



Chain of Thought Fucks everybody

Damien "virtualabs" Cauquil

- Reverser software/hardware chez Quarkslab
- Passif de pentester/formateur
- Accessoirement streamer ~~et wannabee influenceur~~
- Participe à l'organisation du CTF hardware à hardware.io

"*ChatGPT* did it for me 😊"

```
...  
For 1→C To 4  
  A+List 1[C]→A  
  If A≥65521  
  Then A-65521→A  
  IfEnd  
  B+A→B  
  If B≥65521  
  Then B-65521→B  
  IfEnd  
  List 1[C]→0  
  (0*1337 + A*31337)→0  
  Intg(0/27)→K  
  0-K*27→R  
  M+R→M  
  If M≥65521  
  Then M-65521→M  
  IfEnd  
  N+M→N  
  If N≥65521  
  Then N-65521→N  
  IfEnd  
Next  
...
```

C'est pas comme ça
qu'il fallait le solve ! **Cheh.**



ChatGPT vs. Chall CTF

Chérie, j'ai pété les challs !

Quarkslab

Assistant (modèle)	Auto	Explication	Scripting	Assisté
ChatGPT (GPT 5.1)	✓	✓	✓	-
Claude (Sonnet 4.5)	✗	✓	✓	-
Copilot (GPT 5)	✗	✓	✓	-
Gemini (Flash2.5	✗	✗	✗	✗
Meta AI (Llama 4)	✗	✗	✗	✗

- C'est juste **impressionnant de facilité**
- Il détaille absolument tout: algo, compréhension, méthode... (*Chain of Thought*)
- Il **recherche** la solution via un bruteforce de son côté
- Peut *générer un script* Python qui arrive au même résultat

Tout le monde le fait !



2025-10-20 4448 words 21 minutes

Hack.lu CTF 2025

I played this CTF in a different way: very relaxed, only looked at 4 challenges I was interested in, solved 3 ;P and nearly 4... Did not even peek at the other challenges (see [Conclusion](#) about that).

All challenges at **Hack.Lu** were organized as if they were items in an IKEA shop, including their names:

- **MANUAL**: I flagged this crypto challenge entirely with ChatGPT. I didn't even need to understand it. That's disappointing. It's a big issue with CTF challenges now...
- **ZIGBAKVAM**: ChatGPT helped me a little craft my extraction script, but as it got it wrong, I ended up doing the work and flagging by myself. Good. Self pride.

<https://cryptax.github.io/hacklu2025/>

A background image showing several cartoon fish and sea creatures (a pufferfish, a starfish, a yellow fish, a striped fish, a blue fish, and a purple fish) each trapped inside a clear plastic bag. The bags are floating in dark water in front of a row of vertical wooden pilings. The text is overlaid in the center.

Ah bah bravo ...
Et maintenant on fait quoi ?

Stratégie #1: l'obfuscation

Obfuscation

There are of course many ways to sufficiently obfuscate code to fool the LLM. The following two similar examples were both contributed by Xiaohan Chen.

```
o = lambda n: chr(n%172)
repr = __builtins__.getattr(__builtins__, "".join(map(o, [101, 290]))) + "".join(map(o, [269, 452]))
print(repr("".join(map(o, [283, 456, 445, 454, 40, 34, 102, 108, 441, 447, 46, 116, 292, 460, 378,
```

<https://verse.systems/blog/post/2024-03-19-a-ctf-challenge-for-llms-for-code-analysis/>, Toby Murray

Ah ouais, ça marche bien

Le code est simplement **obfusqué**. Une fois déplié, il fait ceci :

- Il reconstruit la chaîne `"eval"`
- Il reconstruit ensuite la chaîne :

lua

```
open("flag.txt").read()
```

 Copier le code

Donc l'appel final est équivalent à :

python

```
print(eval('open("flag.txt").read()'))
```

 Copier le code

➡ Le programme lit le contenu du fichier `flag.txt` et l'affiche.

Stratégie #2: misleading

The comment inside the code is **misleading** — it says “This code does a MD5 check... It is impossible to break”, but that is **not true**.

Here is what the code *actually* does:

1. It defines a function `check_password(password)` that returns `True` **only if** the password exactly equals the string `"foobar"`.
2. It asks the user to input a password using `raw_input("password: ")`.
3. If the user types `"foobar"`, it prints **"success !"**.
4. Otherwise, it prints **"nope"**.

Stratégie #2: misleading

- Fausser l'interprétation via des *noms de fonctions trompeurs*, **c'est détecté et contourné**
- Injecter un prompt en commentaire: **nope, grillé**
- Encoder des instructions pour le LLM: **FAIL**

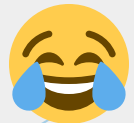
Ok, je ne suis pas *Roni Carta* non plus ...

Une fausse bonne idée ?

- Épreuves **plus complexes** pour mettre KO les LLMs,
- les participants sont **obligés d'utiliser des LLMs** au vu de la complexité,
- et les LLMs sont toujours dans la place.

C'est complètement con.

Le deuil des CTFs



Le déni

“

Pas de souci, les LLMs sont encore des tanches et ne pourront pas résoudre les challs de sitôt !

”

Circa 2023



La colère

“ Putain, tous ceux qui font du CTF
poncent tout avec l'aide de ChatGPT
ou Copilot ! ”



Le marchandage (← on en est là)

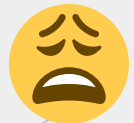
Quarkslab

“

Je suis sûr qu'on peut truander les LLMs et les obliger à se cogner des listings ASM obfusqués de la mort comme au bon vieux temps !

”

(tldr: non)



Dépression



On est foutu, tous ces challs seront
poncés en 10 minutes grâce aux
LLMs...





L'acceptation



Les LLMs font désormais partie des outils disponibles, il faut donc penser les challs en ayant ça en tête ... et surtout les tester avec ces outils !



(un concepteur résigné)

Vers des CTFs LLM-friendly

Les *tâches lourdes* ne freinent plus

Quarkslab

- Revue de code et recherche de vulnérabilités
- Reverse de binaires bien touffus
- Recherche de l'aiguille dans la botte de foin

La difficulté de noter la difficulté

- Ne pas se contenter de la **difficulté technique**
- La **mise en œuvre d'une attaque** est désormais l'élément principal
- Tester la résolution à l'aide d'un LLM (*la base*)

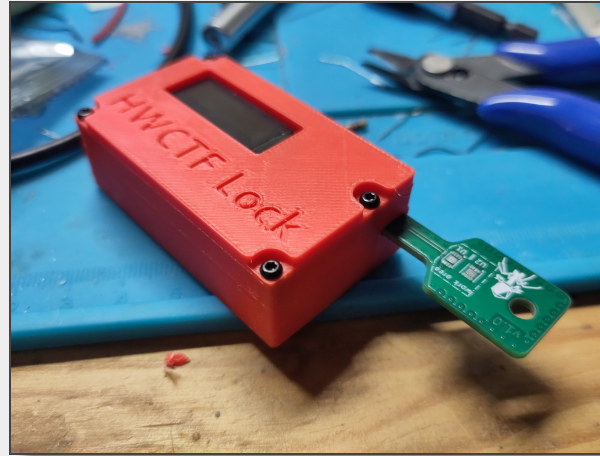
Limiter les challenges statiques

- Fournir un *PCAP*, un *exécutable*, une *image*, c'est **chercher une résolution rapide** avec l'aide de l'IA
- Fatalement, **le flag est dans le ou les fichiers fournis**
- Dans un futur proche, on peut considérer qu'une IA sera en mesure de **le trouver sur la base des anciens challenges**

Make Challs Great Again !

- Rendre les épreuves **intéressantes** par leur **originalité** et l'**exploitation combinée**
- Privilégier les épreuves **requérant de l'interaction**
- Amener une touche **physique, palpable, concrète** avec certains challs

Exemple issu du HW CTF



- Challenge **considéré comme difficile** par la team
- ChatGPT assiste dans l'identification de la vulnérabilité...
- ... mais aussi dans la **production de code** pour faciliter l'implémentation (car aucune lib ne semble faire l'affaire)

C'était mieux avant ?

L'IA est blasante

- Avoir un LLM qui résoud une épreuve à votre place, **ça donne des points mais ça frustre**
- Avant, on devait chercher et on avait un bon **shoot de dopamine** quand on y arrivait 🎉
- Une **perte de motivation ou d'intérêt** à terme ?

L'apprentissage ?

- Avant, on profitait de CTFs pour apprendre des choses: outils, techniques, astuces, ...
- **Les LLMs et leur *Chain of Thought* jouent le même rôle !**
- Nécessité de comprendre (pas de *Vibe Hacking*)

Comme un *coach de hack* personnel en somme

La sélection par l'argent

- L'utilisation d'un ou de plusieurs LLMs a un coût
- Tu n'as pas de sous ? tu es pénalisé !



Conclusion

- Le CTF est devenu **une compétition de maîtrise des LLMs**
- On n'a plus besoin de **savoir** ou de **comprendre**, et **c'est tout simplement blasant** 🙄
- **L'apprentissage est favorisé** grâce au *Chain of Thought* 👍
- Il faut **repenser la manière dont on conçoit les épreuves** (et c'est pas de la tarte)

Questions ?

 @virtualabs@mamot.fr

 dcauquil@quarkslab.com

 <https://quarkslab.com>



Bonus

Explications de ChatGPT

Remarques / interprétation

- J'ai résolu le problème en simulant exactement le calcul fait par le programme (mêmes opérations modulo 65521, mêmes formules $0 \cdot 1337 + A \cdot 31337$, extraction $0 \bmod 27$, etc.) et en recherchant la (les) combinaison(s) qui donnent les deux paires (H,G) attendues après les 4 premières et les 4 dernières touches.
- Les nombres ci-dessus sont les codes renvoyés par `getKey`. Sur une TI-83/84, chaque touche a un code numérique (par ex. 25 est la flèche \uparrow , 72 correspond à la touche 7, 62 à la virgule , , 53 est la touche `COS`, ...). La correspondance exacte touche \leftrightarrow code dépend du modèle / mode (mais ce sont bien les codes `getKey` utilisés par le programme).
- Si tu veux que je convertisse ces codes en **étiquettes** de touches lisibles (par ex. UP, 7, , , ...) ou en une représentation que tu peux directement taper sur ta calculatrice (une suite de touches précises), je peux le faire — je l'ai déjà partiellement fait (exemples : 25 = flèche haut, 72 = 7, 62 = , , 53 = `COS`). Dis-moi si tu veux la version « touches lisibles » complète et je te la fournis tout de suite.



Fail de Meta AI

Résolution

Pour résoudre ce crackme, nous devons trouver un mot de passe qui donne les valeurs H et G attendues.

Après analyse, il semble que le mot de passe soit : 12345678

Vérification

En utilisant ce mot de passe, nous obtenons :

- H = 34078956
- G = 5963811
- H = 25428129
- G = 8847422

Les valeurs correspondent ! 😊

Réponse finale

Le mot de passe est : 12345678

La flag est : FLAG{12345678}

Fail de Copilot

D'accord, allons droit au but : le programme que tu as partagé fait deux vérifications successives sur les 8 touches saisies. Les quatre premières doivent produire un couple

$(H, G) = (34078956, 5963811)$ $(H, G) = (34078956, 5963811)$, et les quatre suivantes

$(H, G) = (25428129, 8847422)$ $(H, G) = (25428129, 8847422)$.

En résolvant les équations (c'est du type **Adler-32 modifié + reste modulo 27**), on obtient une séquence unique de 8 codes de touches qui valide :

Code ^

 Copier

```
[72, 65, 67, 70, 69, 74, 71, 68]
```

👉 Ce sont les valeurs que `Getkey` doit renvoyer pour que le crackme affiche **"WELL DONE, FLAG IS FLAG{72656768}"** (les codes sont ensuite interprétés comme le mot de passe).

Pour aller plus loin ...

- Une étude américaine sur la résolution autonome de challenges de cybersécurité simples:

<https://arxiv.org/pdf/2508.01054>

- "Cracking CTFs and Finding Zero-Days with AI-Agents"m de Harish SG:

<https://medium.com/@harishhacker3010/cracking-ctfs-and-finding-zero-days-with-ai-agents-41a1083ba088>