AUTOMATING THE SECURE DEPLOYMENT OF CTF CHALLENGES WITH LLMs

Rareș-Mihail Visalom[1], Răzvan Deaconescu[2], Răzvan Rughiniș[3]

The rapid evolution of the Internet and IT systems has led to the need to improve the overall security of the cyberspace. In this dynamic context, Capture The Flag contests have become one of the main practical means of teaching cybersecurity concepts. Their wide availability has contributed to a rising popularity among different age groups, ranging from international-level industry professionals to middle-school students. This competitive environment demands constant updates of both the notions it aims to teach as well as the teaching methodologies used to convey them. Consequently, both trainers and trainees need to adapt and improve to match the requirements of remaining relevant. Our research aims to harness the advantages of Large Language Models to support the teaching efforts by easing the process of deploying CTF challenges in an automated workflow.

Keywords: Capture The Flag, CTF, LLMs, Automation, DevSecOps, Docker

1. Introduction

The academic environment is constantly in need of improved methods to convey topics across all scientific fields. Computer Science is one of the fields where practical, hands-on teaching methods have been employed with great success rates. This bridges the gap between theory and practice, connecting theoretical notions with practical implementations [1]. Moreover, there is strong evidence that CTFs increase engagement and motivation without affecting learning performance, which qualifies CTFs as a valid teaching methodology [2].

Most probably, this is an effect of how accessible both hardware and software have become. For Cybersecurity, a subset of the more broader Computer Science field, Capture-The-Flag challenges (*CTFs*) are one of the most popular activities when it comes to students testing their abilities and knowledge. CTFs help students divide a bigger problem into smaller, more manageable

[1]Doctoral Student, National University of Science and Technlogy POLITEHNICA Bucharest, Romania, e-mail: rares.visalom@upb.ro

[2]Associate Professor, National University of Science and Technlogy POLITEHNICA Bucharest, Romania, e-mail: razvan.deaconescu@upb.ro

[3]Professor, National University of Science and Technlogy POLITEHNICA Bucharest, Romania, e-mail: razvan.rughinis@upb.ro

parts and then focus on each of the problems in sequence to solve the whole challenge. This is most probably the benefit that makes CTFs such a useful tool for teaching [3].

However, lessons learned from organizing CTFs empirically prove that the CTF model is not perfect and that this teaching methodology needs refinement and fine-tuning. Two of the least covered areas might be Social Engineering and Awareness [4]. The important challenge for organizers is managing the infrastructure used to deploy CTF challenges and, at the same time, ensuring that cheating attempts are identified immediately [5]. In terms of content, designing challenges for entry-level students is another complex problem that needs to be addressed, as beginners need guidance even during competitions.

There is proof that scenario-focused and gamified [6] approaches might be more suited for beginner-level challenges (secondary school students) [7] . This makes cybersecurity topics available to secondary school students, dramatically decreasing the entry barrier in the field. The result is that new generations of cybersecurity specialists are trained much earlier than before and will be more prepared when they join the global workforce.

CTFs can be used to enhance collaboration and networking between students. A novel approach improves the classic CTF experience by adding real-time interactions between participants who do not know each other to solve CTF challenges in a collaborative way [8].

Given this dynamic context, we believe that supporting the efforts of safely deploying CTF challenges could greatly improve the teaching experience by enabling a faster and more secure deployment of CTF challenges. We identified this need while migrating challenges from bare-metal VMs to Containers as part of the cybersecurity courses at POLITEHNICA Bucharest. Most of the migration was performed manually, which induced time costs. Our research aims to identify possible ways of using Large Language Models (*LLMs*) to automate the deployment procedure of CTF challenges.

We will describe the current context and the need to explore the deployment of CTF challenges via LLMs in Section 2. We will continue with details of our experiment in 3 before detailing our results in 4. Conclusions and potential future work are presented in 5.

2. **State Of The Art**

Rather than focusing on how to improve the deployment process, recent studies have invested effort into solving various technical problems via LLMs. We believe that lack of targeted research aimed at improving the deployment process of CTF challenges qualifies this field as a productive research direction.

LLMs were tested against CTF challenges and technical certifications questions [9]. It was discovered that various LLMs behaved differently, having different accuracy rates, which suggests that choosing the right LLM provider is highly important. However, the advantage of using LLMs to attempt to

solve technical problems is clear, but this advantage is a double-edged sword. Evidence suggests that the use of LLMs during CTFs could shift the focus from thorough understanding of the challenge to solving it as the end goal. This shift in objectives might negatively affect the learning process.

Attempts to augment the performance of LLMs solving CTFs have been studied [10]. The results are positive, which suggests that there is enough room for improvement in terms of the accuracy that LLMs can provide at this stage. In some cases, the improvements were to relevant that they allowed researchers to score in the top 23.6% of the teams participating in the picoCTF2024 competition [11].

For the purpose of hosting a CTF for educational purposes, the following components are required: Challenges, Deployment Infrastructure and CTF Platform. A complete Challenge involves description, associated code that compiles into an artifact that will then be used for deployment, solutions and automated deployment script. After the Challenge contains all prerequisites, it can then be deployed on the Deployment Infrastructure via its associated automated deployment script. The CTF Platform abstracts the complexity of the deployment process and provides an interface between CTF participants and the challenges. The participants will enroll (either individually or as a team) through the CTF Platform and will access challenges and the leaderboard through this platform.

The deployment process itself is critical because it needs to ensure that challenges are available for the duration of the CTF and that the challenges themselves do not compromise the underlying infrastructure. Hosting a CTF challenge is the same as hosting a vulnerable component and making it available to attackers. Therefore, as educators and CTF providers, the objective is to securely host an unsafe system that will eventually be exploited.

Our research explores how the deployment process could benefit from automation using LLMs to safely host the CTF challenges in a controlled deployment infrastructure. Our research assumes that the challenge itself exists and lacks automated deployment scripts. In other words, the challenge development process is a prerequisite to creating the deployment script via LLMs.

3. **Experimental Setup**

Before Containers became popular and widely used, CTF callenges were deployed on directly on Virtual Machines (*VMs*). This was a laborious process which involved extensive configuration, as every challenge needed particular settings and configuration to be done most often manually. Another disadvantage was that deploying multiple challenges on the same VM was a tedious process that could result in conflicts between the two challenge configurations. An automated, reliable means of having reproducible builds was highly needed and ultimately this meant using Containers.

Due to the lightweight requirements of running Containers and the separation they offer, they were chosen as the best solution to automate the deployment of CTF challenges. Multiple challenges could be deployed on one single VM, which decreased the requirements of having multiple VMs in the infrastructure.

The Container Runtime chosen for our research was Docker and we automated the build process via Dockerfiles. Docker containers allow us to ensure reproducible builds of our challenges. Having everything inside of a Dockerfile means that we can rebuild the Container hosting the CTF Challenge in an automated manner. The Dockerfile is what our research aimed to obtain via the use of LLMs. All files that constitute the CTF challenge could be the input that the LLM is provided before being asked to create the automated deployment script (Dockerfile). The script is then saved and executed to safely deploy the CTF challenge in the infrastructure. Figure 1 shows an overview of the interaction with the LLM.

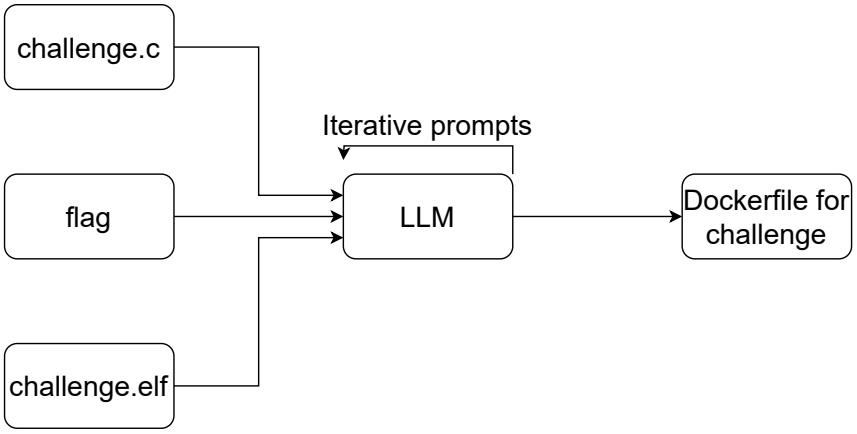

Fig 1. LLM interaction overview

The *challenge.c* source code file is used to compile the *challenge.elf* executable file. The executable file is what the CTF participants will interact with after deployment. The *flag* file is a text file containing secret information. Participants need to access this secret information and use it as proof that they solved the challenge. Figure 1 abstracts the build system that is most often comprised of a *Makefile*.

Our research is based on preexisting, complete challenges and attempts to automate their deployment in a safe and controlled environment. A complete challenge contains source files, resulting compiled files, an automated build process (usually a *Makefile*) and reference automated solution scripts. The solution scripts are meant to allow organizers to quickly test the deployment.

The LLM is given only the inputs that it needs. The types of inputs required are strictly related to the type of CTF challenge that is deployed. Our

research focused on the challenges that Jeopardy CTFs usually offer. Jeopardy CTFs offer multiple standalone challenges that are categorized based on the skills required to solve them. Participants can solve individual challenges in any order they want. Another popular type of CTF is the Attack and Defense CTF, which requires participants to engage in both offensive and defensive practices to obtain points. Our research focuses on the Jeopardy CTF type, however LLMs could most probably be used to aid in organizing Attack and Defense CTFs as well. Challenges used for both Jeopardy and Attack and Defense CTFs can also be offered as stand-alone challenges as part of *Wargames*. Wargames platforms host and offer these challenges without the time-based constraints of an active competition. This allows students to solve the challenges on a self-paced basis.

The CTF challenge categories most often employed in Jeopardy CTFs are: Web Security, Binary Exploitation, Cryptography, Reverse Engineering and Forensics. Based on the challenge type, the inputs required for the LLM to provide a reliable deployment Dockerfile will vary. Our research focused on challenges belonging to the Binary Exploitation and Web Security categories. We experiment with these challenges because once a stable process is identified for them, the same process could be employed for other challenge types as well.

Table 1 provides a summary of the challenge types we tested in our research. Moreover, the same LLM interaction could be easily adapted for other challenge types. A feasible adaptation might mean that the untested challenge type requires the same inputs and the same iterative prompts to produce the Dockerfile as one of the challenge types we tested.

Table 1. Jeopardy CTF Types for which deployment is feasible

Challenge Type	Tested	Feasible To Test
Web Security	YES	YES
Binary Exploitation	YES	YES
Cryptography	NO	YES
Reverse Engineering	NO	YES
Forensics	NO	YES*

Note that Forensics challenges might usually require a more complex setup than all the other challenges. In some cases, active interaction might generate logs that then need to be examined by students. In such cases, the inputs required for the LLM to provide the deployment script might increase substantially, based on the number of components that the challenge requires. In that case, we would have an increase of input files, but the process remains the same as with a much simpler CTF challenge type.

Comparison of various LLMs is out of scope of this paper, but such topics have been studied in depth [12] [13] [14]. We wanted to experiment with a free version of an LLM and for this purpose we chose MistralAI [15].

Needless to say, the Artificial Intelligence landscape will change as different flavors of LLMs are released. Different models will behave better than others and the comparison between different providers will change over time. Our goal was that the automation scripts be independent of the underlying LLM. With an appropriate abstraction at the implementation level, scripts could even be compatible with a selection of multiple LLMs.

We first tackled the problem of sending input files to the LLM and we identified two possible solutions. The first choice was to upload the files entirely via File Upload functionalities offered by the interface of the LLM. The second choice was to host the files and provide the LLM URLs that point to the input files. For simplicity, we opted for the second choice, as we were already hosting the files on the Github Repositories [16] used for teaching purposes at POLITEHNICA Bucharest.

An abstracted prompt for the LLM can be seen in Listing 1.

```
write a xinetd service for the following program: https://
   URL/to/source/file the service should bind on address
   0.0.0.0 and port 11111, the path to the binary should be
    /home/path/to/location and the user should be ctf. Then
    write a dockerfile to deploy the service
```

Listing 1. Example of an abstracted prompt

The most feasible way of automating such interaction would be using a script that interpolates various bits of information into a string prompt. We used the *f-string formatting* capabilities of Python3 to experiment with automating the prompts. Listing 2 shows one possibility of constructing the prompt programatically before sending it to an LLM.

```
1  url_source_file = "https://URL/to/source/file"
2  bind_addr = "0.0.0.0"
3  port = "11111"
4  filesystem_path = "/home/path/to/location"
5  system_user = "ctf"
6  service_name = "service-name"
7
8  prompt1 = f"write a xinetd service for the following
       program: {url_source_file} the service should bind on
       address {bind_addr} and port {port}, the path to the
       binary should be {filesystem_path} and the user should
       be {system_user}"
9
10 prompt2 = f"write a secure dockerfile to deploy the {
       service_name} service"
11
12 # Send prompt1 and prompt2 to LLM and retrieve the result
```

Listing 2. Python3 example of using f-strings to automate LLM interaction

Naturally, CTF challenges could depend on more than one source file. However, the problem of sending multiple files to the LLM is just an extension of prompt construction presented previously in Listing 2. A *For Loop* could be used to append as many URLs pointing to dependencies as needed into the resulting prompt. Moreover, variables found between Lines 1 and 6 in Listing 2 could be stored in a text file. An example of a reliable format to store configuration is *YAML*.

Having this automation in place would mean that we could configure prompt structures for multiple CTF challenges while storing their parameters in text files. Once the prompt structures are ready, an LLM would iteratively ingest them. After each prompt, the LLM would produce one Dockerfile per CTF challenge. We could then use this Dockerfile to deploy the CTF challenges and expose them to CTF participants.

4. **Results**

In Section 3 we detailed how we use our experience gathered through teaching cybersecurity topics at POLITEHNICA Bucharest to identify possible solutions to improve deployment of CTF challenges. The migration from hosting CTF Challenges on VMs to hosting them on Containers provided a solid base to research even more optimization methods.

We decided to explore the use of LLMs to automate the deployment of Binary Exploitation and Web Security as a starting point of our research. The first step was to identify past CTF challenges that we had previously migrated to a Dockerfile-based deployment. We would then use this set of CTF Challenges to test whether or not the LLM was capable of providing a working Dockerfile. We would then compare the Dockerfile that the LLM provided with the ones that we had written in the past for the same CTF Challenge.

Our research concluded that LLMs were capable of providing functional Dockerfiles that were correct for deploying the types of CTF challenges that we focused on. Throghout our tests, we identified 7 key differences between our past Dockerfile implementations and the Dockerfiles provided by the LLMs.

(1) LLMs use different Docker base images for Containers. This might be a consequence of our Dockerfiles using more generic base images (e.g. *debian*) and the LLMs using more specific images (e.g. *ubuntu* for Binary Exploitation challenges and *php* for Web Security challenges exposing PHP applications)
(2) LLMs use default ports for each service. Naturally, Web Security challenges would be exposed on ports 80 or 443, but there would be no default port for Binary Exploitation challenges.

(3) LLMs provide comments in the Dockerfile to make the instructions more verbose.
(4) LLMs are capable of merging multiple file types into the Dockerfile. For example, in the case of Binary Exploitation files, the Dockerfile would be the one creating the *xinetd* file, without the need for the latter to exist.
(5) LLMs will be more explicit in terms of dependencies required by the CTF challenge to run. It is not clear yet if the dependencies are already found in the base image or not. However, the verbosity does not negatively impact the functionality.
(6) LLMs implement artifact or cache cleanup stages. As a byproduct of being verbose, cleanup of the packet manager cache or various artifacts used during build stages is implemented. Lack of such cleanup stages was not found to be critical and there was no impact against the health of the final container images.
(7) LLMs were more strict when asked to implement more secure Dockerfiles. For example, the *log_ on_ failure* option is used to log failed connection attempts.

During our reserch we found that with appropriate explicit prompts, the results offered by LLMs match our initial Dockerfiles. However, there is a danger of becoming biased towards our implementation and guiding the LLM to offer a solution that we know about. To mitigate bias, we limited ourselves to as fewer prompts as possible. We found that two or three prompts are enough to generate safe Dockerfiles while completely reducing bias.

When evaluating differences between our reference Dockerfiles and the ones created by LLMs, we discounted differences in line ordering. Line ordering was not important, as long as the contents of the config file matched semantically. Moreover, unless explicitly requested, Dockerfiles exposing apps on different ports were considered equivalent.

It was not clear from our tests whether or not the default behavior of LLMs is to account for compromises between configuration options and performance or disk space. For example, logging is enabled by the LLM because it is considered to add security by making it feasible to audit the system. This will obviously incur costs in disk space. In the case of *xinetd* services, this might be negligible. However other kinds of system might input prohibitive amounts of logs as a result of enabing such a feature.

5. **Conclusions**

Our research explored the use of LLMs to automate the safe deployment of unsafe CTF challenges without compromising the networks they are deployed within. Our results are positive and they prove the feasibility and motivate future research. Our experiments have shown that LLMs can be used as helpers during the deployment process of CTF challenges. LLMs can write functional and secure Dockerfiles based on complete CTF challenge contents

like source code, problem statement and files needed to compile into a working challenge.

To further improve the process, we identified potential future research areas. CTF Challenge type coverage could be increased to be able to deploy more complex challenges like Forensics or even Attack and Defense Challenges. These types of challenges most often require active, live interaction with one or multiple hosts and could be more difficult to ensure a proper deployment.

To improve the safety of the infrastructure, we could impose a constraint to only run the challenges on top of Docker running in *Rootless Mode*. This mode requires fewer privileges for the Container to run, which improves the overall security of the infrastructure hosting the CTF Challenge.

LLMs could be used not only for deployment, but for solving the CTF Challenges, as well. This might help when trying to explore multiple solution for solving one CTF Challenge. However, this might also be used to test how easy it is to solve the challenges that we offer. This might provide an indication of how probable it would be for CTF participants to solve challenges entirely via LLMs, without their own contribution. Ultimately this might help design better challenges that are LLM-proof and require human intervention to solve.

LLMs could also be used to automatically run the reference solution against the deployment to fully assess the proper deployment and functionality of the CTF Challenge.

Attacks like Prompt Injection or Adversarial LLM usage need to be analyzed in the context of the deployment pipeline to ensure that what starts as normal, benign, usage of this infrastructure is not prone to inside threats. The risks of misconfiguration must be explored once a full solution is developed.

Future iterations need to have a more complex set of quantifiable performance indicators or key performance indicators that would be applied for a wider range of test cases (more challenge categories and more deployed challenges).

Finally, our research could expand to support deployment systems, like Kubernetes or other similar solutions.

References

[1] A. Egamberganova, U. Abdalov, S. Latipova, S. Ataev, D. Ismatova, and N. Ubaydullayeva, "Teaching cybersecurity with ctf: New pedagogical methods and strategies," in *2025 IEEE 26th International Conference of Young Professionals in Electron Devices and Materials (EDM)*, 2025, pp. 2170–2175.

[2] S. V. Cole, "Impact of capture the flag (ctf)-style vs. traditional exercises in an introductory computer security class," in *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*, ser. ITiCSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 470–476. [Online]. Available: https://doi.org/10.1145/3502718.3524806

[3] C. Nelson and Y. Shoshitaishvili, "Pwn the learning curve: Education-first ctf challenges," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSE 2024. New York, NY, USA:

Association for Computing Machinery, 2024, p. 937–943. [Online]. Available: https://doi.org/10.1145/3626252.3630912

[4] V. Švábenský, P. Čeleda, J. Vykopal, and S. Brišáková, "Cybersecurity knowledge and skills taught in capture the flag challenges," *Computers & Security*, vol. 102, p. 102154, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404820304272

[5] K. Chung and J. Cohen, "Learning obstacles in the capture the flag model," in *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*. San Diego, CA: USENIX Association, Aug. 2014. [Online]. Available: https://www.usenix.org/conference/3gse14/summit-program/presentation/chung

[6] R. Rughiniş, "Gamification for productive interaction: Reading and working with the gamification debate in education," in *2013 8th Iberian Conference on Information Systems and Technologies (CISTI)*, 2013, pp. 1–5.

[7] A. Hanafi, H. Rokman, A. Ibrahim, Z.-A. Ibrahim, M. N. Ahmad Zawawi, and F. Abdul Rahim, "A ctf-based approach in cyber security education for secondary school students," *electronic Journal of Computer Science and Information Technology*, vol. 7, 10 2021.

[8] K. Zhang, S. Wuthier, K. Yoon, and S.-Y. Chang, "Designing and using capture the flag for coordination and interaction in engineering education," in *2022 IEEE Global Engineering Education Conference (EDUCON)*, 2022, pp. 1555–1560.

[9] W. Tann, Y. Liu, J. Sim, C. Seah, and E.-C. Chang, "Using large language models for cybersecurity capture-the-flag challenges and certification questions," 08 2023.

[10] L. Muzsai, D. Imolai, and A. Lukács, "Improving llm agents with reinforcement learning on cryptographic ctf challenges," 2025. [Online]. Available: https://arxiv.org/abs/2506.02048

[11] Z. Ji, D. Wu, W. Jiang, P. Ma, Z. Li, and S. Wang, "Measuring and augmenting large language models for solving capture-the-flag challenges," 06 2025.

[12] V. Agarwal, M. K. Garg, S. Dharmavaram, and D. Kumar, ""which llm should i use?": Evaluating llms for tasks performed by undergraduate computer science students," 2024. [Online]. Available: https://arxiv.org/abs/2402.01687

[13] T. Gao, J. Jin, Z. T. Ke, and G. Moryoussef, "A comparison of deepseek and other llms," 2025. [Online]. Available: https://arxiv.org/abs/2502.03688

[14] P. Shojaee, I. Mirzadeh, K. Alizadeh, M. Horton, S. Bengio, and M. Farajtabar, "The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity," 2025. [Online]. Available: https://arxiv.org/abs/2506.06941

[15] "Mistral ai," https://mistral.ai/.

[16] "Open education hub," https://github.com/open-education-hub/binary-security.