The CBC padding oracle attack

Carsten Baum & Tyge Tiessen

1 Background

In order to use block ciphers securely to encrypt data that is not exactly the size of a block one needs to use a mode of encryption. For this, one first needs to use a padding scheme to extend the data to a multiple of the block size and then followed by applying a block cipher in the chosen mode. A common choice for the encryption mode used to be cipher block chaining (CBC) mode. However, as it turns out, if one is not very careful in how one implements the decryption and removal of the padding, it can open the door to a serious side-channel-attack [Vau02]. This vulnerability has lead to serious security issues even years after its original discovery [AP13].

In this project, you will implement a proof-of-concept of this attack.

2 Tasks

1. Find, read and understand a good introduction to the CBC padding oracle attack. Describe the how the attack works.

2. Implement a basic padding oracle attack as a plaintext recovery attack assuming you have access to a padding oracle. The oracle will in this scenario tell the attacker for any given ciphertext whether it decrypts to a correct padding.

3. Implement another attack that uses timing information as a padding oracle: In this scenario, when encrypting a message, we first calculate a MAC on the message, append the resulting tag, pad the message and then encrypt with CBC. Write the receiving end such that:

   - when the padding check fails, an error message is sent immediately,
   - when the padding check passes, the MAC is verified and if that fails, an error message is sent.

The resulting timing difference can be used as a padding oracle. For this, you should have methods that emulate the operations of the client and the server. The communication could either be implemented via multithreading in the same process, or by implementing multiple processes that communicate via the local loopback interface.

4. Attempt to determine the influence of the noise introduced by a network on the attack, for example by running the attack on two different machines on the same network or by simulating a network locally.

3 Deliverables

Your hand-in should consist of the following two deliverables:

1. A report addressing the tasks listed above.

2. A .zip-file containing the code which you wrote, a README file as well as the benchmarks.

4 General Information (same for each project)

Both reports and implementations are part of the assessment and therefore graded. The report must be self-contained and assume that the reader has no familiarity with the project tasks being required for this specific project. You thus need to ensure to introduce and describe the concepts used in the project with the required degree of detail and accuracy. You can expect though that the reader has knowledge similar to the course requirements of this course.

Your report must be **at most 12 pages long**, excluding references and project contributions.

Individual contributions. As we need to be able to give grades on an individual basis, it is a requirement that you indicate individual contributions in both report and code. For the report, you can add a short section with a table denoting which student has worked on which parts of the report (and possibly the project in general). Add this section to the end of your report, it will not be counted toward the page limit.

For the code, we recommend to use Git to track individual contributions. If you use Git, remember to submit your code as a Git repository.

Assessment. We will assess the report and code based on rate of completion of the tasks, correctness and appropriateness of the proposed methods/solutions, quality of writing and solution, and adherence to the project requirements.

All project tasks except bonus tasks *must* be completed to receive the highest grade. To achieve a high grade make sure that you cover the tasks in sufficient clarity and detail, and reflect on any issues and possible solutions that you encounter.

Incorrect solutions, inaccurate (or very superficial) reports as well as non-functioning or undocumented code (or code with unintelligible documentation) will lower the grade.

Suggested structure. We recommend that your report has an introduction, followed by necessary background and definitions, and then description of your project and project outcome covering the individual tasks, possibly concluding with a short discussion. Please also do not forget to add references to your report to provide your sources and justify non-trivial claims that you make.

Code submission. If the deliverables contain a programming assignment, then your code must compile and run on a machine running Ubuntu 24.04.2 LTS. Please also strive to adhere to standard practices in software engineering and structure your code in a readable way. Do not forget to properly document your code, so that readers can understand your software architecture and decisions for optimizations.

Please also deliver a README file that has the following three sections:

Compilation and Installation: What are the prerequisites (installed libraries and versions etc.) to compile and run your code, which commands should be used to compile the code, how is the code installed?

Running the test cases: Describe which commands should be used to re-run your test cases to establish that your code is correct. How should the outputs of your tests be interpreted by a reader?

Running benchmarks: If a task requires to benchmark the performance of an implementation, then describe which commands should be used to re-run your benchmarks. How should the outputs of the benchmark scripts be interpreted by a reader?

Please test if the instructions in the README file work in the described environment before handing in.

AI Usage. While we encourage the use of artificial intelligence tools to improve your learning experience, the use of such tools to write your report or code must be clearly indicated. If you fail to do this, the corresponding part will be considered plagiarism. For more information on referencing when using AI, see `https://www.bibliotek.dtu.dk/en/publishing/reference-management/kunstig-intelligens`.

References

[AP13] Nadhem J. AlFardan and Kenneth G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 526–540. IEEE Computer Society, 2013.

[Vau02] Serge Vaudenay. Security flaws induced by CBC padding - applications to ssl, ipsec, WTLS ... In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*, pages 534–546. Springer, 2002.