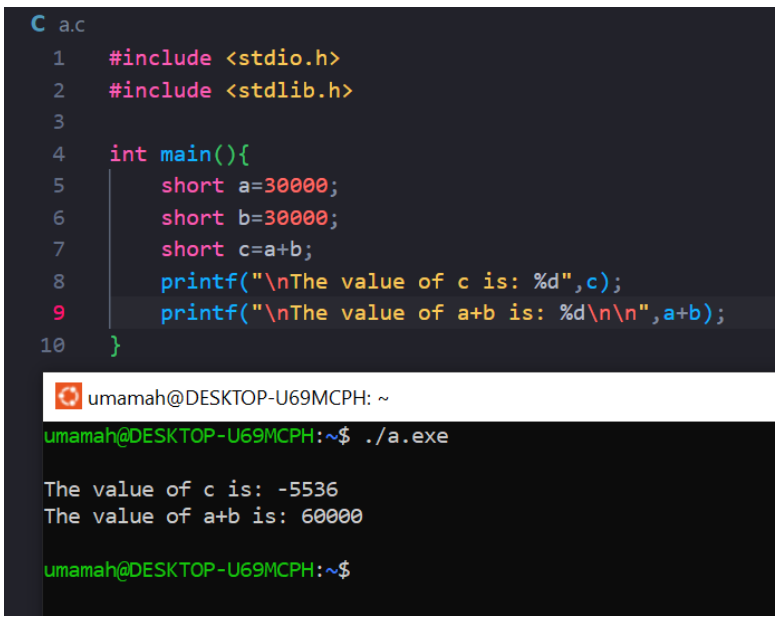
# Lecture 12




**Lecture 12: Integer Overflow and Format String Vulnerabilities**

1. **What is the goal of an attacker in control hijacking?**
   - The goal is to take over a target machine and execute arbitrary code by manipulating the application's control flow.
2. **What types of vulnerabilities are commonly exploited in control hijacking?**
   - Examples include Integer Overflow, Format String vulnerabilities, and Buffer Overflow, especially in C/C++ programs.
3. **Why are C/C++ programs more vulnerable to control hijacking?**
   - These languages provide minimal abstraction, meaning they allow direct memory manipulation, making it easier for attackers to exploit vulnerabilities.
4. **What are the main assumptions of code execution?**
   - Assumptions include: statements execute atomically, functions start and finish in sequence, only one branch of an if-statement executes, and code runs as written without external execution.
5. **What realities counter these assumptions?**
   - Machine code allows jumping to any part of a function or instruction, executing unused code, and even starting in the middle of functions, making it vulnerable to attacks.
6. **Define Integer Overflow and give an example.**
   - Integer Overflow occurs when an arithmetic operation produces a result too large for the storage space. For example:

     ```
     short x = 30000;
     short y = 30000;
     printf("%d\n", x + y); // Causes overflow in a 16-bit short.
     ```

7. **Why does integer overflow matter in programming?**
   - Overflow issues affect memory allocation, array indexing, and conditional checks, potentially enabling unauthorized access or corruption.
8. **Provide a real-world example of integer overflow.**
   - In 2004, Comair Airlines grounded 1,100 flights because their scheduling software hit a 16-bit integer limit (32,768), leading to system failure.
9. **What are countermeasures for integer overflow?**
   - Use range checking (e.g., ensuring positive sums don't turn negative) and prefer data types with larger ranges, like `long`.
10. **What is a format string vulnerability?**
    - A format string vulnerability occurs when format functions (like `printf`) are controlled by user input, potentially allowing the user to read or modify memory.
11. **How can an attacker exploit format strings?**
    - By injecting format specifiers (like `%x` for hex data or `%n` for memory addresses), attackers can print or manipulate memory content.
12. **What functions are vulnerable to format string attacks?**
    - Vulnerable functions include `printf`, `fprintf`, `sprintf`, `syslog`, `err`, and `warn`.
13. **How can format string vulnerabilities be prevented?**
    - Use compiler flags (e.g., `-Wformat`) to detect potential issues, validate input strings, and avoid user-controlled format strings.
14. **How can an attacker crash a program using string format vulnerabilities?**

    **A2:** By using a format string with multiple `%s` specifiers in `printf`, such as `printf("%s%s%s%s%s%s%s%s%s%s");`, the attacker forces the program to read memory addresses from the stack repeatedly, which can lead to a crash.

15. **Q3: Why does using multiple `%s` format specifiers lead to a crash?**
    **A3:** The `%s` format specifier tries to display memory content from an address on the stack. Since this address may contain non-mapped or invalid memory locations, the program will attempt to read from these illegal addresses, leading to a crash.
16. **Q4: What does `%s` format specifier do when used in `printf`?**
    **A4:** The `%s` specifier displays data from a memory address on the stack, interpreting it as a string until it encounters a null terminator.
17. **Q5: Why is the stack vulnerable to such an attack?**
    **A5:** The stack stores a lot of other data, including function pointers and local variables. When `%s` is used repeatedly, there's a high chance that it will attempt to read from an illegal or unmapped memory address, causing the program to crash.
18. **Q1: What is one technique attackers use in format string vulnerabilities to modify memory?**
    **A1:** Attackers can use format strings to **overwrite arbitrary memory**. For example, by using `printf("12345%n", &i);`, they can write a specific number to a variable's memory location.
19. **Q2: What does the `%n` format specifier do in `printf`?**
    **A2:** The `%n` format specifier writes the number of characters printed so far to the memory location of the variable passed to it.
20. **Q3: How can attackers use format string vulnerabilities to control access privileges?**
    **A3:** Attackers can overwrite important program flags that control access privileges, allowing unauthorized access to certain features or data.
21. **Q4: Besides controlling access privileges, what else can attackers overwrite with format string vulnerabilities?**
    **A4:** Attackers can overwrite return addresses on the stack, function pointers, and other critical memory locations, potentially altering program flow and gaining control over execution.
22. **Q5: In the example code, what value will `j` hold after executing `printf("how long is this? %n", &j);`?**
    **A5:** `j` will hold the value `18`, which represents the number of characters in the string `"how long is this? "`.
23. **Q6: What happens if `printf("how long is this? %n", msg);` is used, where `msg` points to an unknown memory location?**
    **A6:** `printf` will interpret the top of the stack as an address, and the `%n` format will write the value `18` to that address, potentially overwriting arbitrary memory.

Using this attack, attackers can do the following:

1. Overwrite important program flags that control access privileges
2. Overwrite return addresses on the stack, function pointers, etc.

---

# Lecture 13

1. **What is a buffer overflow?**
    - A buffer overflow occurs when a program writes more data to a buffer than its capacity, overwriting adjacent memory and potentially allowing an attacker to control the system.
2. **Why is C susceptible to buffer overflow?**
    - C allows direct memory access, lacks bound checking on arrays, and has raw pointers, making it easier for attackers to exploit memory vulnerabilities.
3. **Explain the concept of control hijacking via buffer overflow.**
    - When a buffer overflow overwrites memory, it can alter the program's control flow, potentially executing attacker-specified code, especially dangerous if the target process has high privileges.
4. **What are the effects of a buffer overflow?**
    - Effects include corrupted data, unexpected control flow changes, memory access violations, and execution of malicious code.
5. **How do attackers find buffer overflow vulnerabilities?**
    - Techniques include inspecting source code, tracing execution with oversized input, and using fuzzing tools.
6. **What is the significance of memory layout in buffer overflows?**
    - Understanding memory layout helps attackers predict adjacent memory positions, which they can use to alter program execution.
7. **List some functions that can lead to buffer overflow in C.**
    - Functions include `strcpy`, `strcat`, `gets`, and `scanf`. Safer versions like `strncpy` exist but may still pose risks if not used correctly.
8. **What is the role of the stack in control flow?**

- The stack manages function calls and returns; buffer overflows can overwrite return addresses, altering the control flow to malicious code.

9. **Describe the Morris Worm as a real-life example of a buffer overflow attack.**
   - The Morris Worm in 1988 exploited a buffer overflow, infecting about 60,000 machines and demonstrating the severity of such vulnerabilities.
10. **What are some buffer overflow defenses?**
    - **Compile-time defenses** include safe coding practices, using modern languages, safe libraries, stack protection (e.g., StackGuard).
    - **Run-time defenses** include non-executable memory (making stack/heap non-executable), address space randomization, and guard pages (blocking memory access).
11. **What does StackGuard do?**
    - StackGuard adds code to detect and prevent stack corruption by checking for overwritten values, stopping the program if an anomaly is found.
12. **How does address space randomization help?**
    - By randomizing key data structure locations in memory, it becomes harder for attackers to predict addresses, reducing the risk of exploitation.
13. **Explain the use of guard pages as a defense.**
    - Guard pages are memory sections flagged as illegal; any access to them aborts the process, blocking overflow attempts on critical memory regions.

# Lecture 14

**Database Security and SQL Basics**

**Q1: What is a database system?**
**A1:** A database system is a structured collection of data stored for use by one or more applications, containing relationships between data items and often holding sensitive data that needs security.

**Q2: What is the purpose of a query language in a database?**
**A2:** A query language provides a uniform interface to the database, allowing users to define schemas, manipulate data, and query data.

---

**Relational Databases**

**Q3: What is a relational database?**
**A3:** A relational database is organized as tables with rows and columns, where each row contains specific values for each column, and one column often serves as a unique identifier or key.

**Q4: How does a relational database use unique identifiers?**
**A4:** Unique identifiers in a relational database link tables together, enabling users to request data that meets certain criteria using relational query languages.

---

**Structured Query Language (SQL)**

**Q5: What is SQL, and where did it originate?**
**A5:** SQL (Structured Query Language) is a standardized language developed by IBM in the 1970s, used to define schema, manipulate data, and query databases.

**Q6: Give an example of an SQL command to fetch records.**
**A6:** `SELECT * FROM Person WHERE Username='Vitaly';` – This retrieves all records from the `Person` table where the username is 'Vitaly'.

**Q7: Provide an example of an SQL command to insert data into a table.**
**A7:** `INSERT INTO Key (Username, Key) VALUES ('Vitaly', '3611BBFF');` – This adds a new record with 'Vitaly' as the username and '3611BBFF' as the key.

---

**SQL Injection Attacks**

**Q8: What is an SQL injection attack?**
**A8:** SQL injection is a network-based security threat where an attacker sends malicious SQL commands to the database server to control application behavior, alter data, or even execute OS commands.

**Q9: How can SQL injection attacks manipulate data?**
**A9:** Attackers can modify, delete, or access data without authorization, and even manipulate application behavior, like bypassing authentication.

**Q10: Describe a typical SQL injection attack step-by-step.**
**A10:**

1. The attacker finds a vulnerability in a web application and injects malicious SQL commands through the web server.
2. The web server passes these commands to the application server.
3. The application server forwards them to the database server.
4. The database server executes the malicious code, returning sensitive data.
5. The application server dynamically generates a page with the retrieved data.
6. The web server sends this data (e.g., credit card details) back to the attacker

---

**Techniques of SQL Injection**

**Q11: What is the 'Union' SQL injection technique?**
**A11:** The 'Union' technique combines two queries into a single result set, useful for injecting malicious queries into a SELECT statement.

**Q12: Explain the 'Boolean' SQL injection technique.**
**A12:** Boolean SQL injection uses true/false conditions to test for specific conditions, allowing attackers to verify certain assumptions about the database structure.

**Q13: What is 'Error-based' SQL injection?**
**A13:** Error-based SQL injection forces the database to produce an error, which can reveal details about the database structure, helping the attacker refine their injection.

**Q14: Describe the 'Out-of-band' SQL injection technique.**
**A14:** Out-of-band SQL injection retrieves data through a separate channel, such as an HTTP request to send query results to the attacker’s server.

**Q15: How is 'Time delay' used in SQL injection attacks?**
**A15:** Time delay injections use database commands to delay responses, helping attackers infer information about query results even without explicit output.

---

**Types of SQL Injection Attacks**

**Q16: What are the three main types of SQL injection attacks?**
**A16:**

- **In-band attacks:** Extract data directly through the same channel used for injection.
- **Out-of-band attacks:** Use a separate channel for data retrieval, like email or HTTP requests.
- **Inferential (Blind) attacks:** Infer data by sending specific queries and observing the server's response.

**Q17: Explain 'Tautology' as an in-band SQL injection attack.**
**A17:** Tautology attacks inject code into a conditional statement so that it always evaluates to true, potentially bypassing authentication.

**Q18: What is an 'End-of-line comment' attack in SQL injection?**
**A18:** End-of-line comment attacks insert comments to nullify legitimate code after the injected SQL, ensuring only the malicious code executes.

**Q19: Describe 'Piggybacked queries' in SQL injection.**
**A19:** Piggybacked queries add malicious queries after a legitimate one, executing both in the same database call.

---

**Inferential or Blind SQL Injection**

**Q20: What is an inferential SQL injection attack?**
**A20:** Inferential or blind SQL injection attacks don’t directly return data but allow attackers to deduce information by observing how the server responds to different queries.

**Q21: What is a 'Blind SQL injection' attack?**
**A21:** Blind SQL injection asks the server true/false questions, inferring data based on whether the server returns an error or processes the query.

**Q22: How do illegal or logically incorrect queries help attackers in inferential attacks?**
**A22:** Illegal queries can reveal details about the database structure when the server returns detailed error messages.

---

**Application Interaction with Databases**

**Q23: What are common instances when a web application interacts with a database?**
**A23:**

- **Authentication forms**: To check user credentials against stored usernames and passwords.
- **Search engines**: To retrieve records based on user input in SQL queries.

---

**SQL Injection Countermeasures**

**Q24: What is defensive coding in SQL injection prevention?**
**A24:** Defensive coding involves validating data with type checks (ensuring inputs are digits only when expected) and pattern matching to detect abnormal input.

**Q25: Name three SQL injection detection methods.**
**A25:**

- **Signature-based detection**: Recognizes known malicious patterns.
- **Anomaly-based detection**: Identifies unusual behaviors or queries.
- **Code analysis**: Reviews source code to identify vulnerabilities.

**Q26: How does runtime prevention help counter SQL injection?**
**A26:** Runtime prevention involves checking queries at runtime to ensure they conform to expected query structures, blocking unexpected or malicious queries.

---

## Bad Input:

1. **Malicious Input**: The attacker sets the `user` input to:

   ```
   ' OR 1=1 --
   ```

   This input is URL-encoded, meaning it’s formatted to be sent over the web without causing errors in URLs.

2. **SQL Query Execution**: When this input is processed in the application, it is inserted into an SQL query in the authentication system. The resulting query might look like this:

   ```
   SELECT * FROM users WHERE username = '' OR 1=1 --
   ```

   Here, the `username` parameter is set to an empty string `''`, or the condition `1=1` is added. The `--` part is an SQL comment, which causes the rest of the line to be ignored by the SQL interpreter.

3. **Effect of `--`**: The `--` in SQL is a comment indicator. Everything after `--` is ignored by the SQL interpreter. This means any additional conditions or constraints that might normally follow the `WHERE` clause, such as password checks, are bypassed.
4. **Condition Always True**: The `OR 1=1` part of the injection always evaluates to true. Because of this, the `WHERE` clause essentially becomes ineffective in filtering rows. Instead, it selects all rows, as the condition is universally true.

5. **Login Bypass**: Since the query doesn't filter out any records, the SQL statement allows the login to proceed, regardless of whether the attacker provided a valid username or password. In essence, the system treats the injection as a successful login attempt.
6. **Security Implication**: This is a simple example of how SQL injection can allow unauthorized access. An attacker could use this technique to gain easy access to accounts without knowing any valid credentials, making it a significant security risk.

In summary, this slide illustrates a basic SQL injection where the attacker manipulates the input to bypass authentication checks by inserting an always-true condition (`1=1`) and using comments (`--`) to ignore the rest of the query. This tactic allows unauthorized access to systems that are vulnerable to such SQL injection attacks.

# Lecture 15

**Database Access Control**

**Q1: What is database access control?**
**A1:** Database access control is a security feature provided by DBMS to control user access to specific parts of a database. It assumes the user is already authenticated.

**Q2: What kinds of access rights can a DBMS grant?**
**A2:** A DBMS can grant rights such as create, insert, delete, update, read, and write, which can apply to the entire database, tables, or specific rows or columns. Access rights can also depend on table entry content, like salary information.

**Q3: What are the main policies for database access control?**
**A3:**

- **Centralized Administration**: A few privileged users grant access.
- **Ownership-based Administration**: The creator of a table can grant access.
- **Decentralized Administration**: Owners can grant or revoke rights to other users.

**SQL Access Controls**

**Q4: How does SQL manage access controls?**
**A4:** SQL uses `GRANT` and `REVOKE` commands to manage access controls, assigning privileges to users, roles, or the public.

**Q5: What is the purpose of the `WITH GRANT OPTION` in SQL?**
**A5:** `WITH GRANT OPTION` allows the grantee to further grant access rights to other users.

**Q6: Give an example SQL command to grant insert and delete rights on a table named 'Repository' to a user named 'peter'.**
**A6:**

```
GRANT INSERT, DELETE ON Repository TO peter;
```

**Cascading Authorizations**

**Q7: What is cascading authorization in SQL?**
**A7:** Cascading authorization allows an access right to pass through multiple users. If access is revoked by the original grantor, it should also be cascaded and revoked from subsequent users.

**Q8: What complication can arise with cascading authorizations?**
**A8:** A complication arises if a user receives the same access right multiple times. For instance, if a user has access from multiple sources, revoking from one source might not remove the right if granted by another source.

**Role-Based Access Control (RBAC)**

**Q9: Why is RBAC a good fit for databases?**
**A9:** RBAC is a natural fit because it aligns with the various tasks and applications users perform, each requiring specific access rights, which reduces administrative overhead and improves security.

**Q10: What are the main categories of database users in RBAC?**
**A10:**

- **Application Owner**: Owns database objects as part of an application.

- **End User**: Operates on database objects but does not own them.
- **Administrator**: Manages administrative responsibilities for part or all of the database.

**Q11: What capabilities must an RBAC system provide in a database?**
**A11:**

- Create and delete roles.
- Define permissions for each role.
- Assign and cancel user-role assignments.

**Q12: What types of roles are supported by Microsoft SQL Server?**
**A12:**

- **Server roles**: Fixed roles with specific access rights.
- **Database roles**: Fixed roles with specific access rights.
- **User-defined roles**: Customizable roles defined for specific applications.

**Inference**

**Q13: What is inference in the context of database security?**
**A13:** Inference is the process of deducing unauthorized information from authorized responses by combining data items to infer sensitive information.

**Q14: Give an example of inference in a database.**
**A14:** Suppose we have tables for `Employees` (with name and address), `Salaries`, and `Emp-Salary` (linking employee to salary). By combining information across these tables, an attacker might infer salary details for specific employees.

**Inference Countermeasures**

**Q15: How can inference be prevented at the database design stage?**
**A15:** Inference can be prevented by altering the database structure, such as splitting tables into multiple tables or implementing more fine-grained access controls.

**Q16: What is query-time inference detection?**
**A16:** Query-time inference detection analyzes queries in real-time to identify and deny any that might reveal sensitive information through inference.

**Database Encryption**

**Q17: Why is database encryption important?**
**A17:** Database encryption provides an additional layer of security for sensitive data beyond firewalls, authentication, and access control.

**Q18: What are the different levels at which data can be encrypted in a database?**
**A18:**

- **Entire Database**: Difficult to manage and search efficiently.
- **Individual Fields**: Simple but inflexible.
- **Records (rows) or Columns (attributes)**: More flexible and effective.

**Q19: Explain the role of attribute indexes in encrypted databases.**
**A19:** Attribute indexes assist in data retrieval from encrypted databases by providing an index value for each attribute, helping locate and retrieve records more efficiently.

**Q20: What are the two main approaches to database security?**
**A20:**

- **Outsource DBMS**: Use a third-party provider, with concerns about data confidentiality.
- **Encrypt the Database**: Retain control over data confidentiality but may reduce flexibility in searching and decrypting data.

**Example of Encrypted Query**

**Q21: How does an encrypted query work in a database?**
**A21:** In an encrypted query, the client transforms the query by encrypting the specific attribute values (e.g., department ID), which the server then matches to encrypted database values.

**Example**:

- Original query:

  ```
  SELECT Ename, Eid, Ephone FROM Employee WHERE Did = 15;
  ```

- Encrypted form:

  ```
  SELECT Ename, Eid, Ephone FROM Employee WHERE Did = 1000110111001110;
  ```

**Secured Operations**

**Q22: Describe the process of retrieving an encrypted record from the database.**
**A22:**

1. The user sends a query with a specific primary key.
2. The client encrypts the primary key and modifies the query.
3. The server processes the query with the encrypted key.
4. The client decrypts the returned data.

**Issues with Database Encryption**

**Q23: What is a limitation of database encryption in relation to range queries?**
**A23:** Encrypted values do not preserve the original attribute's ordering, making it difficult to perform range queries, like retrieving all salaries below a certain amount.

**Record Level Encryption and Indexing**

**Q24: How does record-level encryption work?**
**A24:** Each record (row) in a table is encrypted as a block, and attribute indexes help in retrieving data. Index values are assigned based on predefined partitions for each attribute's range.

**Q25: Give an example of indexing for record-level encryption.**
**A25:**

- For an employee ID (eid) in the range [1, 1000], values can be partitioned:
  - [1, 200] → index 1
  - [201, 400] → index 2
  - [401, 600] → index 3
  - [601, 800] → index 4
  - [801, 1000] → index 5