly when they involve direct interaction with malicious code. In this context, the present manuscript investigates an exploratory case study, aiming to provide an in-depth analysis of a single incident rather than seeking to generalize statistical evidence. The study examines a real-world covert attack conducted through a simulated interview, identifying the technical and psychological elements that contribute to its effectiveness, assessing the performance of artificial intelligence (AI) assistants in early detection and proposing mitigation strategies. To this end, a methodology was implemented that combines discursive reconstruction of the attack, code exploitation and forensic analysis. The experimental phase, primarily focused on evaluating 10 large language models (LLMs) against a fragment of obfuscated code, reveals that the malware initially evaded detection by 62 antivirus engines, while assistants such as GPT 5.1, Grok 4.1 and Claude Sonnet 4.5 successfully identified malicious patterns and suggested operational countermeasures. The discussion highlights how the apparent legitimacy of platforms like LinkedIn, Calendly and Bitbucket, along with time pressure and technical familiarity, act as catalysts for deception. Based on these findings, the study suggests that LLMs may play a role in the early detection of threats, offering a potentially valuable avenue to enhance security in technical recruitment processes by enabling the timely identification of malicious behavior. To the best of available knowledge, this represents the first academically documented case of its kind analyzed from an interdisciplinary perspective.

Keywords: social engineering; technical job interviews; malicious code detection; obfuscated JavaScript; AI-assisted cybersecurity; crypto wallet compromise; threat mitigation strategies; remote code execution

Academic Editor: Sherali Zeadally

Received: 20 November 2025
Revised: 27 December 2025
Accepted: 16 January 2026
Published: 18 January 2026

1. Introduction

Social engineering attacks have evolved into increasingly sophisticated forms, exploiting not only technical vulnerabilities but also psychological and contextual ones [1]. Notably, it is estimated that 36% of intrusions in 2025 originated from contextual manipulation tactics, surpassing traditional technical vulnerabilities [2]. This trend confirms that attackers are prioritizing psychological manipulation over code exploitation, leveraging implicit trust within professional environments.

One of the most sensitive environments for this type of attack is the recruitment process in the technology sector, where developers are often required to interact with unfamiliar code as part of technical assessments or simulated interviews [3]. In high-pressure and time-sensitive contexts such as technical interviews, it has been documented

that over 80% of users engage with phishing emails and approximately 25% disclose sensitive information. This makes the recruitment process an increasingly concerning and exploited attack vector [4].

This manuscript investigates a reported real-world case in which a developer nearly executed malware embedded within a test project sent by an alleged recruiter from a blockchain company [5]. The attack combined elements of digital legitimacy (i.e., a verified LinkedIn profile, a company with a professional online presence and convincing technical documentation) with a malicious payload concealed in the source code. This payload was designed to execute server-level privileges and exfiltrate critical information such as credentials, files and cryptocurrencies. This type of attack aligns with contemporary strategies of code obfuscation and remote payload delivery, which have been extensively studied in the context of code repositories and development environments [6].

The relevance of this case study lies in its ability to illustrate how attackers are adapting their strategies to the typical behavior of developers, exploiting their familiarity with code repositories, urgency in meeting deadlines and trust in professional platforms. These tactics are grounded in well-established psychological principles, such as manipulating perceptions of authority, urgency, and familiarity to induce rapid and unreflective decision-making [7].

Furthermore, the case study shows the growing role of AI assistants as preventive tools for detecting malicious patterns before code execution [8]. Recent studies have demonstrated that large language models (LLMs) can outperform traditional symbolic analysis techniques in identifying evasive and malicious code [9]. This advancement opens new possibilities for the proactive identification of risks in development environments, particularly when attack vectors are concealed within legitimate work dynamics. Building on this framework, the present study formulates a set of hypotheses aimed at characterizing these types of attacks and evaluating the role of AI assistants as preventative tools in technical recruitment processes.

Research Hypothesis and Objectives

This study is based on the following hypotheses: (i) technical job interviews constitute a vulnerable environment for social engineering attacks, particularly when they involve code execution by the candidate; (ii) the apparent legitimacy of digital environments—such as verified social media profiles or convincing technical documentation—is hypothesized to increase the effectiveness of such attacks by reducing perceived risk; and (iii) AI-based language models may outperform traditional methods in detecting malicious patterns, even in obfuscated code fragments, and their performance can be comparatively evaluated to identify strengths and limitations in real-world preventive detection scenarios.

Based on these premises, the objective of this manuscript is: (i) to analyze a real-world case of a covert attack conducted through a simulated interview; (ii) to identify the technical and psychological elements that contribute to its effectiveness; (iii) to comparatively evaluate the role of various AI assistants as preventative tools; and (iv) to propose recommendations to mitigate risks in recruitment processes involving direct interaction with code. This work is framed as an exploratory case study, aiming to provide an in-depth analysis of a single incident rather than generalize statistical evidence, while highlighting patterns that may guide future research.

To this end, this manuscript is structured as follows: Section 2 reviews the theoretical background on social engineering in recruitment, code obfuscation and AI-based detection. Section 3 outlines the methodology used to analyze the case study. Section 4 presents the experimentation and main findings, while Section 5 discusses

their implications and limitations. Finally, Section 6 summarizes the conclusions and offers recommendations.

2. Related Work

Professional deception strategies have evolved into one of the most effective tactics in the field of cybersecurity, progressively shifting the focus from technical vulnerabilities to psychological manipulation of the user [10]. In particular, recruitment processes in the ICT sector have emerged as a new attack vector, where developers interact with unfamiliar code under conditions of pressure and professional trust [11].

Beyond the technology sector, studies have shown that the problem of manipulation tactics extends to job interviews across multiple industries [12]. These attacks rely not only on technical vulnerabilities but also on direct manipulation through verbal deception, persuasive emails, or identity spoofing. In contrast to attacks in technical contexts, the modalities observed in broader environments tend to be simpler, reflecting a trend toward increasingly sophisticated attacks targeting individuals with specialized training [13].

The psychology of scam plays a central role in social engineering attacks. Tactics such as phishing, pretexting and baiting rely on the exploitation of cognitive biases, emotions and behavioral patterns [14]. The cyberpsychology literature has identified vulnerabilities such as implicit trust, loss aversion and the need for reciprocity as key factors that increase susceptibility to these attacks [15]. In workplace contexts, these tactics are amplified by time pressure, expectations of professionalism and familiarity with digital tools, making recruitment processes particularly prone to manipulation [16].

Several studies have documented how attackers exploit the apparent legitimacy of platforms such as LinkedIn, GitHub or job portals to establish contact with candidates and distribute malware disguised as technical assessments [17,18]. Common tactics include impersonating companies, using verified profiles and simulating interviews; strategies that leverage cognitive biases such as urgency, authority and familiarity [19]. Moreover, recent research has raised concerns about the use of deepfakes and generative AI tools to falsify the identities of candidates or recruiters, further complicating identity verification in remote interviews [20].

In development environments, attackers employ advanced obfuscation techniques to conceal malicious payloads within code packages, scripts or shared libraries. These techniques include string manipulation, dynamic execution, dead code insertion and conditional activation in specific environments [21]. Documented cases in ecosystems such as npm, PyPI and Maven demonstrate how these strategies can evade static analysis and remain undetected during manual code reviews [22]. The delivery of payloads through technical assessments in simulated interviews represents a concerning evolution of these tactics, as it leverages voluntary execution by the candidate in environments with elevated privileges.

LLMs have demonstrated promising capabilities in detecting malicious code, in some cases outperforming traditional analysis methods [23]. Tools such as MalCodeAI and CodeBERT can deconstruct code, identify evasive patterns and generate interpretable explanations of potentially malicious behavior [24]. Moreover, recent studies have proposed LLM-based mitigation frameworks that integrate semantic analysis, threat classification and automated report generation [25]. These solutions offer significant advantages in dynamic environments such as technical interviews, where preventive analysis can be conducted prior to code execution. However, additional techniques—such as payload dispersion across multiple files, misleading variable naming and the insertion of irrelevant computations—can still mislead AI-based coding assistants.

Despite advances in detecting malicious packages across various programming ecosystems (e.g., npm, PyPI and Maven), the scientific literature has yet to systematically address—from an interdisciplinary academic perspective—the use of simulated technical interviews as an attack vector. This modality represents a concerning evolution of professional deception strategies, as it exploits the voluntary execution of code by candidates in privileged environments under the guise of legitimate technical evaluations. Moreover, these tactics effectively bypass both static analysis and manual review by leveraging the implicit trust embedded in professional recruitment processes. In this context, the present study documents—to the best of the authors' knowledge—the first real-world case of malicious payload delivery via a simulated interview and proposes an interdisciplinary framework for its analysis, integrating technical, cognitive and AI-based preventative dimensions.

3. Methodology

This section outlines the technical and procedural anatomy of a targeted social engineering attack disguised as a remote job interview. It presents the phases of interaction between entities, including the identity legitimization strategy, the delivery and execution of malicious code and the forensic analysis of its behavior. For reference, the attack model discussed in this section—had it not been detected in time—is illustrated in a clockwise sequence in Figure 1.

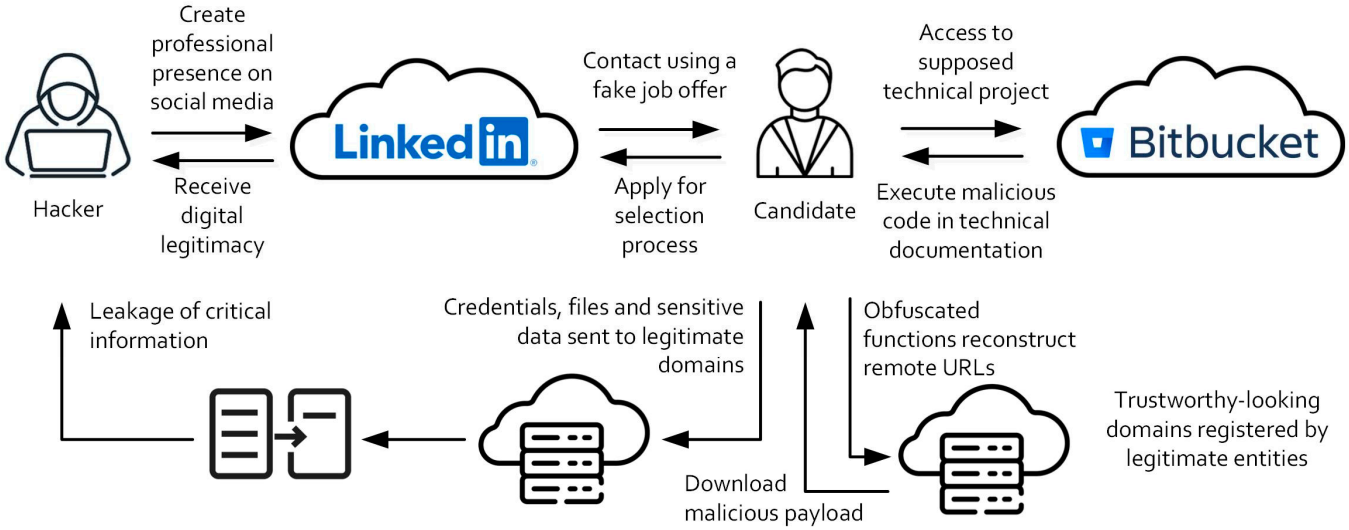

Figure 1. Social engineering attack using technical job interviews.

3.1. Social Engineering Attack

This study documents a real-world case involving an advanced manipulation tactic in which a developer was contacted by the company Symfa, which maintained an active corporate page on LinkedIn featuring posts about blockchain and digital transformation in the real estate sector, multiple connected employees and professional branding. The initial contact was made via a private message from Mykola Yanchii, who introduced himself as the Chief Blockchain Officer (CBO) of the company, in the context of a job offer to collaborate on the development of a platform called BestCity (Figure 2).

This type of targeting is not random; attackers deliberately select blockchain environments due to the high likelihood that candidates possess cryptocurrency wallets or associated private keys, thereby increasing the potential value of the attack. This strategy has been documented in campaigns targeting developers with digital assets, where the

objective is not only the theft of funds but also the acquisition of credentials that enable transaction signing or the modification of smart contracts [26].

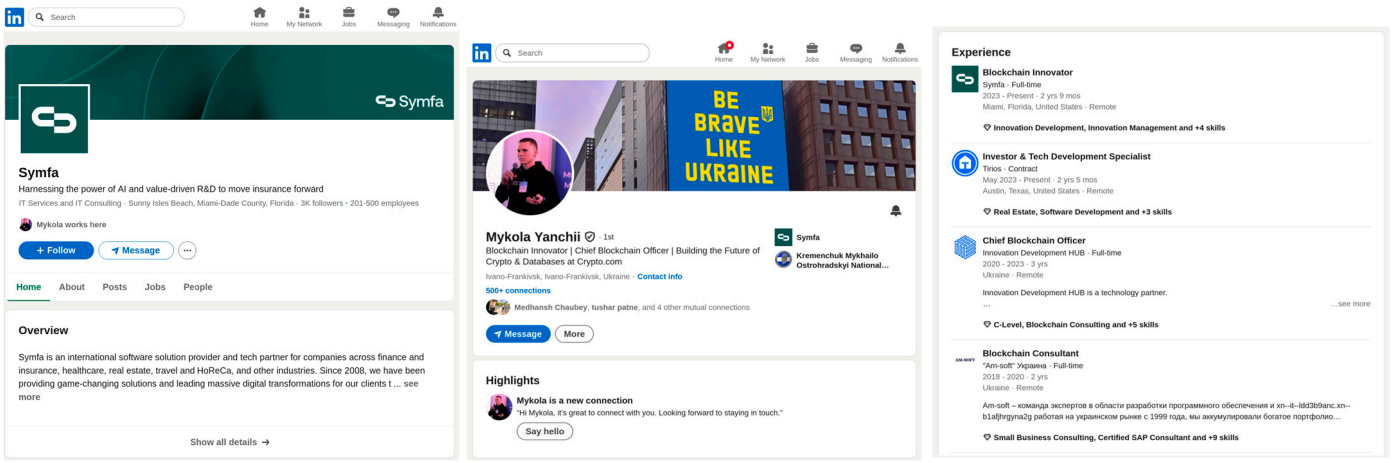

Figure 2. Digital identities of the company and the CBO on social networks.

The initial contact was made via a private message on LinkedIn, followed by an invitation to a technical interview scheduled through Calendly, which reinforced the perceived legitimacy of the process (Figure 3). As part of the preliminary evaluation, the candidate received a technical project hosted in a Bitbucket repository—subsequently deleted—that simulated a React/Node.js application. The repository included standard documentation (README.md), corporate branding assets and a typical technical assessment structure (Figure 4).

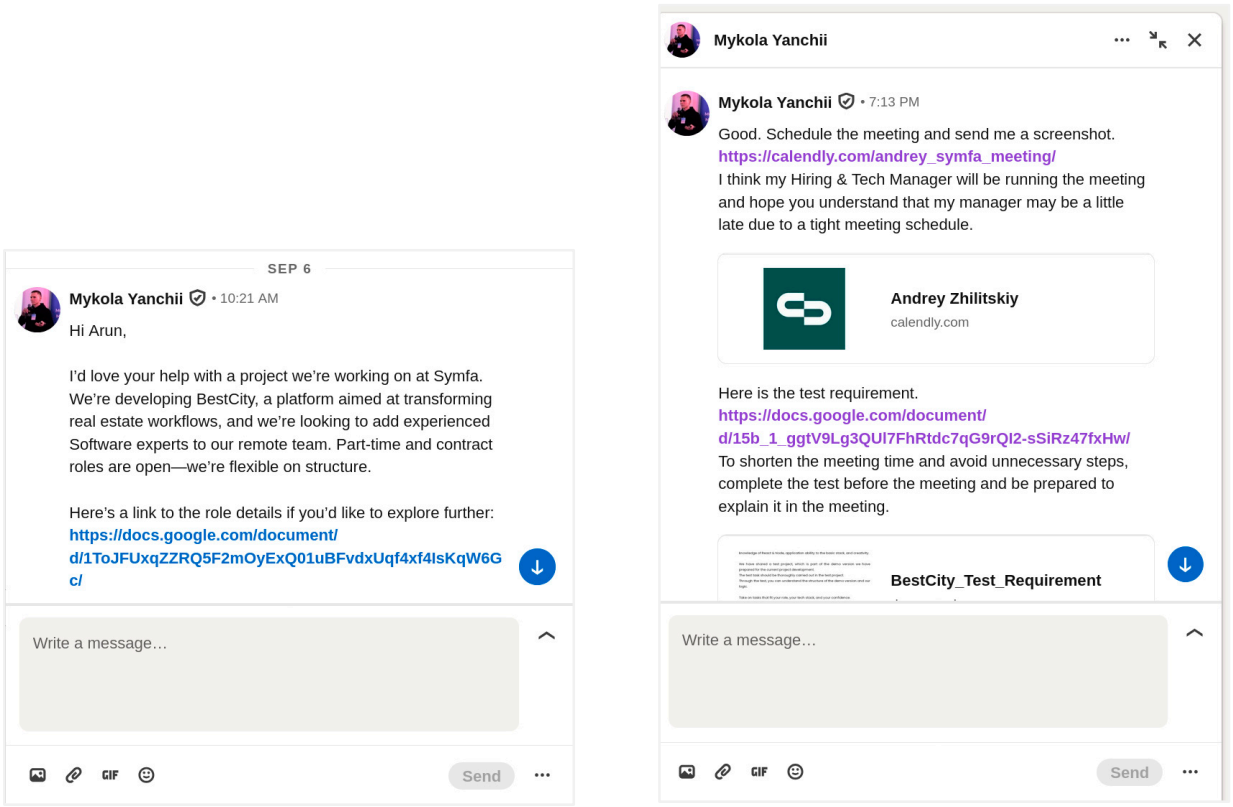

Figure 3. Invitation to fake technical job interview.

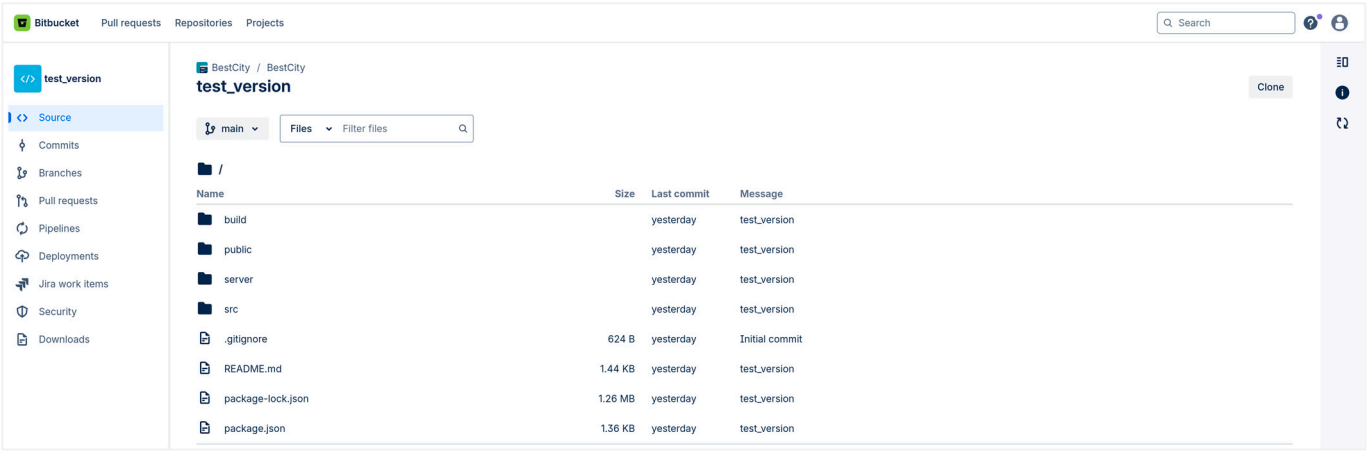

Figure 4. Repository with technical documentation provided by the attacker.

The candidate was given 30 min to review the code prior to the interview, creating a sense of urgency. During this time, he made minor corrections, added a docker-compose.yml file and prepared the environment for execution. However, before running the npm start command, the candidate requested a code review from their assistant, Cursor AI, to check for suspicious patterns: "*Before I run this application, can you see if there are any suspicious code in this codebase? Like reading files it shouldn't be reading, accessing cryptowallets etc*". This AI assistant is a code editor based on Visual Studio Code, integrating models such as ChatGPT and Claude Sonnet to support the entire development lifecycle—from code writing to debugging and refactoring.

3.2. Code Exploitation

Within the file server/controllers/userController.js—which typically handles user-related operations such as registration, authentication or profile management—a fragment of obfuscated malicious code was embedded and concealed among legitimate administrative functions (Figure 5).

JavaScript

```
//Get Cookie
(async () => {
   const byteArray = [104, 116, 116, 112, 115, 58, 47, 47, 97, 112, 105, 46, 110, 112, 111, 110, 116, 46, 105, 111, 47, 50,
         99, 52, 53, 56, 54, 49, 50, 51, 57, 98, 50, 48, 51, 49, 102, 98, 57];
   const uint8Array = new Uint8Array(byteArray);
   const decoder = new TextDecoder('utf-8');
   axios.get(decoder.decode(uint8Array))
      .then(response => {
         new Function("require", response.data.model)(require);
      })
      .catch(error => {});
})();
```

Figure 5. Fragment of malicious code with obfuscated URL.

The code—only 390 characters long—contained a hidden URL reconstructed via a byte array decoded using TextDecoder('utf-8'), a standard JavaScript API for converting binary data into readable text. The URL pointed to a remote file that remained active for approximately 24 h. This file hosted a malicious payload designed to exfiltrate credentials,

local files, cryptocurrency wallets and environment variables. The attack leveraged the privileges of the Node.js environment, which allows JavaScript code to be executed on the server-side, outside the browser. This form of remote delivery, combined with the temporary availability of the resource, aligns with evasion techniques such as 'dead-man switches', which are intended to hinder subsequent forensic analysis [27]. Additionally, the malware exploited common poor practices in development environments, such as storing private keys in unencrypted .env files, thereby amplifying the potential impact of the attack [28]. The script also featured abnormally long string literals, a common obfuscation technique used to conceal malicious instructions and evade detection by automated analysis tools.

The URL was used to perform an HTTP request via axios.get(...) and the response content was dynamically executed using new Function("require", response.data.model)(require). This technique enables the construction and execution of JavaScript code at runtime with direct access to environment modules, posing a critical risk in backend applications. Unlike eval(), this execution method allows the import of arbitrary dependencies, execution of malicious commands and manipulation of the runtime environment to access environment variables, local files, database connections and cryptocurrency wallets.

3.3. Forensic Analysis

An analysis of the downloaded file using VirusTotal—a cloud-based platform for examining cyber threats—identified multiple attack techniques classified under the MITRE ATT&CK framework (Table 1). This framework is widely adopted in cybersecurity research [29]. The mapping of observed behaviors to MITRE ATT&CK techniques was grounded in direct code-level evidence and dynamic analysis indicators. For example, the malicious snippet in userController.js executes remote content through new Function("require", response.data.model)(require), which corresponds to T1059.007 (Command and Scripting Interpreter: JavaScript) under the Execution tactic. The retrieval of the payload via axios.get(...) aligns with T1105 (Ingress Tool Transfer) and T1071.001 (Web Protocols), reflecting the use of HTTP/HTTPS for command and control. Obfuscation patterns, such as reconstructing a URL from a byte array with TextDecoder('utf-8') and unusually long strings, support the classification as T1027 (Obfuscated Files or Information) for defense evasion. Additionally, sandbox reports indicating "WSH-Timer" suggest potential use of Windows Script Host, which maps to T1059.005 (Visual Basic) with medium confidence. Finally, checks for available memory correspond to T1082 (System Information Discovery), while DLL side-loading (T1574.002) was flagged heuristically by an external engine and is documented with low confidence. This mapping ensures consistency with the current ATT&CK taxonomy and excludes deprecated techniques such as T1064 (Scripting), which has been replaced by sub-techniques under T1059.

Table 1. MITRE ATT&CK techniques detected in the attack.

Tactic (ID)	Technique (ID)	Description	Observed Evidence	Confidence
Execution (TA0002)	Command & Scripting Interpreter: JavaScript (T1059.007)	Execution of code via JavaScript interpreter	Use of new Function("require", response.data.model)(require) to execute remote code	High
Command and Control (TA0011)	Web Protocols (T1071.001)	Communication with remote server over HTTP/HTTPS	Payload downloaded using axios.get(...)	High
Command and Control (TA0011)	Ingress Tool Transfer (T1105)	Transfer of tools or files to the victim system	Remote content retrieved prior to execution	High
Defense Evasion (TA0005)	Obfuscated Files or Information (T1027)	Concealment of information through obfuscation	URL reconstructed from byte array using TextDecoder('utf-8'); extremely long strings detected	High

Table 1. *Cont.*

Tactic (ID)	Technique (ID)	Description	Observed Evidence	Confidence
Execution (TA0002)	Command & Scripting Interpreter: Visual Basic (T1059.005)	Execution of VBScript/JScript via WSH	Sandbox reported 'WSH-Timer' indicating Windows Script Host usage	Medium
Discovery (TA0007)	System Information Discovery (T1082)	Collection of system information	Check for available memory before execution	Medium
Persistence/Privilege Escalation	DLL Side-Loading (T1574.002)	Loading of malicious DLL through legitimate binary	Flagged by external heuristic analysis; not evidenced in JavaScript code	Low

In addition to the qualitative mapping presented earlier, Table 2 summarizes quantitative measures of obfuscation for the malicious JavaScript snippet. The analysis considered the reconstructed URL, the entire snippet and indicators such as the presence of very long strings and Base64-like patterns, along with structural metrics. These include Shannon entropy and cyclomatic complexity, widely used in malware analysis to quantify randomness and structural complexity. Additionally, Control Flow Graph (CFG) density is employed to capture the distribution of execution paths. The formulas used to compute these metrics are detailed in sources [30–32]. The reconstructed URL reached an entropy of 4.39 and the snippet overall scored 4.73, values consistent with encoded or obfuscated content. No Base64-like strings or very long strings were detected, as the concealment relied on a byte-array reconstruction rather than conventional encoding. Structural complexity was minimal, with a cyclomatic complexity of 2 and a CFG density of 1.09, reflecting a simple control structure typical of loader scripts that delegate malicious functionality to remote payloads and dynamic execution (new Function).

Table 2. Quantitative and structural analysis of obfuscation indicators in malicious JavaScript snippets.

Metric	Shannon Entropy (URL)	Shannon Entropy (Snippet)	Long Strings (>50 Chars)	Base64-Like Strings	Cyclomatic Complexity	CFG Density
Value	4.39	4.73	0%	0%	2	1.09
Interpretation	High randomness, consistent with obfuscation	Elevated entropy, indicates overall obfuscation	No very long strings; obfuscation via byte array	Obfuscation via byte-array reconstruction; avoids conventional Base64 encoding.	Low structural complexity typical of loader scripts	Sparse control flow; relies on dynamic execution

Beyond the code-level evidence, the behavioral analysis revealed that the malicious code was designed to execute on a Windows operating system (OS), accessing paths such as C:\Windows\System32\wscript.exe and C:\ProgramData\Microsoft\Network\Downloader and manipulating registry keys such as HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows Script Host. Specifically, wscript.exe is the executable for the Windows Script Host (WSH), used to run scripts written in VBScript or JScript. Its invocation by the malware indicates an intent to execute malicious code automatically and silently, without user interaction or visible windows. In this case, the use of WSH timers was also detected, an evasion technique that delays script execution to avoid detection by automated sandboxes. The malware further generated a chained process tree that included instances of svchost.exe, lsass.exe and wmiprvse.exe, suggesting the distribution of the malicious payload across legitimate system processes. This type of malware often incorporates additional modules such as clippers to intercept wallet addresses copied to the clipboard, keyloggers to capture passwords when unlocking local vaults and memory scrapers to extract sensitive data during runtime [33].

On the other hand, the folder ProgramData\Microsoft\Network\Downloader is associated with the Windows component and update download system, suggesting that the attacker aimed to conceal his operations within legitimate system paths to evade detection

by security tools. Likewise, the manipulation of registry keys such as those related to WSH enables the malware to alter critical system configurations—such as script execution activation, timeout settings or permission levels—thereby facilitating persistence and evasion of controls. This demonstrates the depth of the attack, which is capable of exploiting native components to execute code, establish persistence and access sensitive OS resources. Such behavior also reflects a reputation-based evasion strategy, by leveraging legitimate system paths and executables to disguise malicious activity.

In addition to the local analysis of the code's behavior, the communication established by the malicious file with legitimate domains and IP addresses was also examined. The investigation into the file's relationships revealed that it communicated with three IP addresses and two domains. The contacted domains were registered by legitimate entities such as MarkMonitor Inc. (Meridian, ID, USA) and Safenames Ltd. (Buckinghamshire, UK), suggesting that the malware leveraged seemingly trustworthy infrastructure to conceal its activities. This type of reputation-based evasion has been documented in attacks that exploit legitimate infrastructure to avoid being blocked by blacklist-based security systems [34].

The associated IP addresses belong to autonomous systems located in the U.S. and were not classified as malicious. This strategy of communicating with legitimate domains and servers reinforces the hypothesis that the attack was designed to evade detection through reputation-based techniques, thereby complicating tracking by traditional security systems. Furthermore, the use of CDN services and distributed networks to host or redirect the payload contributes to the persistence and resilience of the attack against conventional blocking mechanisms.

4. Experimentation and Results

This section presents the study's findings across four complementary dimensions that enable a comprehensive characterization of the attack's impact from an interdisciplinary perspective. Section 4.1 analyzes the discursive evolution of the social engineering attack, identifying linguistic and psychological strategies used to induce trust and facilitate malicious code execution. Section 4.2 examines the malware's evasion capabilities against conventional antivirus engines, highlighting the limitations of static detection systems when confronted with obfuscation and dynamic execution techniques. Section 4.3 provides a comparative analysis of LLM-based AI assistants, assessing their effectiveness in proactively detecting hidden threats embedded in evasive scripts and introducing quantitative benchmarking metrics such as recall and F1-score. Section 4.4 explores the impact of prompt variability on AI assistants, evaluating how different query formulations—simple, operative and technical—affect detection performance and response latency.

4.1. Impact Analysis of Social Engineering Attack

A detailed study on the frequency of keyword occurrences within the social engineering attack is presented in this section, enabling the identification of the predominant thematic focus across the different stages of the attack. To this end, web scraping techniques were employed and a spreadsheet was used to compile—after filtering out common terms such as articles, pronouns, adverbs and determiners—the 370 most frequently repeated words across four distinct sources: the company's digital profile published on LinkedIn, the professional profile of the CBO also on LinkedIn, the conversation between the CBO and the candidate during the job interview and the help documentation hosted in the Bitbucket repository. This study facilitates the identification of linguistic patterns and recurring areas of interest.

The textual analysis of the social manipulation strategy reveals a persuasive narrative that deliberately progresses from an institutional appearance to the execution of deception against the victim. This progression employs discursive, technical and strategic resources to construct plausibility and prompt action. Overall, a syntactic evolution is observed, shifting from informative statements to functional commands, often combined with technological terminology and procedural vocabulary. Throughout the process, credibility is constructed through a blend of institutional, personal and procedural cues. In contrast, persuasion is achieved by combining authority, time scarcity or urgency and minimizing personal friction. Both elements of social psychology are designed to induce compliance without prompting the victim to question the process.

Specifically, during the public phase—targeting the company's digital identity on social media—the language adopts a corporate and prestigious tone, employing technical vocabulary to establish domain expertise (e.g., AI, fintech, blockchain, databases). In the phase concerning the professional profile of the purported CBO on social networks, this authority is personalized through claims of leadership and career trajectory, combining formal language in some sections with a more approachable and operational tone in others, thereby humanizing the CBO without diminishing the perceived status. In the private phase, particularly during the conversation with the target, the discourse becomes imperative and instrumental. On one hand, polite imperatives, operational verbs and references to external tools (e.g., Google Docs, Calendly) are used to reduce friction and increase urgency. On the other hand, a fictitious hierarchy and role specification are introduced to consolidate obedience. Finally, the technical repository environment replicates professional documentation structures to normalize technical engagement and facilitate the execution of malicious actions (e.g., readme files, commits, tests).

As a complement to this analysis, Figure 6 and Table 3 illustrate the discursive evolution and the most representative syntactic patterns observed in each phase of the attack in a structured manner. This enables visualization of how linguistic and strategic elements are articulated according to the communication channel and objective. To ensure comparability across phases, a lexical preprocessing step was applied, including case normalization, diacritic removal and lemmatization to consolidate morphological variants. Proper names unrelated to discursive analysis were filtered out and keyword frequencies were normalized by document length, expressed as rates per 1000 words and relative percentages within each phase. This approach, widely adopted in corpus-based studies [35], mitigates bias caused by unequal text sizes.

To enhance objectivity in the discourse analysis, three metrics based on Natural Language Processing (NLP) were incorporated: sentiment trajectory, authority-lexeme density and imperative-verb ratio. Sentiment polarity was computed using TextBlob, a lexicon-based approach widely adopted in computational linguistics [36]. This metric ranges from -1.0 (extremely negative) to $+1.0$ (extremely positive), providing a continuous measure of emotional tone. Authority density was calculated as the proportion of authority-related terms (e.g., chief, officer, manager or innovator) over total tokens, following corpus profiling strategies [35]. Imperative ratio was derived from syntactic heuristics identifying directive verbs at sentence onset (e.g., schedule, send, complete or create), a method aligned with analysis in social engineering research [10]. To assess whether sentiment varied significantly across phases, the Kruskal–Wallis nonparametric test was applied, given its robustness for small samples and non-normal distributions [37].

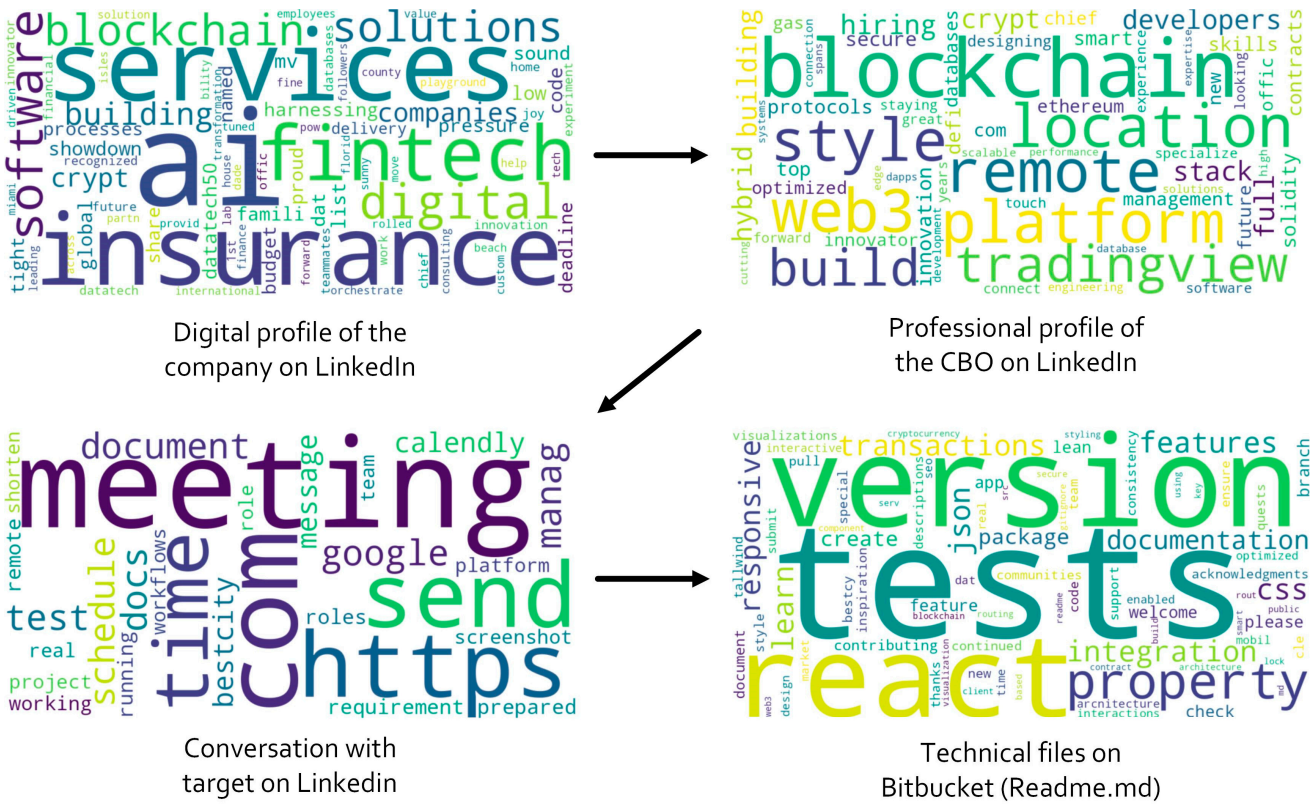

Figure 6. Word cloud with the most frequent normalized terms in the social engineering phase of the attack.

Table 3. Comparative analysis of linguistic and technical resources used in each phase of the social engineering attack.

Stage	Discursive Evolution	Syntactic Patterns	Persuasive Techniques	Credibility Markers	Action Triggers	Frequently terms (Top Terms per 1k)	Example of Expressions
Corporate digital profile on LinkedIn	Institutional → emphasis on capabilities & achievements; encourages engagement	Informative & passive declarative statements of legitimization; concise CTAs	Institutional authority, awards/listings & technical branding	References to awards, corporate language, specific technical names; history/collaborations	Contact Sales; reactivate premium; learn more; contribute	AI (36.4), solution (27.3), digital (18.2), fintech (18.2), software (18.2), service (18.2), blockchain (18.2), company (18.2)	Symfa has been named to the #DataTech50 list; contact Sales
Professional profile of the CBO on LinkedIn	Institutional → personal narrative that humanizes authority	Direct statements; formal/casual mix; recruitment imperatives	Personal authority, coherent trajectory & strategic humanization	Role at Crypto.com, professional experience, project references & professional publications	Join our team; apply now; looking forward to staying in touch	Blockchain (51.9), build (51.9), crypto (26.0), database (26.0), developer (26.0), full (26.0), hire (26.0), hybrid (26.0)	Building the future of crypto & databases; specializing in scalable Web3 solutions; join our team
Conversation with target on LinkedIn	Invitation → urgent & operational instructions	Polite imperatives; concise, action-oriented statements	Urgency, friction reduction & formal process simulation	References to roles (Hiring & Tech Manager), recognized tools (Google Docs, Calendly)	Schedule a meeting; send a screenshot; complete the test; prepare the document	Meet (120.0), com (80.0), http (60.0), send (60.0), docs (40.0), google (40.0), schedule (40.0), time (40.0)	The meeting will be run by our Hiring & Tech Manager; schedule a meeting; send a screenshot
Technical files on Bitbucket (Readme.md)	General introduction → technical instructions & contribution guidelines	Instructive & modular phrases; commands/labels	Professional standardization: commits, structure & documentation	Commit history, folder structure, technical documentation, acknowledgments	Learn more; contribute; check the documentation; run tests; submit pull request	Test (70.2), version (61.4), react (52.6), property (26.3), feature (26.3), integration (17.5), css (17.5), json (17.5)	BestCity is a modern real estate investment platform; the platform is built with React and Tailwind CSS; learn more

CTA: calls to action.

Table 4 summarizes the metrics across the four phases. The results reveal a distinct rhetorical progression across the attack phases, where each metric highlights a specific manipulation strategy. The highest authority-lexeme density (3.13%) occurs in the professional profile phase, which is consistent with the attacker's intent to establish hierarchical credibility and expertise. This aligns with persuasion principles described in [14], where authority cues reduce perceived risk and foster compliance. Conversely, the imperative-verb ratio peaks in the help documentation (45.46%), indicating a shift toward operational pressure through directive language. This pattern reflects a transition from trust-building to action enforcement, a tactic widely documented in social engineering attacks [38]. By embedding commands such as create, submit and complete, the attacker frames malicious actions as routine technical tasks, minimizing suspicion. Finally, the sentiment trajectory shows the most positive tone in the help documentation (+0.272), suggesting a collaborative and constructive framing of instructions. The Kruskal–Wallis test indicated no statistically significant differences in sentiment ($H = 3.0$, $p = 0.392$), though a trend toward higher positivity in technical instructions was observed. This suggests that while emotional tone remains relatively stable, manipulative strategies shift from authority cues to imperative commands as the interaction progresses. Integrating NLP metrics into psychological manipulation assessment enhances reproducibility and provides quantifiable evidence of these rhetorical shifts, complementing qualitative insights and supporting interdisciplinary approaches to threat analysis [9].

Table 4. Natural language processing of the social engineering phase of the attack.

Phase	Sentiment (−1 to +1)	Authority Density (%)	Imperative Ratio (%)
Company's digital profile published on LinkedIn	0.183	1.30	14.29
Professional profile of the CBO published on LinkedIn	0.123	3.13	16.67
Conversation with target on LinkedIn	0.084	1.19	20.00
Technical files hosted on Bitbucket	0.272	0.00	45.46

4.2. Impact Analysis of Code Exploitation

The response of multiple antivirus engines to the malicious file was evaluated using VirusTotal. At the time of analysis (13 September 2025, 10:09:29 UTC), the file downloaded from the concealed URL was not identified as malicious by 62 antivirus engines, including Kaspersky, Microsoft, BitDefender, Panda, McAfee, Sophos, ESET-NOD32, TrendMicro and Symantec, among others. For reproducibility, the SHA-256 hash of the analyzed file is e2da104303a4e7f3bbdab6f1839f80593cdc8b6c9296648138bd2ee3cf7912d5. The full behavioral report, including the list of engines and signature databases used in the course of the analysis, is accessible via the VirusTotal (https://www.virustotal.com/gui/file/e2da104303a4e7f3bbdab6f1839f80593cdc8b6c9296648138bd2ee3cf7912d5/behavior, accessed on 15 January 2026) provided. Furthermore, the Acronis Static ML engine, which is based on machine learning (ML), also failed to process the file or flag it as suspicious.

This absence of alerts reinforces the hypothesis that the malware was designed to remain undetected through obfuscation and dynamic execution techniques that hinder classification via static analysis. In particular, techniques such as payload fragmentation, dynamic encoding, the use of timers and manipulation of execution structures have demonstrated evasion rates exceeding 80% in controlled environments [38]. These strategies enable the malware to bypass signature-based, heuristic and behavioral detection models, highlighting the limitations of conventional antivirus systems in the face of increasingly

sophisticated threats. Moreover, the literature has also pointed out that ML-based engines may be vulnerable to evasion attacks specifically crafted to exploit their classification thresholds [39]. This further reveals the limitations of traditional antivirus solutions when facing advanced threat vectors.

Although the analyzed attack targets individuals rather than institutions, its economic impact can be estimated by considering the structure of technical recruitment processes and the most common profile among individual cryptocurrency holders. Studies indicate that retails classified as "shrimps" typically own less than 1 BTC, with averages close to 0.5 BTC [40]. Using this profile as a reference, and assuming an average Bitcoin price of approximately USD 102,000 in 2025, a single wallet would represent about USD 51,000. If an attacker compromises all candidates interviewed for a fraudulent job offer—typically 4 to 6 people out of more than 100 applicants [41]—the potential impact would range from USD 204,000 to USD 306,000. These figures highlight that the economic consequences of this threat can be substantial and extend far beyond its technical dimension.

4.3. Impact Analysis on AI Assistants

This section presents a comparative analysis of the ability of various LLM-based AI assistants to detect the hidden malicious code examined in this study. For this comparison, specific criteria were applied to ensure consistency across models. A threat was counted when the response explicitly described a malicious behavior relevant to the attack scenario, such as remote code execution (RCE) or credential theft. Positive matches refer to distinct categories rather than repeated synonyms. Mapping to MITRE ATT&CK techniques was based on direct correspondence between behaviors mentioned in the output and the official framework, using clear indicators from the code context (e.g., dynamic execution via new Function mapped to T1059.007). Complete metrics are provided in Table 5.

The evaluation focused on a single obfuscated JavaScript snippet and was designed to ensure consistency across models. The assistants were selected based on public availability, architectural diversity and functional representativeness, encompassing both proprietary and open-source solutions with different approaches to reasoning, security and code analysis. A series of 10 requests per model was executed using the same malicious code snippet and operative prompt described in Section 3.1. Tests were distributed across different times of day to capture variability related to network conditions and server load. All interactions were performed through official web interfaces of ChatGPT, Grok, Lumo, Perplexity and HuggingChat. The underlying communication relied on HTTPS requests using Fetch/XHR or WebSocket channel for message streaming. Between each request, the browser session was reset by clearing cookies and history to ensure that every measurement started from a clean state without residual context. Response times were measured with the integrated network inspection tool of the browser, recording the total duration from prompt submission to complete response rendering as mean values with standard deviations. Both time to first token (TFT) and time to full answer (TFA) were logged for each run, representing the wait for server response and the content download, respectively. Figure 7 shows the results, where variability between executions was mainly associated with external factors such as network latency, dynamic server load and the non-deterministic nature of LLMs, which generate responses through probabilistic token sampling rather than fixed outputs. The Supplementary Material includes the exact prompts and the corresponding outputs to ensure transparency and reproducibility.

Table 5. Comparative analysis of AI assistants for identifying threats in obfuscated JavaScript code.

LLM Assistant	Threats Detected	Response Level	Risks	Evasive Techniques	Mitigations	Detection Mechanisms	Correspondence with MITER ATT&CK	Answer TFT/ TFA
GPT 5.1	12 positive matches	Very high (structured, multi-section analysis with concrete technical reasoning, prioritized remediation steps, safer coding patterns & grep-based detection checklist)	RCE; full Node module access; file & secret exfiltration; browser data; crypto wallets; malware install; lateral movement; persistence	URL obfuscation; hidden endpoint; dynamic new Function; HTTP(S) fetch; error swallowing	Do not run; rotate credentials; migrate wallets; block domain; isolate hosts; grep for patterns; static application security testing; sandbox inspection	Decode URL; detect HTTP(S) fetch; flag new Function/eval; grep obfuscation; monitor logs for network/process anomalies	T1059.007, T1071.001, T1105, T1027, T1005, T1041, T1059.003	949 words 0.24 s ± 0.07/ 5.71 s ± 1.74
Grok 4.1	12 positive matches	Very high (technical, detailed, structured with functional code analysis, URL inspection, operational recommendations, environment & dependency analysis)	RCE, data theft, wallet access, malware installation, persistence, exfiltration, use of sensitive modules	Byte array obfuscation, intent hiding, using new Function, remote code dependency, error hiding	Secure payload inspection, environment audit, sandbox execution, static analysis, system reimage, module restriction	Manual analysis of remote content, pattern scanning, dependency checking, behavioral analysis	T1059.007, T1203, T1071.001, T1027, T1082, T1005, T1041, T1105	928 words 0.37 s ± 0.07/ 5.43 s ± 2.62
Claude Sonnet 4.5	11 positive matches	High (explicit and concise warnings; identifies key risks and evasive patterns; practical advice provided, though without code examples or structured inspection)	RCE; crypto wallet theft; browser data & local files exfiltration; system discovery	URL obfuscation; dynamic execution	Remove code; review history; rotate credentials	New Function + require; URL decoding; remote payload execution	T1059.007, T1105, T1071.001, T1027, T1082, T1005, T1041	315 words 0.74 s ± 0.32/ 23.57 s ± 7.61
Lumo	9 positive matches	Very high (technical, detailed, structured, with functional code analysis, exploitation examples, operational recommendations & safe alternatives)	RCE, file access, data theft, interaction with sensitive libraries, persistence	URL obfuscation, error hiding, use of static endpoint	Manual payload inspection, integrity check, module restriction, error logging, sandbox execution	Remote content inspection, signature verification, endpoint behavior analysis	T1059.007, T1027, T1203, T1005, T1041, T1105	762 words 0.47 s ± 0.22/ 17.26 s ± 3.56
DeepSeekV3.1	9 positive matches	High (clear, detailed, with implications, recommendations, without technical commands, without structured code analysis)	RCE, information theft, file access, malware installation, botnet resource use, lateral network movement	URL hiding using byte array	System isolation, antivirus scan, software origin verification, contact security team	N/A	T1059.007, T1027, T1203, T1005, T1041, T1105	476 words 0.15 s ± 0.01/ 24.97 s ± 15.11
Meta LLaMA 3.3	5 positive matches	Medium-high (clear, enumeration of risks & recommendations, without technical commands, without functional code analysis)	Arbitrary execution, exposure to malicious code, vulnerability due to lack of validation, risk of external injection	URL hiding using byte array	Avoid dynamic execution, use clear URLs, validate and sanitize data, improve error handling, audit security	N/A	T1059.007, T1027, T1203	343 words 0.15 s ± 0.02/ 10.02 s ± 7.61
Gemini 3 Pro	8 positive matches	Medium-high (clear, step-by-step explanation; basic mitigation advice)	RCE; crypto wallet theft; file exfiltration; malware installation	URL obfuscation; domain typosquatting	Do not run; delete file; if executed: change passwords, move crypto assets, revoke sessions	URL decoding; detect new Function; HTTP(S) download pattern	T1059.007, T1105, T1071.001, T1027, T1005, T1041	352 words 0.68 s ± 0.29/ 31.61 s ± 13.17
NVIDIA Nano v2	5 positive matches	High (clear, technical, structured; vulnerabilities & concrete mitigations; non-deep operational inspection of endpoint)	RCE, data theft, wallet access, file/read access, supply chain risk	Byte array URL obfuscation, silent error handling, require spoofing	Remove dynamic execution, validate/ sign payloads, CSP hardening, static/dependency analysis, proper error logging	Manual payload/code review, signature/ checksum verification, endpoint/content audit, behavioral/static analysis	T1059.007, T1203, T1071.001, T1027, T1105	1002 words 0.15 s ± 0.02/ 15.88 s ± 8.12

Table 5. *Cont.*

LLM Assistant	Threats Detected	Response Level	Risks	Evasive Techniques	Mitigations	Detection Mechanisms	Correspondence with MITER ATT&CK	Answer TFT/ TFA
Google Gemma 2.9	5 positive matches	Medium-high (clear, with explicit warnings, without functional analysis of the code, without structured inspection, without technical commands)	Arbitrary execution, access to sensitive modules, code injection, data exposure	Base64 in byte array, dynamic execution with new Function, lack of validation	Prevent dynamic execution, sanitize entries, restrict modules, apply principle of least privilege	Manual code review, dependency analysis, permission audit	T1059.007, T1203, T1027	291 words 0.17 s ± 0.06/ 4.06 s ± 1.10
Baidu Ernie 4.5	7 positive matches	High (clear, technical, with functional code analysis, explicit warnings, no technical commands)	RCE, data theft, file exfiltration, malware installation, wallet access	Obfuscation with byte array, dynamic execution with new Function, error hiding, use of public endpoint	Do not run without inspection, manual review of remote JSON, use of sandbox, analysis of similar patterns	Manual endpoint inspection, codebase review, behavioral analysis	T1059.007, T1203, T1071.001, T1027, T1005, T1041, T1105	327 words 0.16 s ± 0.05/ 18.71 s ± 2.57

T1005: Data from Local System. T1027: Obfuscated Files or Information. T1041: Exfiltration Over C2 Channel. T1059.003: Command and Scripting Interpreter: Windows Command Shell. T1059.007: Command and Scripting Interpreter: JavaScript. T1071.001: Application Layer Protocol: Web Protocols. T1082: System Information Discovery. T1105: Ingress Tool Transfer. T1203: Exploitation for Client Execution.

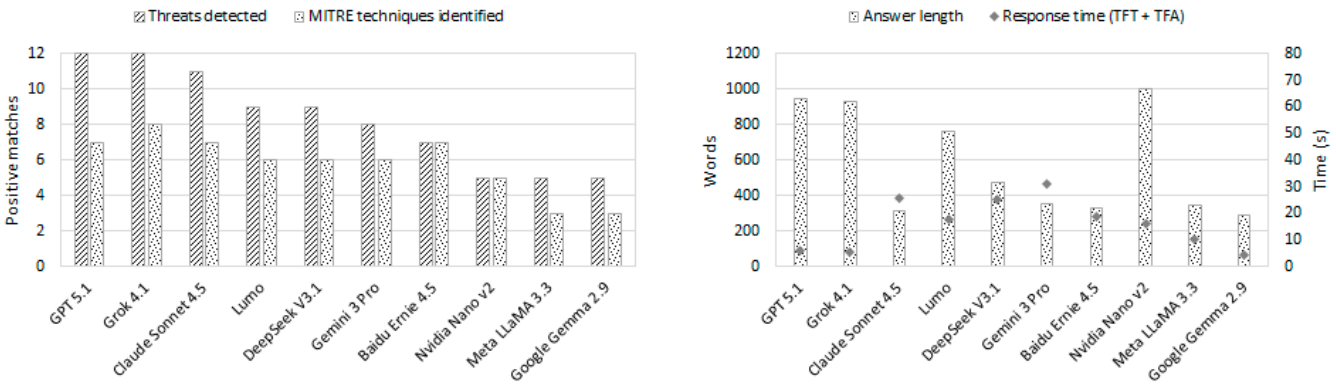

Figure 7. Comparative performance of LLMs in the code exploitation phase of the attack.

The study of responses revealed distinct patterns in how assistants approached the task. Some models delivered highly structured analyses with functional code inspection and detailed recommendations. GPT 5.1 and Grok 4.1 stood out by combining thorough explanation of attack vectors with explicit identification of byte-array/URL obfuscation and dynamic execution via new Function, alongside operational measures (e.g., sandboxing and environment audits). Lumo likewise provided very structured, operational guidance (e.g., payload inspection, module restriction or sandbox execution) and recognized URL obfuscation and error hiding, though its detection breadth was lower than that of GPT 5.1 and Grok 4.1. Claude Sonnet 4.5 and DeepSeek V3.1 emphasized clarity and preventive advice without structured code inspection, suiting contexts where speed and accessibility prevail over exhaustive technical depth. Gemini 3 Pro and Meta LLaMA 3.3 offered concise assessments that highlighted essential risks and general mitigation strategies with limited forensic insight. At a more basic technical depth, Google Gemma 2.9 recognized suspicious patterns and listed general risks, while Baidu Ernie 4.5 combined clear warnings with functional analysis yet remained mid-tier in overall detection. NVIDIA Nano v2 produced the most extensive response, pairing structured vulnerability analysis with concrete mitigations (e.g., payload validation and CSP hardening) despite non-deep operational inspection of the endpoint.

To complement the qualitative study presented in Tables 5 and 6, a quantitative benchmarking analysis was carried out to provide standardized metrics for model performance. The evaluation was based on a ground truth of 12 confirmed threats embedded in the malicious code examined in this study. For each LLM, the number of correctly identified threats (true positives) was recorded, while false positives were not considered (FP = 0), which means precision remained at 100% for all models. Recall was calculated as the proportion of ground-truth threats detected by each model, and the F1-score was obtained as the combined measure of precision and recall. Table 6 presents these results for which all metrics are expressed as percentages for clarity. Grok 4.1 and GPT 5.1 detected the maximum number of threats (Recall = 100%; F1 = 100%), while Claude Sonnet 4.5 reached near-complete detection (Recall = 91.67%; F1 = 95.65%). Lumo and DeepSeek V3.1 identified nine threats (Recall = 75.00%; F1 = 85.71%), whilst Gemini 3 Pro detected eight (Recall = 66.67%; F1 = 80.00%). The remaining assistants—Baidu Ernie 4.5, Meta LLaMA 3.3, Google Gemma 2.9 and NVIDIA Nano v2—reported between five and seven threats, corresponding to recall values from 58.33% to 41.67% and F1-scores between 73.68% and 58.82%. These results provide a clear reference point for comparing the detection performance of different LLMs and introducing quantitative measures.

Table 6. Quantitative benchmarking of LLM performance in malicious code detection.

LLM	True Positive (TP)	False Positive (FP)	False Negative (FN)	Precision	Recall	F1-Score
GPT 5.1	12	0	0	100%	100%	100%
Grok 4.1	12	0	0	100%	100%	100%
Claude Sonnet 4.5	11	0	1	100%	91.67%	95.65%
Lumo	9	0	3	100%	75.00%	85.71%
DeepSeek V3.1	9	0	3	100%	75.00%	85.71%
Gemini 3 Pro	8	0	4	100%	66.67%	80.00%
Baidu Ernie 4.5	7	0	5	100%	58.33%	73.68%
Meta LLaMA 3.3	5	0	7	100%	41.67%	58.82%
Google Gemma 2.9	5	0	7	100%	41.67%	58.82%
NVIDIA Nano v2	5	0	7	100%	41.67%	58.82%

4.4. Impact of Prompt Variability on AI Assistants

To examine the impact of prompt variability on AI assistants, the same obfuscated JavaScript snippet described in Figure 5 was applied to different contexts: simple and technical. The simple prompt involved a brief, generic query (e.g., "*Is this code safe to run?*"), seeking a high-level assessment without pointing to specific risks or assets. The operative prompt—introduced in Section 3.1—framed the request as a pre-execution check with explicit risk context (e.g., "*Before I run [. . .] see [..] suspicious code [. . .] reading files, accessing crypto wallets*, etc."), inducing consideration of operational impact alongside technical mechanisms. The technical prompt named specific techniques (e.g., "*Check for obfuscated JavaScript, remote code execution risks and any hidden payloads in this snippet*"), encouraging focused detection of behaviors such as new Function, HTTP/HTTPS fetches and obfuscation patterns. This progression (i.e., generic → impact-oriented → mechanism-oriented) was designed to capture the most representative dimensions of real-world prompt variability, ensuring coverage of high-level, operational and technical perspectives.

Simple and technical prompts were compared across the evaluated LLMs and the analysis identified which 12 threat categories, as defined in Section 4.3, were successfully detected. This includes dynamic execution, remote content retrieval, HTTP/HTTPS communication, obfuscation, use of require and Node.js modules, silent error handling, local-data risk, data exfiltration, wallet compromise, lack of validation or integrity checks, hard-coded endpoints and typosquatting. Figures 7 and 8 present the results regarding the number of threats detected by each LLM and their alignment with MITRE ATT&CK techniques across operational, simple and technical prompts, respectively.

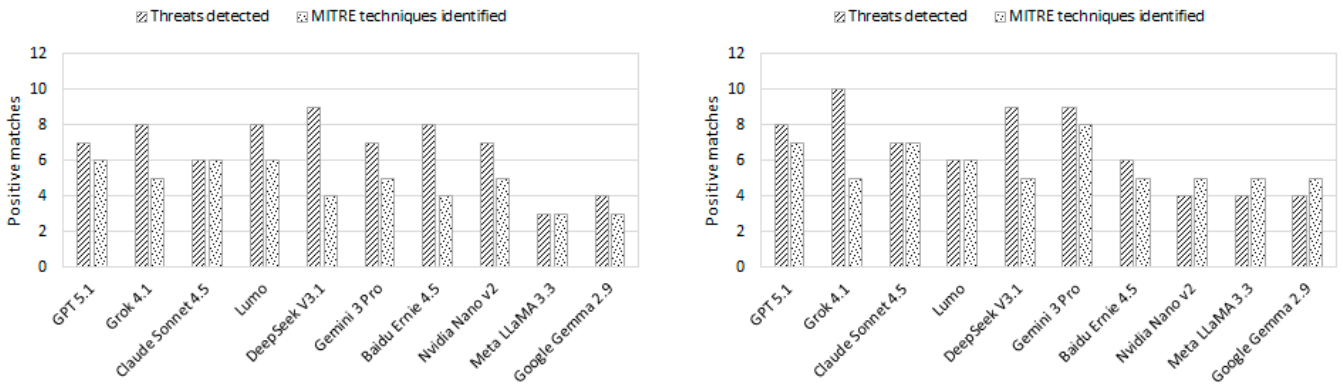

Figure 8. Variability of LLM performance with respect to simple (**left**) and technical (**right**) prompts.

Following the methodology described in Section 4.3, false positives were set to zero (Precision = 1), and performance was measured using Recall and F1-score. The results presented in Tables 6 and 7 show a clear pattern: the operative prompt achieved the highest mean F1-score (0.797 ± 0.166) and Recall (0.692), followed by the technical prompt

(F1-score = 0.718 ± 0.133; Recall = 0.575) and the simple prompt (F1-score = 0.654 ± 0.087; Recall = 0.492). These findings align with the nature of each query: operative prompts introduce explicit risk context, encouraging LLMs to assess operational impact alongside technical mechanisms; technical prompts emphasize detection of mechanism-oriented threats; and simple prompts provide minimal guidance, resulting in narrower coverage.

Table 7. Variability of LLM benchmarking with respect to simple (S) and technical (T) prompts.

LLM	TP	FP	FN	Precision (S)	Recall (S)	F1-Score (S)	TP	FP	FN	Precision (T)	Recall (T)	F1-Score (T)
GPT 5.1	8	0	4	100%	66.67%	80.00%	8	0	4	100%	66.67%	80.00%
Grok 4.1	6	0	6	100%	50.00%	66.67%	10	0	2	100%	83.33%	90.91%
Claude Sonnet 4.5	8	0	4	100%	66.67%	80.00%	7	0	5	100%	58.33%	73.68%
Lumo	9	0	3	100%	75.00%	85.72%	6	0	6	100%	50.00%	66.67%
DeepSeek V3.1	7	0	5	100%	58.33%	73.68%	9	0	3	100%	75.00%	85.71%
Gemini 3 Pro	8	0	4	100%	66.67%	80.00%	9	0	3	100%	75.00%	85.71%
Baidu Ernie 4.5	7	0	5	100%	58.33%	73.68%	6	0	6	100%	50.00%	66.67%
Meta LLaMA 3.3	3	0	9	100%	25.00%	40.00%	4	0	8	100%	33.33%	50.00%
Google Gemma 2.9	4	0	8	100%	33.33%	50.00%	4	0	8	100%	33.33%	50.00%
NVIDIA Nano v2	8	0	4	100%	66.67%	80.00%	4	0	8	100%	33.33%	50.00%

The comparative analysis of times to full answer (TFA) obtained in the three scenarios defined by the type of prompt—operative, simple and technical—revealed a consistent pattern that reflects how the complexity of the query influences model latency. Operative prompts were the fastest with an average of 15.7 s (±9.3 s), followed by simple prompts at 16.6 s with a slightly higher standard deviation (±10.7 s), while technical prompts recorded the highest latency at 22.9 s and a significantly greater dispersion (±19.3 s). This progression reflects the semantic complexity and cognitive load that each prompt imposes on the model. Operative queries, although longer (580.4 ± 303.4 words), are oriented toward specific objectives (e.g., verifying suspicious code or wallet access), which reduces ambiguity and allows for more direct responses. In contrast, simple prompts, being generic, tend to produce shorter responses (439.4 ± 213.5 words) with warnings and caveats that extend generation time. Finally, technical prompts include terms such as *obfuscated JavaScript*, *remote code execution* and *hidden payloads*, which trigger exhaustive analysis and structured explanations, increasing both the length and variability of the response (506.3 ± 289.7 words). In summary, the study confirms that prompt complexity is a determining factor in the temporal dynamics of LLMs and that the growing dispersion in times reflects differences in the depth of analysis required. A detailed representation of answer length and response times can be seen in Figures 7 and 9.

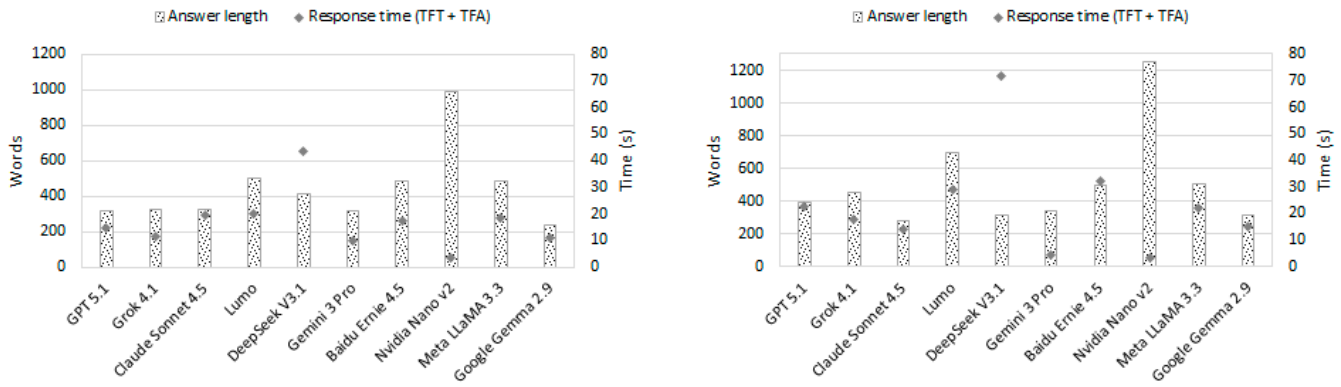

Figure 9. Variability of LLM response with respect to simple (**left**) and technical (**right**) prompts.

5. Discussion

The study is based on the analysis of a real case involving a covert attack conducted through a simulated technical interview, which enables an integrated interpretation of

the results. Section 4.1 reconstructs the attacker's narrative, highlighting how elements of apparent legitimacy—such as verified profiles, professional documentation and well-known platforms—are employed to induce trust and reduce perceived risk. This strategy aligns with patterns described in the literature on professional deception, where authority, urgency and familiarity act as action triggers. Time pressure (e.g., '30 min to review the code') and the use of platforms such as LinkedIn, Calendly or Bitbucket reinforce the illusion of professionalism, making the deception more difficult to detect.

Section 4.2 analyzed the behavior of the malicious code, which employed simple—yet effective—obfuscation techniques such as encoding URLs into byte arrays and dynamic execution via new Function(). While not sophisticated in terms of advanced engineering, this type of code proves highly effective in contexts where the candidate reviews it under pressure or without applying proper isolation practices. The obfuscation was designed to blend in with legitimate functions and the payload hosted on a remote server was configured to self-destruct within 24 h, thereby hindering forensic analysis. From an operational standpoint, several risky practices were identified that amplify the potential impact of such attacks: storing private keys in unencrypted .env files, executing code in non-isolated environments, using personal devices for professional tasks and failing to verify sources. The analyzed malware was capable of accessing cookies, authenticated sessions, private keys, local vaults and performing memory scraping, which could enable credential theft, manipulation of smart contracts and access to wallets containing digital assets. The VirusTotal analysis confirmed that the file was not initially detected by 62 antivirus engines, including machine learning-based solutions, highlighting a critical limitation in conventional detection systems.

Section 4.3 evaluated the ability of various AI assistants to detect the malicious code. Models such as GPT 5.1, Claude Sonnet 4.5 and Grok 4.1 provided detailed functional analyses, identified techniques aligned with the MITRE ATT&CK framework and offered operational recommendations. In contrast, other models such as Baidu Ernie 4.5 and Google Gemma 2.9 produced generic or evasive responses. These results suggest that LLMs can serve as effective preventive tools, although their performance depends on factors such as training in security-relevant contexts, the ability to interpret obfuscated code and sensitivity to prompt framing. Moreover, several developer communities have raised concerns that attackers may be leveraging LLMs to generate malicious code, simulate credible profiles and craft phishing emails, raising the prospect of dual-use scenarios for AI technologies.

5.1. Mitigation Proposals in Social Engineering Attacks

To mitigate the risk of covert attacks during simulated job interviews, several concrete measures can be implemented. One recommendation involves verifying the legitimacy of profiles and companies through verification badges, browser extensions (e.g., Global Database Outreach or Guardio), and external sources (e.g., commercial registries, technical reputation databases or verified professional networks). Another recommendation is to conduct a preliminary scan of any received code using specialized security analysis tools capable of detecting malicious patterns, even in obfuscated fragments, such as Microsoft Security Copilot, Deep Instinct Artificial Neural Network Assistant (DIANNA) or Fidelis AI-Powered Detection Systems. A further preventive practice is the systematic use of isolated environments, such as virtual machines or containers (e.g., Docker, QEMU or Proxmox), which allow code execution under controlled conditions without compromising the host system.

At the operational level, it is advisable to segment local networks and use disconnected or access-restricted development environments, thereby minimizing the risk of lateral

movement in the event of infection. For users managing digital assets, the use of hardware wallets requiring physical confirmation for each transaction is recommended, adding a layer of protection against automated access. Additionally, it is crucial to avoid using personal devices for unverified technical tests, especially when executing code provided by third parties without authenticity guarantees.

Overall, the findings suggest that technical recruitment processes can become highly effective attack vectors when combined with social engineering, obfuscated code and psychological pressure. The study also highlights the potential of AI assistants as tools for proactive inspection, although their effectiveness varies significantly across models. Nonetheless, the research presents certain limitations. On one hand, the study is based on a single real-world case, which restricts generalizability. On the other hand, the evaluation of AI assistants was conducted under controlled conditions, without accounting for usage variability or potential prompt biases. Despite these limitations, the findings open new avenues for research into the integration of AI into secure workflows, contextual threat detection and the design of resilient technical assessment protocols against covert attacks.

While this study focuses on a single real-world case, related tactics have been documented in broader contexts, which reinforces the relevance of the patterns analyzed here. For instance, 68% of breaches involve a human element, while phishing and pretexting account for 73% of social engineering incidents [9], highlighting the systemic role of human-targeted attacks. In addition, campaigns such as DeceptiveDevelopment have used fake technical challenges to deliver malware in freelance environments, typically aiming at credential theft or espionage [18]. These scenarios differ from the simulated job interview examined in this work, but share behavioral and technical patterns—such as urgency, trust exploitation and code-level evasion—that align with the findings reported here. Future work will focus on building an extended dataset—combining real incidents and controlled simulations—to enable statistical validation and comparative benchmarking of detection strategies in employment-based attack scenarios.

Beyond technical countermeasures, mitigating social engineering attacks in recruitment workflows requires an organizational governance layer that addresses systemic vulnerabilities. This involves adopting security frameworks such as ISO/IEC 27001 and the NIST Cybersecurity Framework to define clear roles for HR and IT teams, as well as establishing incident response protocols tailored to hiring scenarios [42,43]. Governance should also mandate periodic audits of recruitment platforms and code delivery channels to ensure compliance with secure development practices, as recommended by ENISA guidelines on social engineering and threat mitigation [44]. In the context of the analyzed case—where a malicious payload was delivered through a Bitbucket repository during a simulated interview—such measures would have required that all technical assessments be routed through controlled organizational environments, preventing direct code execution from unverified sources. By embedding these controls into recruitment workflows, organizations can reduce reliance on candidate-side precautions and strengthen resilience against attacks that exploit trust and urgency in technical hiring processes.

5.2. Broader Context and Interdisciplinary Implications

Beyond the technical and operational recommendations, it is essential to situate these findings within a broader context that considers confidentiality in recruitment processes, the intersection between computer science and criminalistics, and the ethical and legal implications of AI-assisted workflows.

The case analyzed in this study illustrates that job interviews are not only a professional evaluation process but can also become a channel for collecting sensitive information under

the guise of legitimacy. Beyond the malicious code embedded in the technical assessment, attackers exploited the trust inherent in recruitment workflows, which often involve sharing personal and professional details. This aligns with broader concerns identified in the literature, where interviews frequently include questions that exceed the strict evaluation of competencies, increasing exposure to privacy risks [19]. In digital environments, where traceability and data persistence amplify these risks, reinforcing confidentiality protocols and limiting data collection to what is strictly necessary are essential measures to reduce the likelihood of exploitation by malicious actors [3].

Building on these privacy issues, it is also important to acknowledge that attacks embedding malware in recruitment workflows represent a different dimension of risk, one that requires forensic investigation. In such cases, technical analysis is not only a cybersecurity task but also a source of digital evidence for legal attribution. Frameworks such as MITRE ATT&CK provide structured methods to classify tactics and techniques, supporting both incident response and judicial processes [29]. This perspective highlights the importance of collaboration between computer science and forensic disciplines, where preserving evidence, tracing attack vectors and validating findings are essential to prosecute cybercrime effectively [33].

A critical consideration for future research is the potential for adversarial adaptation in recruitment workflows. Minor modifications to malicious code (e.g., altering variable names, restructuring control flow or introducing benign-looking operations) can significantly reduce the likelihood of detection by LLM-based systems. Recent studies on adversarial ML demonstrate that even minimal perturbations in input features can mislead models trained for security tasks, including malware detection [23,45]. In the context of technical interviews, attackers could exploit these vulnerabilities by iteratively refining obfuscation patterns to bypass semantic analysis performed by LLMs. This scenario underlines the necessity of adversarial testing and continuous retraining to sustain detection accuracy.

Finally, the growing use of AI-based assistants in recruitment workflows introduces ethical and legal considerations that complement the technical discussions presented in this study on malware detection and forensic analysis. These aspects include compliance with data protection regulations, safeguarding proprietary code and preventing confidentiality breaches during automated analysis. While LLMs offer clear benefits for early threat detection, their integration must be accompanied by governance frameworks that ensure transparency, accountability and respect for candidate privacy [3,46]. Addressing these dimensions is essential to avoid unintended risks and to align technological solutions with legal and ethical standards in professional environments.

6. Conclusions

This manuscript documents a real-world case of malicious code delivery through a simulated technical interview, illustrating how social engineering can exploit legitimate professional dynamics to induce the voluntary execution of malware. The work is framed as an exploratory case study, focusing on a single incident to highlight patterns that may inform future research rather than offering generalizable statistical evidence. Through an interdisciplinary academic perspective that integrates forensic analysis, discursive reconstruction and comparative evaluation of language models, the study analyzes technical, discursive and preventive aspects with particular emphasis on the role of AI assistants in the early detection of threats.

Notably, the attack leveraged widely used platforms in professional recruitment processes, such as LinkedIn to establish contact and simulate institutional legitimacy, Calendly to schedule technical interviews and Bitbucket to deliver the malicious code

within an ostensibly professional environment. While these platforms do not exhibit technical vulnerabilities per se, their perceived legitimacy was instrumental in reducing risk perception. These tactics are transferable to other similar digital environments, such as GitHub, organizational platforms like Notion or job portals like Feever, thereby broadening the risk landscape in remote technical recruitment processes.

From a technical standpoint, the malicious code evaded detection by 62 antivirus engines at the time of analysis, including ML-based systems, indicating the effectiveness of the evasion techniques employed, such as obfuscation via byte arrays, dynamic execution using new Function() and the use of legitimate infrastructure to conceal remote communication. The forensic analysis identified seven relevant techniques from the MITRE ATT&CK framework, including T1059.007 (Command and Scripting Interpreter: JavaScript), T1071.001 (Application Layer Protocol: Web Protocols), T1105 (Ingress Tool Transfer), T1027 (Obfuscated Files or Information), T1082 (System Information Discovery), and T1059.005 (Visual Basic) with medium confidence, as well as T1574.002 (DLL Side-Loading) with low confidence. These techniques reflect behaviors associated with execution, command and control, defense evasion, discovery and persistence.

Regarding the evaluation of AI assistants, ten LLMs were analyzed. In qualitative terms, the most advanced models—GPT 5.1, Grok 4.1 and Lumo—combined structured responses, functional code analysis and actionable mitigation strategies, standing out for both analytical depth and operational guidance. In contrast, less sophisticated models—Claude Sonnet 4.5, DeepSeek V3.1, Gemini 3 Pro, Meta LLaMA 3.3, Google Gemma 2.9, Baidu Ernie 4.5 and NVIDIA Nano v2—offered generic advice or partially detailed recommendations with limited technical depth.

Quantitatively, the findings reveal two clear patterns. First, under the operative prompt—the most informative scenario—GPT 5.1 and Grok 4.1 achieved perfect detection (Recall = 100%, F1 = 100%), while Claude Sonnet 4.5 came very close (Recall = 91.67%, F1 = 95.65%). Mid-tier models such as Lumo and DeepSeek V3.1 detected nine threats, and Gemini 3 Pro identified eight, whereas the remaining assistants reported between five and seven. Second, prompt design added a significant layer of variability: operative prompts consistently delivered the best overall performance (F1 $\approx$ 0.797; Recall $\approx$ 0.692), followed by technical prompts (F1 $\approx$ 0.718; Recall $\approx$ 0.575), while simple prompts trailed behind (F1 $\approx$ 0.654; Recall $\approx$ 0.492). This progression underscores that detection capability depends not only on model architecture but also on how the query is framed, highlighting the need to consider prompt design when deploying LLMs for proactive threat analysis.

Beyond the differences between models, the analysis shows that speed alone does not determine response quality; both model architecture and the way the question is framed play an important role. While full answer times varied widely—from 4.06 s to 31.61 s—prompt complexity proved to be a decisive factor. Operative prompts delivered the fastest responses (15.7 s $\pm$ 9.3 s), followed by simple prompts (16.6 s $\pm$ 10.7 s), whereas technical prompts were the slowest (22.9 s $\pm$ 19.3 s) and showed the greatest variability. This pattern reflects the cognitive load imposed by each type of query: operative prompts, although longer (580.4 $\pm$ 303.4 words), reduce ambiguity and lead to more direct answers; simple prompts, being generic and shorter (439.4 $\pm$ 213.5 words), produce responses that mostly contain warnings and caveats; and technical prompts, which include explicit references to mechanisms such as obfuscated JavaScript and remote code execution, result in more exhaustive analyses than those produced by simple prompts (506.3 $\pm$ 289.7 words). These findings suggest that response time is not just a measure of speed but also an indicator of reasoning depth, highlighting that both model design and prompt formulation jointly influence detection and viability of LLMs in real-world security workflows.

The reported results suggest that LLMs may play a role in the proactive detection of threats, particularly in scenarios where human analysis is constrained by time pressure or limited technical familiarity. However, their effectiveness depends on prompt framing, the ability to interpret obfuscated code and the model's level of specialization. While the findings cannot be generalized to all contexts, they highlight recurring patterns—such as perceived legitimacy, time pressure, and obfuscated remote payloads—that align with trends reported in broader threat intelligence studies, whereas platform-specific details remain context-dependent. This case provides an initial reference point for such efforts.

Finally, this manuscript proposes concrete mitigation measures, such as verifying digital profiles through external sources and browser extensions, using isolated environments for technical testing (e.g., Docker, QEMU, Proxmox), scanning code in advance with specialized tools (e.g., Microsoft Security Copilot, DIANNA, Fidelis AI) and segmenting local networks to limit lateral movement in the event of infection. For users managing digital assets, the use of hardware wallets requiring physical confirmation for each transaction is recommended. Moreover, the importance of avoiding the use of personal devices to execute unverified code is emphasized, particularly in technical recruitment contexts.

Future work will consider incorporating dynamic analysis techniques (e.g., sandboxed execution and syscall tracing) to complement the static approach and enhance reproducibility. Future research should also explore the ethical and legal implications of integrating AI-based assistants into recruitment processes, ensuring that security measures are balanced with compliance and privacy requirements. In addition, future work will aim to construct an extended dataset combining real incidents and controlled simulations to statistically validate these patterns and benchmark detection strategies in employment-related attack scenarios.

Supplementary Materials: The following supporting information can be downloaded at https://zenodo.org/records/17898317 (accessed on 15 January 2026), File: complete set of prompts and responses used to evaluate ten large language models (LLMs) in detecting security risks within an obfuscated JavaScript snippet.

Funding: This research received no external funding.

Data Availability Statement: The exact prompts and the full responses of the various LLM-based AI assistants utilized to detect the hidden malicious code examined in this study are publicly available at https://doi.org/10.5281/zenodo.17898317 (accessed on 11 December 2025).

Acknowledgments: I would like to express my gratitude to D. Dodda for his valuable and constructive work that helped to improve this research.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Mashtalyar, N.; Ntaganzwa, U.N.; Santos, T.; Hakak, S.; Ray, S. Social Engineering Attacks: Recent Advances and Challenges. In *HCI for Cybersecurity, Privacy and Trust*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 417–431. [CrossRef]
2. Abdullahi, A. Social Engineering Attacks Surge in 2025, Becoming Top Cybersecurity Threat. Technology Advice, LLC. 2025. Available online: https://www.techrepublic.com/article/news-social-engineering-top-cyber-threat-2025/ (accessed on 15 January 2026).
3. Hunkenschroer, A.L.; Luetge, C. Ethics of AI-Enabled Recruiting and Selection: A Review and Research Agenda. *J. Bus. Ethics* **2022**, *178*, 977–1007. [CrossRef]
4. Akkaya, O.; Keleştemur, S.A. Quantifying Social Engineering Impact: Development and Application of the SEIS Model. *Int. J. Sci. Res. Eng. Dev.* **2024**, *7*. Available online: https://www.ijsred.com/volume7/issue5/IJSRED-V7I5P58.pdf (accessed on 15 January 2026).

5. Dodda, D. How I Almost Got Hacked By A 'Job Interview'. Technical Report. 2025. Available online: https://blog.daviddodda.com/how-i-almost-got-hacked-by-a-job-interview (accessed on 15 January 2026).
6. Dang, Q.-V. Enhancing Obfuscated Malware Detection with Machine Learning Techniques. In *Future Data and Security Engineering*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 731–738. [CrossRef]
7. Norris, G.; Brookes, A.; Dowell, D. The Psychology of Internet Fraud Victimisation: A Systematic Review. *J. Police Crim. Psychol.* **2019**, *34*, 231–245. [CrossRef]
8. Mateo Sanguino, T.J. Enhancing Security in Industrial Application Development: Case Study on Self-generating Artificial Intelligence Tools. *Appl. Sci.* **2024**, *14*, 3780. [CrossRef]
9. Wong, M.Y.; Valakuzhy, K.; Ahamad, M.; Blough, D.; Monrose, F. Understanding LLMs Ability to Aid Malware Analysts in Bypassing Evasion Techniques. In Proceedings of the ACM ICMI Companion '24, San Jose, Costa Rica, 4–8 November 2024. [CrossRef]
10. Hadnagy, C. *Social Engineering: The Science of Human Hacking*, 2nd ed.; Wiley: Hoboken, NJ, USA, 2018. [CrossRef]
11. Bloxberg, D. Social Engineering a Fake Interview or a Fake Job Candidate. Inspired eLearning. 2024. Available online: https://inspiredelearning.com/blog/social-engineering-fake-interview-candidate (accessed on 15 January 2026).
12. Steinmetz, K.F.; Holt, T.J. Falling for Social Engineering: A Qualitative Analysis of Social Engineering Policy Recommendations. *Soc. Sci. Comput. Rev.* **2023**, *41*, 592–607. [CrossRef]
13. Zaoui, M.; Yousra, B.; Yassine, S.; Maleh, Y.; Ouazzane, K. A Comprehensive Taxonomy of Social Engineering Attacks and Defense Mechanisms: Toward Effective Mitigation Strategies. *IEEE Access* **2024**, *12*, 72224–72241. [CrossRef]
14. Cialdini, R.B. *Influence: The Psychology of Persuasion*; HarperCollins Publishers Ltd.: New York, NY, USA, 2021. Available online: https://ia800203.us.archive.org/33/items/ThePsychologyOfPersuasion/The%20Psychology%20of%20Persuasion.pdf (accessed on 15 January 2026).
15. Stajano, F.; Wilson, P. Understanding Scam Victims: Seven Principles for Systems Security. *Commun. ACM* **2011**, *54*, 70–75. [CrossRef]
16. Workman, M. Gaining Access with Social Engineering: An Empirical Study of the Threat. *Inf. Syst. Secur.* **2007**, *16*, 315–331. [CrossRef]
17. Madhavi, D.; Reddy, M.S.M.; Ramya, M. Internet Employment Detection Scam of Fake Jobs via Random Forest Classifier. In *Springer Proceedings in Mathematics & Statistics*; Springer: Cham, Switzerland, 2024; Volume 438. [CrossRef]
18. Havránek, M. Deceptive Development Targets Freelance Developers. ESET Research. Technical Report. 2025. Available online: https://www.welivesecurity.com/en/eset-research/deceptivedevelopment-targets-freelance-developers/ (accessed on 15 January 2026).
19. Whysall, Z. Cognitive Biases in Recruitment, Selection, and Promotion: The Risk of Subconscious Discrimination. In *Hidden Inequalities in the Workplace; Palgrave Explorations in Workplace Stigma*; Palgrave Macmillan: Cham, Switzerland, 2018. [CrossRef]
20. Gambín, Á.; Yazidi, A.; Vasilakos, A.; Haugerud, H.; Djenouri, Y. Deepfakes: Current and future trends. *Artif. Intell. Rev.* **2024**, *57*, 64. [CrossRef]
21. Pandya, K. Obfuscation 101: Unmasking the Tricks Behind Malicious Code. Socket, Inc. Technical Report. 2025. Available online: https://socket.dev/blog/obfuscation-101-the-tricks-behind-malicious-code (accessed on 15 January 2026).
22. Dutta, T.S. Threat Actors Weaponize Malicious Gopackages to Deliver Obfuscated Remote Payloads. Cryptika Cybersecurity. Technical Report. 2025. Available online: https://cybersecuritynews.com/threat-actors-weaponize-malicious-gopackages/ (accessed on 15 January 2026).
23. Al-Karaki, J.; Al-Zafar Khan, M.; Omar, M. Exploring LLMs for Malware Detection: Review and Framework. *arXiv* **2024**. Available online: https://arxiv.org/abs/2409.07587 (accessed on 15 January 2026).
24. Gajjar, J. MalCodeAI: AI-Powered Malicious Code Detection. GitHub. 2024. Available online: https://github.com/JugalGajjar/MalCodeAI (accessed on 15 January 2026).
25. Akinsowon, T.; Jiang, H. *Leveraging LLMs for Behavior-Based Malware Detection*; Sam Houston State University: Huntsville, TX, USA, 2024. Available online: https://hdl.handle.net/20.500.11875/4866 (accessed on 15 January 2026).
26. Hamdi, A.; Fourati, L.; Ayed, S. Vulnerabilities and attacks assessments in blockchain 1.0, 2.0 and 3.0: Tools, analysis and countermeasures. *Int. J. Inf. Secur.* **2024**, *23*, 713–757. [CrossRef]
27. Gavrila, R.; Zacharis, A. Advancements in Malware Evasion: Analysis Detection and the Future Role of AI. In *Malware; Advances in Information Security*; Gritzalis, D., Choo, K.K.R., Patsakis, C., Eds.; Springer: Cham, Switzerland, 2025; Volume 91. [CrossRef]
28. Siddiqi Prity, F.; Islam, M.S.; Fahim, E.H.; Hossain, M.M.; Bhuiyan, S.H.; Islam, M.A.; Raquib, M. Machine learning-based cyber threat detection: An approach to malware detection and security with explainable AI insights. *Hum.-Intell. Syst. Integr.* **2024**, *6*, 61–90. [CrossRef]
29. Jiang, Y.; Meng, Q.; Shang, F.; Oo, N.; Minh, L.T.H.; Lim, H.W.; Sikdar, B. MITRE ATT&CK Applications in Cybersecurity and The Way Forward. *arXiv* **2025**, arXiv:2502.10825v1. Available online: https://arxiv.org/html/2502.10825v1 (accessed on 15 January 2026). [CrossRef]
30. Shannon, C.E. A Mathematical Theory of Communication. *Bell Syst. Tech. J.* **1948**, *27*, 379–423. [CrossRef]

31. McCabe, T.J. A Complexity Measure. *IEEE Trans. Softw. Eng.* **1976**, *SE-2*, 308–320. [CrossRef]
32. Schrittwieser, S.; Wimmer, E.; Mallinger, K.; Kochberger, P.; Lawitschka, C.; Raubitzek, S.; Weippl, E.R. Modeling Obfuscation Stealth Through Code Complexity. In *Computer Security—ESORICS 2023 Workshops*; Springer: Cham, Switzerland, 2024; pp. 392–408. [CrossRef]
33. Kumar, A.; Dubey, K.K.; Gupta, H.; Lamba, S.; Memoria, M.; Joshi, K. Keylogger Awareness and Use in Cyber Forensics. In *Rising Threats in Expert Applications and Solutions*; Rathore, V.S., Sharma, S.C., Tavares, J.M.R., Moreira, C., Surendiran, B., Eds.; Springer: Singapore, 2022; pp. 719–725. [CrossRef]
34. Kührer, M.; Rossow, C.; Holz, T. Paint It Black: Evaluating the Effectiveness of Malware Blacklists. In *Research in Attacks, Intrusions and Defenses; RAID 2014; Lecture Notes in Computer Science*; Stavrou, A., Bos, H., Portokalidis, G., Eds.; Springer: Cham, Switzerland, 2014; p. 8688. [CrossRef]
35. Rayson, P.; Garside, R. Comparing corpora using frequency profiling. In Proceedings of the Workshop on Comparing Corpora, ACL, Hong Kong, 7 October 2000. Available online: https://dl.acm.org/doi/10.3115/1117729.1117730 (accessed on 15 January 2026).
36. Loria, S. TextBlob: Simplified Text Processing. 2018. Available online: https://textblob.readthedocs.io (accessed on 15 January 2026).
37. Kruskal, W.H.; Wallis, W.A. Use of Ranks in One-Criterion Variance Analysis. *J. Am. Stat. Assoc.* **1952**, *47*, 583–621. [CrossRef]
38. Chatzoglou, E.; Kambourakis, G. Bypassing Antivirus Detection: Old-School Malware, New Tricks. *arXiv* **2023**, arXiv:2305.04149. Available online: https://arxiv.org/pdf/2305.04149.pdf (accessed on 15 January 2026). [CrossRef]
39. Mawgoud, A.A.; Rady, H.M.; Tawfik, B.S. A Malware Obfuscation AI Technique to Evade Antivirus Detection in Counter Forensic Domain. In *Enabling AI Applications in Data Science*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 597–615. [CrossRef]
40. Satriawan, A. From Shrimps to Whales: Knowing the Holder Crypto Hierarchy. BizTech. Technical Report. 2024. Available online: https://biztechcommunity.com/news/from-shrimps-to-whales-knowing-the-holder-crypto-hierarchy/ (accessed on 15 January 2026).
41. Mostyn, S. How Many Job Applications Does It Take to Get a Job? Career Agents. Technical Report. 2025. Available online: https://careeragents.org/blog/how-many-job-applications-does-it-take-to-get-a-job/ (accessed on 15 January 2026).
42. Verizon. 2024 Data Breach Investigations Report. Verizon Business. 2024. Available online: https://www.verizon.com/business/resources/T6de/reports/2024-dbir-data-breach-investigations-report.pdf (accessed on 15 January 2026).
43. *ISO/IEC 27001:2022*; Information Security, Cybersecurity and Privacy Protection—Information Security Management Systems—Requirements. International Organization for Standardization: Geneva, Switzerland, 2022. Available online: https://www.iso.org/standard/82875.html (accessed on 15 January 2026).
44. National Institute of Standards and Technology. *Security and Privacy Controls for Information Systems and Organizations (NIST Special Publication 800-53, Revision 5)*; U.S. Department of Commerce: Gaithersburg, MD, USA, 2020. [CrossRef]
45. European Union Agency for Cybersecurity. *ENISA Threat Landscape 2023*; ENISA: Heraklion, Greece, 2023. Available online: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2023 (accessed on 15 January 2026).
46. Floridi, L.; Cowls, J. A Unified Framework of Five Principles for AI in Society. *Harv. Data Sci. Rev.* **2019**, *1*, 535–545. [CrossRef]