CS161 Fall 2026 — Introduction to Computer Security — Discussion 10

Q1 *Isolation Potpourri* **(3 points)**

Q1.1 (1 point) True or false: In virtual memory, two processes can use the same virtual address and refer to different physical memory locations.

Solution: True, the same VM across various applications will be translated to different phyical memory locations by the OS.

Q1.2 (1 point) True or false: Software interposition checks while running the code while emulation rewrites code before it runs.

Solution: False, it is the opposite, where software interposition rewrites the code before it runs to be able to add checks to unsafe code. Emulation happens during execution to prevent users from taking advantage of unsafe code.

Q1.3 (1 point) Which security principle is best represented by OpenSSH's isolation design of using the Monitor Box, Authentication Box, and User Box?

Solution:

Least privilege: Each component is only able to access partial information, just enough to complete it own functions.

Separation of Responsibility: No single component is able to allow a user to login.

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

Q2 *Plug It In* **(4 points)**

PlugIn is a company whose products allow customers to upload third-party plugins (compiled to WebAssembly) that run inside a single long-lived server process. For example, each plugin handles a different user's data, Plugin A handles User A (Alice), Plugin B handles User B (Bob), and so on.

Assume the system has complete OS-level isolation between the PlugIn server process and everything else.

Q2.1 (1 point) Plugin A and Plugin B both run inside the same OS process and share the same virtual address space.How does WASM prevent Plugin A from reading Plugin B's memory?

Solution: Each Wasm module gets a contiguous 2^{32} byte slice of the overall 2^{64} bytes of the process. By construction, each module can only run within its own memory space, and there are pointer and instruction checks that ensure this. Each pointer within the module are only 32 bits, so each address is confined to the section it is allocated.

Q2.2 (1 point) Mallory, a malicious user, noticed that Bob (another user) has a bank account access function that lives at a known offset in their module's memory block region. Mallory attempts to write Wasm bytecode that computes that address and jump directly into it mid-execution. Why does this attack fail?

Solution: Wasm has no arbitrary jump instruction in the first place which blocks all attempts to change the control flow arbitrarily. The only valid jumps that can be made are to pre-defined targets (functions or labels) within Mallory's own module. If any jump points outside Mallory's declared targets, compilation from Wasm to x86 fails entirely and the bytecode never runs.

Q2.3 (1 point) Mallory found that the previous attack didn't work, so instead she attemps to write bytecode with only valid jumps within her own module, passing compilation successfully. Next, during execution, Mallory attempts the following:

- Mallory first writes a carefully crafted sequence of raw bytes into her own memory region that happen to encode a raw x86 jmp instruction pointing into Bob's bank account access function
- Mallory then attempts to overwrite her own compiled x86 function call with a pointer to those bytes, so that the next time that function is called, it will jump into Bob's module.

Why does this attack fail?

Solution: This attack fails because once compiled, the x86 code becomes immutable as it lives **outside** the contiguous 2^{32} byte chunk of memory at a 64-bit offset that Mallory's pointers cannot reach, since Mallory can only form 32-bit pointers within the range $[0, 2^{32} - 1]$, so the overwrite never succeeds.

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

Q2.4 (1 point) Assume the only 2 users are Mallory and Bob, both have access to the function `get_time()` which can read the server's clock, and each call allocates 512 MB of memory. Mallory is still a valid user, how can Mallory exploit this feature?

Solution: Mallory can call this function repeatedly and when malloc fails, the server is out of memory. So Mallory can compute:
`Total server memory - Amount allocated for Mallory = Amount allocated for Bob`

This is due to resource limitations and the distinction between virtual and physical memory. While Mallory cannot address Bob's memory directly, all virtual address windows ultimately map down to the same physical RAM on the server, which is a hard ceiling shared by every user.

So while Bob's virtual address window is completely untouched and his data is never accessed, Mallory is still able to infer the amount of physical memory that Bob holds.

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