Lab 5 – Side-Channel Attacks
Due Date: 12/08/2025 11:59 pm
Upload Submissions to Canvas
Update for Recorded Demo at the last page.

Description:

Security Engineering boils down to identifying vulnerabilities in technology and understanding how to exploit them. While encryption or randomness provide robustness to security designs, it is always possible to find tiny cracks that can unintentionally reveal information. Timing side-channels for example, enable an attacker to infer secret data (conveniently byte-by-byte in this lab) by carefully measuring latency and using statistical analysis, often without needing any form of privileged access or specialized hardware. This lab demonstrates timing side-channels by implementing an intentionally vulnerable password check which an attacker can recover by repeated measurements of latency.

Learning Outcomes:

By the end of this lab you will understand:

- How small timing differences in code (early-exit comparisons) can leak secret information.
- Implementing an automated side-channel attack on a vulnerable server using timing analysis.
- Implementing and verifying defenses for the server and their respective effectiveness

What's needed:

- Raspberry Pi 5 with Python 3
- attacker.py
- server_v(1|2).py

TODO (for Lab report): Essentially, you'll be running two python files that communicate over a Unix socket. You might think that the Pi is irrelevant apart from the final demo and you could simply test these codes over your WSL/default Linux machine. However there's an issue with this approach that has to deal with the processing power of these different devices and how your attacking code handles the jitter induced by process scheduling. Explain this issue in your lab report and how it informs you about the significance of understanding your hardware before attacking it.

Starter Code:

You are provided with three Python files in your Lab 5 folder:

- server_v1.py – A server code that communicates with a client over a unix socket and checks the secure password entered by the client using byte-by-byte comparison.
- attacker.py – Skeleton code in which you implement a side-channel timing analysis attack on the server
- server_v2.py – Same server code as the previous one in which you implement safeguards to mitigate side-channel vulnerabilities.

Part 1 – attacker.py TODOs (25 points)

You will implement two functions in attacker.py:
Implement the measure function such that:

- It creates a unix socket, sends candidate bytes to the server, record the timing measurements of these different bytes over multiple trials and returns the average round-trip time for the candidate.
- OS process scheduling is weird and there are always high chances of outliers. You might need helper functions to make your mean more robust.

Implement the recover function such that:

- Builds the recovered bytes (b””) for each position of the password by analyzing the time for each character using the measure function above.
- It might be helpful to print measurements and password recovery progress for debugging.

Deliverable: measure(), recover(), any other functions for statistical analysis.

Part 2 – server_v2.py (15 points)

Part 2 is a lot more open-ended. Your goal is to design and implement three different defenses for the vulnerable password checker so that attacker.py no longer recovers the secret reliably. For each defense implemented in server_v2.py, evaluate its effectiveness (ideally they should all stop attacker.py completely), and discuss pros and cons (if any) for each approach. While it is not necessary to add any additional measures apart from what stops attacker.py from functioning, are there still any possibilities for side channel attacks? How can those be mitigated?

Deliverable: Better server_v2.py.

Part 3 – Lab Report (30 points)

Required Sections:

- Summary: Short description of objective and key results.
- Environment and Hardware: Note about hardware differences: Pi vs PC on scheduling/jitter differences, how it could affect measurements. Additional Linux devices used apart from Pi (Optional).
- Design & Implementation:
  - server_v1.py baseline functionality (what makes it vulnerable).
  - attacker.py: Brief description of implemented measure() and recover(), any helper stats functions. Is it perfect in gaining all forms of passwords against server_v1.py?
  - server_v2.py — each mitigation, how it was implemented
- Experiments:
  - Baseline: Attach a screenshot where attacker.py succeeds in gaining the provided password (max length = 7) with 120 trials.
  - Trial sweep: Plot showing success rate with various amounts of trials. Note: process scheduling is random at times and you might not get the plot that you consider accurate. Apart from the plot, also include a few lines talking about what your expectations were with the plot and whether you got what you expected. If not, why?
- Analysis:
  - Pros and cons of different mitigation strategies in server_v2.py.
  - Can attacker.py be improved further? Are there still possibilities for other forms of side-channel attacks? How can those be mitigated?
- Conclusion: What did you learn?

Part 4 – Recorded Demo (30 points)

Instructions:

- Make a video showing your code and how it runs (Your camera needs to be turned on and both team members should be clearly visible in the video along with a screenshare of your terminal). If you have any issues with recording a video, please email me @ samanvayupad@umass.edu.
- After showing the successful extraction of password from server_v1, member 1 needs to explain how the attacker.py successfully captures the password. Member 1 should also explain the context of process scheduling while explaining why you chose a certain number of trials in attacker.py
- After showing how attacker.py works on server_v2.py, member 2 needs to explain what the mitigation strategies are and how they protect the server against the attacker. Member2 should also explain the possibilities of how server_v2 can still be exploited in other forms.

Alternative option:

- If you have completed your lab 5 by your lab 4 demo, feel free to just do your lab 5 demo alongside lab 4 instead of recording.

What to submit

1) attacker.py (with TODOs filled): 25 points
2) server_v2.py: 15 points
3) Lab report: 30 points
4) Recorded demo (needs to be submitted by 12/08 too): 30 points