RUTGERS UNIVERSITY

Department of Computer Science

Computer Security

Exam 2

March 25, 2024

Discussion

---

100 POINTS – 25 QUESTIONS – 4 POINTS EACH – For each statement, select the *most* appropriate answer.

1. Why might a company configure a device to use biometric authentication with a higher FAR (false acceptance rate)?
   (a) So that it will be easier to add multifactor authentication.
   (b) For improved security, since that also implies a higher FRR (false rejection rate).
   (c) To comply with international biometric data collection regulations that mandate higher FARs.
   (d) To provide a better user experience.

A higher FAR means the system is more lenient in matching biometric inputs, which reduces the frequency of false rejections (False Rejection Rate, or FRR). This can make the system faster and more convenient for users, as they are less likely to experience the need to repeatedly attempt authentication.

2. Which of the following is an example of a behavioral biometric?
   (a) Iris pattern.
   (b) Signature dynamics.
   (c) Facial recognition.
   (d) Palm print.

The iris, facial geometry, and palm patterns are invariant with your mental state while a signature can change, which makes it a behavioral biometric (voice too).

3. How does biometric authentication differ from other authentication systems, such as passkeys?
   (a) Biometric authentication is much more secure than passkeys or passwords.
   (b) Biometric authentication is a form of two-factor authentication, providing more robust security.
   (c) Biometric authentication uses thresholds to allow for variations in data.
   (d) Biometric authentication can be performed faster and more efficiently than using either passkeys or passwords.

Biometric authentication requires physical measurements, which will always have some margin of error. Even DNA testing is not 100% accurate.
(a) It's not necessarily more secure: it depends on the biometric and on what it's compared with. A fingerprint with 25 minutiae points conveys about 82 bits of information, which is equivalent to a 16-character random text password.

(b) Biometric authentication is just one factor, as is a password or physical key.

(d) It takes longer to perform biometric authentication since you need to perform signal processing on the data and then a fuzzy match to determine if the sampled data is close enough to the stored version.

4. How do hash pointers contribute to the *immutability* of a blockchain?
   (a) By ensuring that any change in the content of a block would invalidate the hash pointer in the next block.
   (b) By encrypting the contents of each block.
   (c) By linking blocks with a unique identifier that does not depend on the block's content.
   (d) By hashing the pointer so that attackers cannot identify the reference.

A hash pointer is a tuple containing a pointer and a hash of the object to which it points. Hence, if someone makes a change to a block, the pointer in the previous block will have to have its hash recomputed. That, in turn, changes the contents of that block, which means the hash in the hash pointer of the earlier block will have to be recomputed, and so on. As long as the head pointer can be stored securely, the integrity of the blockchain can be verified. With a decentralized blockchain such as bitcoin, there is no trusted place for a head pointer, which is why bitcoin uses proof of work. A proof of work requires tweaking the contents of a block (the proof of work field) so that the block's contents hash to a value in a certain range. This is phenomenally computationally intensive.

5. The price of Bitcoin recently increased in anticipation of the upcoming halving event. *Halving* is when:
   (a) The payment for adding a block to the chain gets cut in half.
   (b) The fee for each transaction is reduced by 50%.
   (c) The time to add a block to the blockchain is cut in half, which will double the transaction rate.
   (d) The mining difficulty will be cut in half, making adding a block to the chain more computationally efficient.

The bitcoin algorithm is designed to adjust a target hash so that a new block is added to the chain approximately every 10 minutes. Whoever successfully computes a proof of work for a block and thus adds it to the chain gets a reward in a certain amount of bitcoin. This is called mining. By design, every 210,000 blocks, the reward is cut in half until it falls to one Satoshi, which is defined as 0.00000001 bitcoin, and is the lowest denomination of bitcoin and cannot be halved. This will bring the total amount of bitcoin created to approximately 21 million.

6. Computing a *proof of work* in Bitcoin is:
   (a) Discovering a number that results in a hash smaller than some specified value.
   (b) Proving that every single transaction in a block is valid.
   (c) Computing the inverse of a hash.
   (d) A negotiation process among servers to agree on the transaction details before recording them on the blockchain.

In Bitcoin, computing a proof of work involves finding a nonce (a variable part of a block's header) that, when hashed with the rest of the block header, results in a hash output that is below a predefined target. This process is known as mining and is primarily a trial-and-error method that secures the network and validates transactions.

7. The *setuid* (set user ID) mechanism in Linux/Unix systems:
   (a) Allows a program to run with the authority of the user running the program rather than the program's owner.
   (b) Allows a program to run with the authority of the owner of the file rather than the user who is running it.
   (c) Enables users to change their login name while keeping the same underlying user ID.
   (d) Enables the operating system to generate a unique numeric user ID for a user.

The setuid (set user ID on execution) mechanism in Unix-like operating systems allows users to execute files with the permissions of the file owner rather than the permissions of the user who is running the executable. This is particularly useful for allowing users to perform specific administrative tasks without giving them full administrative privileges.

8. The first versions of Microsoft Windows did not use timer interrupts. What was the side effect of this?
   (a) A process could freely overwrite any files in the system.
   (b) The operating system could not protect a process from overwriting memory used by another process.
   (c) A process could keep other processes from running.
   (d) Race conditions between two processes could arise, leading to security vulnerabilities.

In the initial versions of Microsoft Windows, the lack of timer interrupts meant that the operating system used a cooperative multitasking model. Under this model, each process needed to yield control voluntarily to enable other processes to run. If a process did not yield (either due to a programming error or malicious design), it could monopolize the CPU, preventing other processes from running. This was a significant limitation compared to preemptive multitasking, where the operating system forcibly switches between processes at regular intervals, ensuring all processes get CPU time.

9. Which statement is *true* about *capability lists*?
   (a) A capability list is part of an Access Control List and is a set of access permissions for a specific user or group.
   (b) Capability lists define the set of computer resources that a process requires to execute correctly.
   (c) A capability list enumerates the privileged system calls that a process is permitted to invoke.
   (d) A capability list is a slice of an Access Control Matrix representing all resources a user can access.

A capability list is part of an Access Control Matrix. Capability Lists and Access Control Lists are two ways of breaking apart the matrix. Capability lists look at slice of a row: what permissions does a subject have for each object. Access control lists look at a slice of a column: a list of subjects and their permissions associated with each object.

10. In a strict implementation of the Bell-LaPadula which operation *cannot* take place?
    (a) A user with top-secret clearance writes a secret document.
    (b) A user with top-secret clearance reads a top-secret document.
    (c) A user with secret clearance writes a top-secret document.
    (d) A user with top-secret clearance reads a secret document.

The Bell-LaPadula model enforces two main rules: the "no read up" (simple security property) and the "no write down" (star property). The "no write down" rule means that a user cannot write information to a lower classification level than their own. Therefore, a user with top-secret clearance cannot write to or create a secret-level document, as this would involve writing down, which is not allowed in the model. The primary goal of Bell-LaPadula is to protect secret information at each level of secrecy to that it will not be leaked to lower levels.

11. How are access rights determined in a *Mandatory Access Control* (MAC) system?
    (a) Users determine their own access rights, subject to approval by a central IT department.
    (b) Through a centralized policy set by the system administrator, which cannot be modified by end users.
    (c) Based on the roles assigned to each user within an organization, adjusting access as users change positions.
    (d) Access rights are determined by the file or resource owner, who sets permissions for other users.

In Mandatory Access Control (MAC) systems, access rights are determined by policies that are enforced by the operating system, based on security labels and classifications. These policies are centrally defined by the system administrator and are not alterable by end users. This approach is typically used in environments where protection of information is of critical importance and requires strict control over who can access what resources based on their clearance and the classification of the information.

12. Which access control model was designed to prevent a process from overwriting "more important" files?
    (a) Bell-LaPadula.
    (b) Role-Based Access Control.
    (c) Biba.
    (d) Chinese Wall.

The Biba model is an integrity model designed to prevent data from being corrupted by lower integrity levels. It complements the Bell-LaPadula model, which focuses on confidentiality. The Biba model employs the "no write up" rule, which prevents a subject at a lower integrity level from writing to an object at a higher integrity level. This is intended to stop processes from overwriting or corrupting more important or trustworthy files, thus preserving the integrity of the data.

13. Which security model will change object access permissions based on what objects you read in the past?
    (a) Bell-LaPadula.
    (b) Biba.
    (c) Role-Based Access Control.
    (d) Chinese Wall.

The Chinese Wall security model is designed to prevent conflicts of interest by controlling access to information based on prior accesses. This model is particularly useful in environments where ensuring the impartiality of users is necessary, such as in financial institutions or consulting firms. Under this model, once a user accesses data from one set of information (such as from one company within a competitive grouping), they are prevented from accessing conflicting information (from a competitor), thus creating a "wall" that blocks access based on the user's previous interactions with data. This model dynamically changes the access permissions based on the history of objects that the user has previously accessed.

14. A *stack canary* is:
    (a) A thread that monitors the behavior of the stack, checking for overwrites of the return address.
    (b) A technique to place the stack at random memory locations each time the program is run.
    (c) Compiler-generated code that adds checks to ensure a function cannot overflow its buffers.
    (d) Data on the stack that, if overwritten, indicates that there is a risk of the return address having been overwritten.

A stack canary is a security mechanism used to detect and prevent buffer overflow attacks by having the compiler generate code to push a small integer (the canary) before the return address on the stack. During runtime, before a function returns, the canary is checked; if it has been altered, this indicates that a buffer overflow has occurred, potentially corrupting data on the stack including the return address. If the canary is altered, the program can take action, such as terminating, to prevent further damage or exploitation. This is a preventive security measure embedded by compilers to safeguard the integrity of the runtime stack.

15. A *return-to-libc* attack:
    (a) Requires modifying only the frame (base) pointer and not the return address on the stack.
    (b) Does not involve placing executable code in the buffer.
    (c) Overwrites only the return address on the stack and does not modify any buffer contents.
    (d) Exploits weaknesses in standard libraries rather than in the application.

A return-to-libc attack is a type of exploit where an attacker takes advantage of a buffer overflow vulnerability to redirect the execution flow of a program to a function within the existing standard C library (libc), rather than injecting their own malicious code. This method is particularly effective in circumventing security mechanisms like non-executable stack protections. By manipulating the stack to point to library functions and setting up appropriate parameters, the attacker can make the program execute standard library functions in a way that serves their malicious purposes, such as opening a shell.

16. In buffer overflow attacks, *unsafe functions* are those that:
    (a) Might accidentally overwrite their own stack frame.
    (b) Use local variables that are allocated on the stack.
    (c) Write data into an array without knowing the size of the array.
    (d) Contain bugs in their implementation.

In buffer overflow attacks, unsafe functions are those that fail to perform adequate bounds checking when writing data into an array. This lack of checking allows more data to be written into the buffer (array) than it can hold, which can lead to the overwrite of adjacent memory. Functions like *strcpy()* and *gets()* in C are classic examples of unsafe functions because they do not check the size of the input against the size of the array, leading to potential overflows if the input exceeds the buffer size. This vulnerability is a primary target for buffer overflow attacks.

17. *Address Space Layout Randomization* (ASLR):
    (a) Makes it difficult to inject meaningful memory locations into the stack or heap.
    (b) Makes it impossible to overflow the function's stack frame.
    (c) Prevents heap buffer overflows in addition to stack buffer overflows.
    (d) Mixes up the ordering of the stack, heap, static data, program code, and libraries in memory.

Address Space Layout Randomization (ASLR) is a security technique that adds a random variation to the base addresses of key data areas of a process, including the base of the executable and the positions of the stack, heap, and libraries, each time a program is run. This randomness makes it significantly harder for an attacker to predict where to inject malicious code, as the memory addresses change every time the application is executed. While ASLR does add randomness to the memory locations, the relative positions in memory are not mixed up. For example, the program code will still be in low memory and the stack in high memory. The primary security benefit is that ASLR complicates the task of injecting and executing code at predictable addresses, thus directly addressing option (a). Option (d) is partially correct in describing what ASLR does (but falsely implies that the ordering is scrambled), but it doesn't focus on the security implication, which is making the injection of meaningful memory locations difficult.

18. A *shadow stack*:
    (a) Is a backup copy of the main stack.
    (b) Stores only a function's parameters and local variables but not return addresses.
    (c) Stores a sequence of stack frames.
    (d) Only stores return addresses.

A shadow stack is a security feature that specifically stores copies of return addresses as a way to detect and prevent certain types of stack manipulation attacks, such as return-oriented programming (ROP) and buffer overflows. By maintaining a separate, secure list of return addresses, the integrity of return addresses can be verified before a function returns. If the return address on the main stack has been altered due to an exploit, the discrepancy with the shadow stack's copy can be detected, and appropriate security measures can be triggered. This specific focus on return addresses enhances security by guarding against unauthorized changes to control flow.

19. R*eturn Oriented Programming* (ROP) is designed to get around:
    (a) Stack canaries.
    (b) Address Space Layout Randomization (ASLR).
    (c) Data Execution Prevention (DEP).
    (d) Encrypted pointers.

Return Oriented Programming (ROP) is a sophisticated attack technique that is primarily used to circumvent Data Execution Prevention (DEP), a security feature that prevents the execution of code from the stack. It's a generalization of a *return-to-libc* attack. DEP is designed to make it difficult to execute malicious code via common exploits like buffer overflows by making the stack area non-executable: the processor will simply refuse to execute code because the no-execute (NX) bit is set for those pages of memory. ROP circumvents DEP by using pieces of code already present in a program's memory, specifically in executable sections, to perform malicious actions without needing to inject new executable code. This method leverages what is essentially legitimate code—already marked as executable—arranged in a sequence that serves the attacker's purpose, thereby bypassing DEP's restrictions.

20. An *integer overflow* in C might:
    (a) Create an overflow into adjacent memory.
    (b) Turn a positive number into a negative one.
    (c) Lose precision by truncating the least significant bits.
    (d) Cause a process to exit with an exception.

An integer overflow occurs in Java, Go, C and other languages that use fixed-size (e.g., 8-bit, 64-bit) integers when an arithmetic operation attempts to create a numeric value that is outside the range that can be represented with a given number of bits. For example, if you add 1 to the maximum value that an integer can hold (in a signed integer), it wraps around to the minimum value, effectively turning what was a positive number into a negative one due to the binary representation of integers (using two's complement notation). This behavior can lead to unexpected and erroneous program behavior if not properly handled, but it does not involve memory overflow, loss of bits (except in the conceptual sense of wrapping around), or exceptions.

21. What can replace the need for *setuid* programs on Linux?
    (a) Control groups.
    (b) Chroot.
    (c) Namespaces.
    (d) Capabilities.

In Linux, capabilities are a fine-grained access control mechanism that can replace the need for *setuid* programs. Traditionally, *setuid* programs run with elevated privileges, often as the root user, which can pose security risks if such programs are exploited. Capabilities partition the privileges traditionally associated with the superuser into distinct units, which can be independently enabled or disabled for individual programs. This approach allows for more controlled and limited granting of privileges compared to *setuid*, reducing the potential security risks by limiting the power of any single process or user.

22. The first isolation mechanism in Unix was *chroot*. What isolation did it provide?
    (a) Limited access to the file system.
    (b) Restricted access to privileged system calls.
    (c) A separate virtualized network interface.
    (d) Limitations on how much memory or CPU time a process could use.

The *chroot* command in Unix systems changes the root directory for the current running process and its children to a specified path. This effectively isolates the file system for that process because it cannot access files outside the designated directory tree. This kind of isolation helps in creating a simple sandbox environment where the process can operate safely without affecting the wider system. It does not directly restrict system calls, network interfaces, or resource usage but confines the process's view of the filesystem, preventing it from seeing or interacting with the entire file system of the host.

23. An example of a TOCTTOU vulnerability is:
    (a) Creating a query that contains a string provided by the user.
    (b) Deleting an empty file.
    (c) Creating a file with read-write permissions for everyone.
    (d) Checking that a user-provided filename does not contain a `/` character before creating the file.

TOCTTOU (Time of Check to Time of Use) vulnerabilities occur when a program checks a condition (like a security check) and uses a resource after the check, under the assumption that the condition has not changed between the time of checking and the time of use. The expected answer here would be (d), given the working of "checking … before creating." However, there isn't a legitimate TOCTTOU vulnerability here since the program acquired a name (a string) that the program is parsing before creating. No outside factors will change the value of the string.

The correct answer is (b), *deleting an empty file*. The reason for this is that systems allow you to delete a file (e.g., the *unlink* system call or the *rm* command) but do not offer to do so with an atomic "delete-only-if-empty" option. Hence, the programmer will have to first check the file to see that it's empty and then delete it. This creates a race condition where another process could write content into the file after the check of the file size, causing the program to delete a non-empty file. Like many TOCTTOU vulnerabilities, this can be remedied by forcing the operation to be atomic by locking access to the file (if the system and environment support doing so; a shell script or Java program, for example, does not).

24. *Command injection* may be possible when:
    (a) Prepared statements were used to include user-supplied parameters in a database query.
    (b) A system does not use data execution protection (DEP).
    (c) User input is used as an argument in a command.
    (d) A program doesn't check buffer limits.

Command injection vulnerabilities occur when user-controlled input is directly incorporated into a command that is executed by a system shell or interpreter. In option (c), when user input is used as an argument in a command without proper sanitization or validation, an attacker may manipulate this input to inject additional commands or modify the behavior of the original command, potentially leading to unauthorized actions, data breaches, or system compromise. This vulnerability is commonly exploited in scenarios like web application forms, where user input is passed to system commands without proper validation or escaping.

25. What is a common goal of a *pathname parsing* attack?
    (a) To cause a denial of service by creating recursive links.
    (b) To intercept and modify data in transit between the client and server.
    (c) To change access permissions on files, making them accessible to the attacker.
    (d) To gain access to files or directories that the server should not provide.

Pathname parsing attacks aim to exploit vulnerabilities in the way a server interprets and handles file paths provided by users or clients. The goal is typically to bypass access controls and gain unauthorized access to files or directories that are outside the intended scope of the application. By manipulating or crafting the file path, an attacker may attempt to navigate to sensitive system files, configuration files, or other directories that should not be accessible to the user or client. This type of attack can lead to data breaches, disclosure of sensitive information, or unauthorized modification of files.