




Casper & Graiding CTFs



November 21, 2023

Contents

1	Introduction	2
1.1	Memory safety	2
1.2	Web	2
2	Practicalities	3
2.1	Report	3
2.2	Tarball with exploits	3
2.3	Grading	4
2.4	Materials and contact	5
2.5	Working on the server	5
	2.5.1 Rules	5
	2.5.2 Environment	6
3	Level descriptions	7
3.1	Resources	7
3.2	casper0	7
3.3	casper1	7
3.4	casper2	8
3.5	casper3	8
3.6	casper5	9
3.7	casper7	10
3.8	casper8	10
3.9	Web challenges	10
	3.9.1 Submission	11
	3.9.2 Resources	11
4	Debugging pointers	13

2 Practicalities

This project is to be worked on **individually** and is expected to take about 30 hours of your time. If you do not manage to finish in time, complete as many levels and their reports as possible.

You are expected to work alone and write the exploits yourself. It is not allowed to share code, exploits or (parts of) your report. Please also do not upload your solution to GitHub or other code sharing sites.

If you built on code you did not write yourself, mention the source in your report. Make sure you understand all code and text you submit. After the deadline, we will check all solutions for plagiarism.

The deadline for the project is **December 15th (Friday) 23:59**. Submitting late is not possible, do not ask for an extension on the deadline, it will not be granted! To make sure we have something to grade, upload something well before the deadline, even if you plan to work until the last minute.

2.1 Report

We expect 2 files as deliverables: a PDF report (**report.pdf**) and a tarball with exploits (**exploits.tar.gz**). These files have to be submitted through Toledo. An example report discussing **casper3** can be found on Toledo.

The report must be a single PDF file and contain an overview table and one section per level. The overview should contain a table listing the levels, together with the flag for each level and the total time spent on exploiting it. This time includes analysis, experimentation, exploitation and reporting.

For each level you solved, the following topics must be discussed in the report:

- What does the level do?
- What is the vulnerability?
- How did you exploit it?
- For the memory safety challenges: What solutions studied in the course could be used to mitigate the vulnerability and why? Make sure to mention **all** applicable defenses studied in the course, at the source code, and, if applicable, the compiler/OS level.

Be clear and complete in your answers, but do not include unnecessary information (do not write a book).

2.2 Tarball with exploits



The evaluation of your submitted exploits will be automated. Therefore, it is extremely important that you follow these guidelines **precisely**:

1. Use a tarball (.tar.gz) to send in your exploits. This tarball should be named **exploits.tar.gz** and contain a **Makefile** in its top-level directory.

2. The Makefile must have a target per level (for the low-level challenges) you have an exploit for. The target should be called the same as the name of the level it exploits. E.g., if you have exploited casper8, then `make casper8` should build and execute your exploit for the casper8 level successfully. For the SQLi challenges, you include a script for each exploit: `exploit-sqli-<n>.sh`. When run using `bash ./exploit-sqli-<n>.sh` they should print the flag.
3. The end result of the exploit is to display the flag (in the case of the C programs, by launching the `/bin/xh` shell – make sure you launch the correct shell). In the case of the SQLi vulnerabilities by using `curl` and `grep`.
4. Make sure the tarball unpacks fine with the command `tar -xzf exploits.tar.gz`, that the make targets work when executed on the server as user `casper0` and that the SQLi exploit work when running the bash script on Ubuntu (run them from your own machine). Exploits against the memory safety levels often fail because of changing environment variables. Make sure that this is not the case for you! (See also the `env` command recommended later.)
5. Make sure you pack **all** the files you need for your exploits. Your exploits will be tested on a machine with an empty `/tmp` directory.
6. Do not include binary files in your tarball. If your exploits are written in C, then include the source code and adapt the Makefile to compile them, instead of including the binary itself.
7. Finally, test your tarball with the command: `casper_verify_tarball exploits.tar.gz`. This tool will verify that your tarball is valid for submission. Only submit your tarball if the tool accepts it.

You might notice some difference between exploiting the memory safety levels using make and exploiting it without make (see point 4 above). Your exploits have to work when running them using make.

2.3 Grading

Your solutions will be graded based on the contents of the report, the automated evaluation of the submitted tarball, and the oral defense. You can submit as many times as you want before the deadline, only your last submission will be graded.

To test your understanding of the exploits and to make sure you did not submit someone else's work, we will organize an oral defense (in English) in the last couple days of the semester and in the exam period. For the precise days and sign-up forms, look on Toledo later in the semester.

Resubmitting the project in the third exam period is possible if you are retaking the exam.

You are required to submit 3 (casper5, casper7, casper8) working low-level exploits and 3 working SQL injection exploits against unique endpoints (so, each of the 3 exploits has to be for another input form). If you submit the 6


required exploits, a perfect report and score a perfect defense, you can receive the maximum score on the project.

2.4 Materials and contact

The following materials may be useful for solving the challenges and preparing for the defense:

- The course material
- The article in the references section
- Level descriptions (see section 3) and the resources mentioned there (also uploaded to Toledo)
- DuckDuckGo, stackoverflow.com, security.stackexchange.com, ...:)

If you have any general questions, please ask them on the discussion forum on Toledo. **Do not include any information in these questions that could spoil the solution for other students!**


With more sensitive questions, you can contact us at  but please do not just send in your code or exploits to debug.

There will be some exercise sessions on November 21-22 (and depending on demand optionally November 23 as well) where you can work on and ask your questions about the assignment. (But you should definitely start working before then!)

2.5 Working on the server

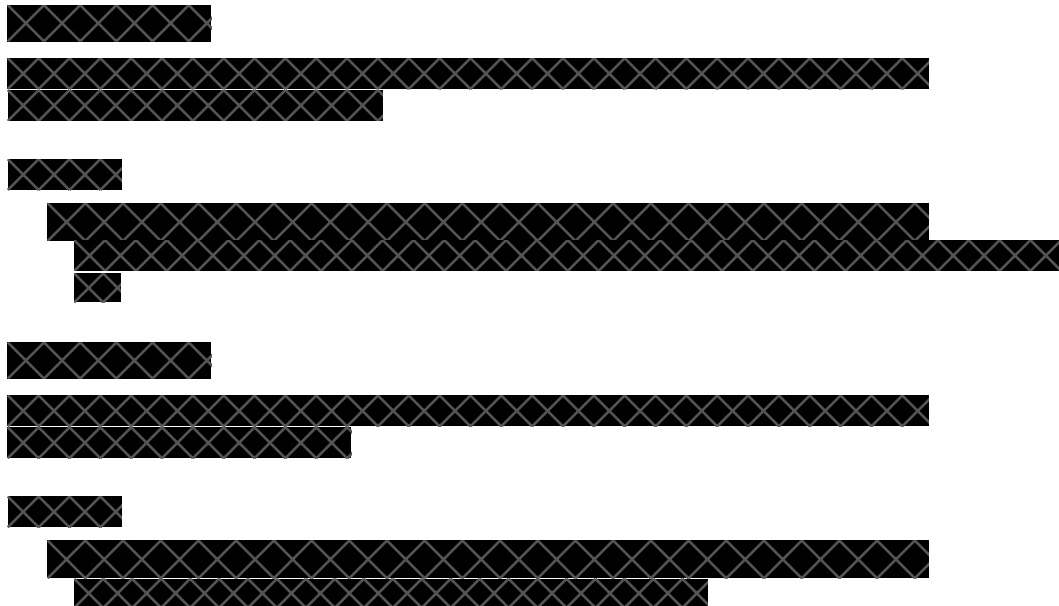
2.5.1 Rules

No denial of service Do not launch denial of service attacks. This includes DoS attacks that are aimed at overloading the web server, hogging memory, CPU, disk space and others that make the system unusable. If you plan to use automated tools for finding vulnerabilities, limit the tool to at most **one request per second**. In case the tool does not have a rate-limiting option, you are not allowed to use it. (Make sure you absolutely understand how the exploit works if you have used some tool. We recommend not using tools at least for the first three challenges).

Infrastructure is absolutely off limits! You are not allowed to hack our infrastructure. In case you accidentally find a vulnerability of which you think it is not supposed to be there, contact us immediately. In case you break something while working on the project, contact us immediately. Use .

Play nice and fair Do not interfere with other players. Do not mess with their processes, files or their work in general. Do not spoil the fun for others by publishing passwords or exploits.

Clean up after yourself Please do not leave any directories, files or processes around after you are done playing. Forgotten files and directories will eat up disk space needed by other players. Forgotten processes will eat up CPU and memory, also needed by other players.



3.9 Web challenges

To access the web challenges, navigate to <https://dss-lab.edu.distrinet-research.be/> using your browser.

You will be redirected to a ‘verification’ page. This page exists such that no random scanner on the Internet can access the website and therefore the vulnerabilities. **This verification page is not part of the assignment**, you just have to pass it to reach the assignment website.

Enter and submit the password: 7COGMBEWEO

Welcome, now you have reached the web assignment! This small application is vulnerable to SQL injections at six locations. Take a first look around in the information listed on the pages, you might find some information that will be useful ;). If you at any point find ‘user’ information (all data is fake) in the application, you are free to use it at all places you deem interesting. In other words, you are allowed to try to log in as any user in the application.

All vulnerabilities are located within **SELECT** statements. **Do not** (even when you think you can) try to execute **UPDATE**, **DELETE**, ... queries. It is never necessary to batch queries (e.g. **query1;query2;query3**) to come to a working exploit, so don’t try to do it. If you think you need batched queries, try another approach. You are allowed to execute other operators such as **LIKE**, **BETWEEN**, ... if you think you need them for your exploit.

Each of the vulnerabilities will undoubtedly have many solutions. You are free to discuss alternatives in the report, but you don’t have to. That being said, **each solution you submit should be unique and exploit a different vulnerability (i.e. each exploit should use a different type of injection: don’t use a basic payload/UNION SELECT/... twice in case you use it)**. If you find a payload that works for multiple vulnerabilities, you will have to find another way of exploiting one of both before submitting. In case you do submit the same exploit for multiple vulnerabilities, your exploit will only be counted once and the **other submissions will count as invalid**.

For every vulnerability, an exploit can be crafted through the browser, additional tools like Burpsuite are not necessary, but may be helpful. All of the forms in the application are written using GET requests, which is absolutely not good practice but this makes it easier to interact with the site through the browser.

3.9.1 Submission

You will implement each of your exploits using `curl`. If you correctly exploit a level, a green banner will be shown that displays a flag of the format: `DSS{[a-z0-9]{40}}`. Your goal is to retrieve those flags. (The flags are also reflected in a HTTP header value, that makes it easy to automatically extract them).

For each exploit you implement `exploit-sqli-<n>.sh`, where `<n>` is obviously replaced by the number (1-6) of the exploit (it does not matter which vulnerability you exploit in which script, the vulnerabilities are not numbered, as long as each script exploits a different vulnerability it's fine). Use the provided template (in `exploits.tar.gz` on Toledo) to implement your solution and verify that it prints the flag **and only the flag** (that means, not a request or a bunch of headers that contain the flag somewhere) when you run it. The only characters printed should be part of the flag. The `SITE` and `DSS_VERIFICATION_PASSWORD` variables should be used, **not** hardcoded values. Make sure you include all exploit script in your tarball (this is not part of the automatic check!). Each of your scripts will have to include a first request to pass the dss-verification, this request is already in the template.

To understand the options used in the `curl` or `grep` commands in the template, refer to their respective man-pages. Specifically make sure to look into `-s`, `-L`, `-I`, `--cookie` and `--cookie-jar` for `curl` and `-o` and `-P` for `grep`. Many more options might be useful, use your favorite search engine if you need to achieve a specific goal. Do not use custom tools in your exploits, limit yourself to `curl` and `grep`, otherwise we might not be able to evaluate your exploits.

Before executing multiple `curl` commands together, it makes sense to manually verify which response curl receives. Also note that **some of your payloads will have to be url-encoded to work** (the browser does this automatically when you submit a form or enter a url, but curl does not). You are allowed to submit your solutions with all of your payloads completely url-encoded.

You are required to hand in an exploit for 3 different vulnerabilities, you can choose which ones. Of course, if you have the time, you are encouraged to hand in more exploits for the remaining three vulnerabilities.

3.9.2 Resources

There are excellent resources on SQL injections out there:

- <https://portswigger.net/burp> - A tool to inspect and alter requests and responses, if you want to use 'a' tool, this would be where to start
- <https://getfoxyproxy.org/> - A browser plugin that can redirect traffic through Burpsuite
- <https://portswigger.net/web-security/sql-injection> - Labs where you can read about and practice exploiting SQL injections

- <https://book.hacktricks.xyz/pentesting-web/sql-injection> - More information about SQLi than you probably wanted
- https://owasp.org/www-community/attacks/SQL_Injection - A description of SQL injection with examples