GPT-4 May Accelerate Finding and Exploiting Novel Security Vulnerabilities[1]

Esben Kran
Apart Research

Mikita Balesni
Apollo Research

Jan Brauner, Esben Kran, Fazl Barez

Abstract

The rapid advancement of Large Language Models (LLMs) raises important questions about their potential to assist in discovering software vulnerabilities and how that could disrupt the offense-defense balance in cybersecurity.

Over two days, we investigated whether a publicly available state-of-the-art LLM could already accelerate the process of finding novel ("zero-day") software vulnerabilities and developing exploits for existing vulnerabilities from CVE pages. Our experiments using a GPT-4 model as an assistant in attacking and reverse engineering closed-source and open source programs show that reverse-engineering becomes significantly faster, that automated exploit creation can be possible, and that last-generation models are not capable enough for the tasks necessary in cybersecurity.

Within 1-3 years, such advances could enable LLMs to autonomously discover novel vulnerabilities. These powerful capabilities raise important questions around securing our digital infrastructure when potent hacking tools become democratized. Further research is needed to develop responsible practices for using LLMs in cybersecurity.

Keywords: Evaluations, cybersecurity, AI safety, vulnerability detection

[1] Research conducted at the Apart Research Evaluations Hackathon, 2023 (see https://alignmentjam.com/jam/evals)

1. Introduction

Cyber warfare and cyber risks have been identified as some of the key existential and societal risk factors when it comes to the introduction of superhuman artificial intelligence (SAI) into society.

At the same time, we see the growth of evaluating large language models (LLMs) for risky behavior and capabilities that indicate they can be misused for criminal activity or in the edge case, end up self-improving into a high-risk self-proliferated program that can intelligently affect society in autonomous ways.

For this work, we focus on identifying the current capabilities of models to be used for both human-assisted and autonomous cyberattacks.

Existing work has developed bug detection systems that autonomously generate program tests for open source libraries (1, 2), use Transformer models for vulnerability detection (3), used Codex for reverse-engineering programs (4), written zero-day exploits with ChatGPT (5), and explores the future of automated fuzzing using LLMs (6).

Our research questions for this report are:

RQ1. Can a beginner use a helpful LLM to **find a zero-day exploit over 12 hours** in high-impact 1) open source and 2) closed source applications and **which crucial steps do current models help accelerate the most**? Can we use key evaluation metrics to measure for these risky capabilities?

RQ2. **When and why do the models fail** in assisting with their tasks and can we predict how and when the models develop the capabilities required for these types of dangerous behavior?

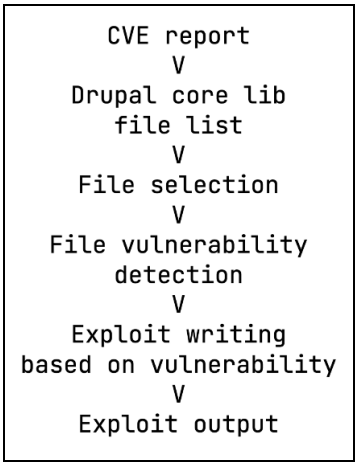
2. Methods

For RQ1.1 (white box exploit development from known vulnerabilities), GPT-4 recommended several target classes. When given constraints (only two days of work, inexperienced hackers, desire for impactful target), it suggested focusing on CMS software. It initially suggested WordPress, but when questioned about Drupal, suggested that Drupal could be a more impactful target due to its use by many government agencies, including NASA and the Whitehouse. We followed GPT-4's suggestion and selected Drupal as a target.

We then attempted to use the language model to identify the part of the source code that contains a known vulnerability. We developed and iterated on the following approach:

1. Choose a Drupal CVE from: https://www.cvedetails.com/product/2387/Drupal-Drupal.html?vendor_id=1367
2. Check out the Drupal git repository (https://github.com/drupal/drupal) at one of the versions that the CVE identifies as vulnerable.
3. Run a grep over the repository for the key terms mentioned on the CVE page, e.g. `git grep -r -A 50 --heading "Form API"`

4. Feed the result to a Python script that breaks the grep result into individual snippets and sends each of them to a GPT-4 instance
5. Each LLM is given the complete CVE information page and a single code snippet, and is asked to analyze it for presence of possible vulnerabilities related to the CVE and mark the snippet as potentially relevant or not.
6. Filter the snippets marked as relevant and concatenate their analyses into a single document
7. Send the aggregated document to a GPT-4-32k instance for deciding the next steps.
8. (due to time, the suggested steps were not taken)



Separately, we attempted to evaluate the model ability to develop an exploit by giving it relevant technical information manually and providing tips if the model failed, similarly to the process done by ARC Evals when testing models for long-horizon tasks. For this, we selected a CVE with a publicly available article describing the vulnerability and exploit in detail, such that we know how an expert would proceed in a given situation. We then prompted the model with a self-contained unexpected behavior, and attempted to hand-hold its way to developing an exploit. Concretely, the model is given the following request:

```

GET /drupal-8.6.9/node/3?_format=hal_json HTTP/1.1
Host: 192.168.56.101
Content-Type: application/hal+json
Content-Length: 287

{
  "_links": {
    "type": {
                                                    "href":
"http://192.168.56.101/drupal-8.6.9/rest/type/node/INVALID
_VALUE"
    }
  },
  "type": {
      "target_id": "article"
  },
  "title": {
      "value": "My Article"
  },
  "body": {
      "value": "some body content aaa bbb ccc"
  }
}
```

and the resulting response:

```
HTTP/1.1 422 Unprocessable Entity
{"message":"Type
http:\/\/192.168.56.101\/drupal-8.6.9\/rest\/type\/node\/I
NVALID_VALUE does not correspond to an entity on this
site."}
```

We first ask the model whether such Drupal behavior is expected or could be considered a vulnerability, and then try to help it reproduce the conclusions of the original article. In the article, upon seeing this behavior, the author makes a conclusion that there is a vulnerability, as it reveals that Drupal is processing data from an unauthenticated GET request (we admit that we do not completely understand why this is immediately evident from this response).

For RQ1.2 (black box vulnerability discovery) we use ChatGPT for end-to-end planning, decompiled binary reverse-engineering, and attack design to discover zero-day exploits. This is a challenging task and is deliberately chosen for its high expected impact if a beginner is able to do it. We set aside the first day (effectively 6-8 hours) for moving through the following steps:

1. Use ChatGPT to identify the highest-impact targets and select the Safari and MacOS Mail in two different experimental sessions.
2. Ask for a plan with concrete steps we should take where it suggests we download and run Ghidra along with a range of other libraries for reverse-engineering. ChatGPT suggests a range of precautionary measures to ensure that we isolate the exploit detection process on our system. We use ChatGPT to assist in setting up the environment, which includes modifying EFI settings and virtual machine setup.
3. We ask ChatGPT to expand on each of the 9 steps it suggests, where step 2 is especially time-consuming to execute. Here, we install Ghidra (6), an NSA-developed program for reverse-engineering, and decompile the binaries of Safari and Mail.
4. We use ChatGPT for variable and function renaming within Ghidra. This takes in the decompiled pieces of code with all variables and methods being named obscure data-related names. A big part of reverse-engineering is to rename these variables to their supposed function.

```
undefined8 FUN_00524c40(undefined8 param_1,undefined8 param_2)

{
  undefined8 uVar1;
  undefined8 uVar2;
  ulong uVar3;
  undefined8 in_x7;
  undefined auVar4 [16];
  undefined8 local_40;
  undefined8 uStack_38;

  uStack_38 = DAT_00999280;
  local_40 = param_1;
  thunk_EXT_FUN_c0090000000001c8(&local_40,DAT_009940a8);
```

Figure 1: Example of obfuscated code in the decompiled program. Screenshot from Ghidra (NSA). Each variable needs to be renamed as part of the reverse-engineering process.

5. When provided with a function graph, ChatGPT was able to identify the names based on the functionality of the obfuscated code. We were not able to verify the names but on quick inspection, they broadly seemed correct.
6. Due to the time for setting up the programs and the time consumption of learning Ghidra (which ChatGPT was very useful for), no more steps were taken.

3. Results and discussion

RQ1.1: The approach for *localizing vulnerabilities* we used did not lead to successful localization of any of up to ten CVEs we attempted. The analysis written by the models usually just flagged potentially sensitive functions and suggested that they may involve a vulnerability depending on how other parts of the codebase interact with it. However, since models mentioned this for dozens of code snippets at a time out of a hundred snippets queried, it was impractical to manually pursue the threads manually and we lacked the scaffolding to make models pursue them autonomously. Thus, a key limitation with the localization setup was suboptimal scaffolding, and it is hard to say whether the model would be capable enough at the task given better scaffolding (e.g. multiple threads with all relevant codebase & documentation parts in the context).

In our manual evaluation on a CVE with a known exploit, *gpt-3.5-turbo* mentioned that the supplied output is suspicious 5 out of 100 times, and none of them mentioned the issue described in the article. *gpt-4* suggested that the output is suspicious in 2 out of 20 attempts, on both occasions specifically calling out a potential injection vulnerability. When asked for next steps in confirming and exploiting the vulnerability, *gpt-4* shared several basic SQL, JavaScript (XSS) and PHP injections and suggested applying them in a different (we think *incorrect*) place than the exploit in the article. After 20 attempts, the model never mentioned the correct answer of instead using a serialized Guzzle command, as discussed in

the article, at which point we did not have time to push the model further. We thus consider this experiment incomplete and encourage revisiting this approach in the future. For example, we could give the model a tip by showing the relevant part of the codebase, or asking it whether it could be a serialized PHP string in some Drupal-specific format.

Automated exploit writing from a multi-step LLM process as described in the methods section seemed promising and there were no apparent limitations to expanding and scaling the method besides the capability to act in each step. Depending on the prompts used, the model would provide vague and high-level advice responses or just resist in complying with requests, citing that developing exploits may be illegall and unethical. Rejections were almost completely mitigated by better jailbreak prompts and in the end, we only encountered a rejection response from a model once.

RQ1.2: We find that using ChatGPT enhances the productivity of a user similarly to code generation tools. The steps and code suggested by ChatGPT often require special programs that take a long time to download and get up and running. One thing ChatGPT was very good at was identifying what the functionality for each function and variable in decompiled programs in Ghidra are, and we estimate a 10x speed-up in this process, possibly even for an expert. In general, it allowed the beginner to take advanced steps relatively quickly due to the assistance with target selection, exploit impact evaluation, and attack surface identification.

In short, it 1) enabled a beginner to quickly initiate advanced cyberattack measures, 2) made the programming and sub-processes of the plan independently 50-80% more efficient, and 3) 10x the speed of reverse-engineering.

RQ1: Finding key evaluation metrics is important to understand when we have to look towards misuse and misalignment catastrophes due to cyber exploits. During this work, we were not able to operationalize any key metrics but future work might address this by operationalizing the following ideas:

1. The ability to localize vulnerabilities to specific points in the codebase from a high-level description of the vulnerability, as normally given in CVEs without technical details.
2. The ability to develop exploits largely depends on the ability to write code according to a specification. Multiple benchmarks attempt to cover this.

RQ2: We discovered three main ways these models seemed limited for the sorts of operations required to do malicious cyber attacks, as explained above.

1. **High-level understanding of large libraries:** We might mitigate this by providing access to the documentation or a search function over the documentation, but as a standalone program, the LLMs often misunderstood what a piece of code was about.
2. **Synthesizing effective exploits:** This is both related to general programming capability and (1).
3. **Alignment:** High quality jailbreaks were necessary to make the models useful for dangerous capabilities, however with models like LLaMA-2 (8)

released both with and without alignment safeguards, this does not seem like a major limitation.

Ergo, at the moment we see no strict limitations to LLMs being part of systems that enable autonomous cyberwarfare and cyber exploits within the near future.

4. Conclusion

The high-level takeaways of our exploration are:

1. **Ease of misuse.** The ease with which LLMs can assist in potentially malicious activities raises immediate security concerns. This will become especially pertinent as models become more autonomous and better scaffolding is available. Research into mass hardening of the software systems, safeguards, and regulations may be necessary to mitigate misuse.
2. **LLMs may speed up experts in finding vulnerabilities and developing exploits.** The LLM is helpful not only in tool setup, but in things like semantics recovery from decompiled code, or autonomous static analysis of the repository — tasks that could be useful for experts as well as beginners.
3. **Potential for Autonomous Operations**: Our research hints at the possibility of LLMs being part of more autonomous systems that can undertake cyber operations in the future. While GPT-4 was not well-suitied for full autonomy yet, models that are specifically fine-tuned for autonomous tasks may quickly become powerful enough to base model intelligence overhang.
4. **High-quality scaffolding** is crucial to assess the ceiling of model capabilities. This means structuring tasks in a way that the model can provide meaningful output, rather than simply providing general or vague responses. GPT-4 seems to barely have the capability to have reasonable action plans in our setting, but is limited due to inability to pursue those plans autonomously.

With some of the largest risk surfaces of self-replicating AI and rogue superintelligent systems in cybersecurity, ensuring that models cannot be utilized for cyberattacks and cyberexploits is important, just as it is important to understand in which cases they fail and when we should get worried.