CS161 Fall 2026 — Introduction to Computer Security — Homework 6

Complete the following practice isolation question at by the end of the day on Friday, November 20. Please submit it to Gradescope once you are finished.

After the deadline, we'll release the solutions and a self-response form. Review the solutions, then submit the form to evaluate how you did.

Your grade is based solely on completing both the practice question and the self-response form, regardless of whether your answer is correct. The point of this homework is to prepare is to help you prepare for the final, so please don't use AI!

Q1 *Isolation: Blip Functions™* **(6 points)**

CodaBot is evaluating Blip Functions™, a platform where customers upload code that is run on shared servers on demand.

Design 1: The server is a single process. Each customer's code runs as its own thread inside that process. While running, each customer's code keeps its customer's request data on the single shared process heap.

Q1.1 (1 point) Select all that are true.

- ■ One customer's code can read another customer's request data.
- ■ One customer's code can modify another customer's request data.
- ■ Each customer's code can prevent another customer's code from completing.
- ○ None of the above

Solution:
- Threads of one process share a single address space, so a function can simply dereference a pointer into another function's data. Non-leakage fails (A is true, C is false).
- A function can call `exit()`, crash the process (e.g., by dereferencing NULL), or loop forever, which kills or starves every other function in the process (B is true). This breaks non-interference.

Design 2: Each invocation now runs in its own Linux process under a unique, non-root UID. Nothing else changes. Each function writes scratch files to a shared `/tmp` directory. Files are readable by everyone but writable only by their owner, and customers can't delete each other's files.

Q1.2 (1 point) Select all that are true.

- ■ One customer's code can read another customer's scratch files.
- □ One customer's code can modify another customer's scratch files.
- ■ One customer's code can slow down other customers' code.
- ○ None of the above

This content is protected and may not be shared, uploaded, or distributed.

Solution:
- Only the owner has write permission, and the sticky bit stops other users from deleting or renaming the files. Modification is not possible.
- All processes share the same CPU, memory, and disk. A function that spins on the CPU or allocates heavily slows everyone else, which breaks non-interference.

Q1.3 (1 point) Blip changes each code's scratch space to a private directory that only that code's customer's can access. Which of the following is most easily exploited?

- ◯ Customers can read other users' scratch files.
- ◯ Customers can modify other users' scratch files.
- ◯ Customers can read other users' process memory.
- ● Customers can exhaust shared resources.

Solution: File permissions fix A and B. Different UIDs also stop one process from using `ptrace` on another or reading its memory, so C is not easy. Permissions don't isolate resources, though. A function can fill the disk or fork-bomb the machine, so D is correct. Resource limits (quotas, cgroups) are needed to address this.

Design 3: Blip moves each customer's code into its own virtual machine. All VMs run on the same physical host, managed by a hypervisor.

Q1.4 (1 point) Select all that are true.

- ■ A bug in the hypervisor could break isolation between customers.
- ☐ A customer must trust the guest OS in other customers' VMs for their own data to stay private.
- ☐ Two VMs cannot learn anything about each other through shared hardware.
- ◯ None of the above

Solution:
- The hypervisor is part of the TCB: it enforces isolation, so a bug in it can compromise everyone (A is true).
- Other customers' guest kernels are not in your TCB. Even a fully compromised neighbor guest is still confined by the hypervisor (B is false).
- VMs share a physical CPU, caches, and so on, so timing side channels can still leak information (C is false).

Share links: A customer can generate an unguessable share link for their function. Anyone who has the link can invoke the function. The platform checks only the link, not who is calling.

This content is protected and may not be shared, uploaded, or distributed.

Q1.5 (1 point) Select all that apply.

- ● Capability-based access control is in place.
- ○ Mandatory Access Control is in place.
- ● Discretionary Access Control is in place.
- ○ Roles-based Access Control is in place.

Solution:
- The unguessable link is itself the access token: whoever holds it can use it, with no identity check. That is a capability.
- The function's owner decides who gets the link, which is discretionary. Under MAC, a central policy the owner can't override would decide. Under RBAC, access would depend on the caller's assigned role. Neither applies here.

Q1.6 (1 point) In 10 words or fewer, give one security drawback of share links compared to a per-user access list.

Leaked link gives access to anyone; hard to revoke per-person.

Solution: Possession is the only check. If the link leaks (browser history, logs, a forwarded message), the attacker has the same access as the intended user. The owner also can't revoke one person's access without invalidating the link for everyone. Other acceptable answers: no auditing of who actually used it, or no link between access and identity.

This content is protected and may not be shared, uploaded, or distributed.