Data store

Data store

Data Stores in Applications

- Manage data essential for application logic.
- Store accounts, permissions, settings, etc.
- Structured data and query languages.
- Built-in logic for data management.

Common Vulnerabilities

Applications often use the same privilege for all users.

Attackers can:

- Modify or retrieve unauthorized data.
- Bypass application-layer controls.

Types of Data Stores

1. SQL databases

i. Structured data storage in tables

ii. Use SQL (Structured Query Language) to manage data.

iii. MySQL, PostgreSQL, Microsoft SQL Server

iv. Storing user accounts, online store orders, blog posts.

Example

```
SELECT * FROM Users WHERE age > 21;
```

Types of Data Stores

2. XML repositories

- Stores data in XML (eXtensible Markup Language).
- Organized using tags (similar to HTML).
- Ideal for structured, hierarchical data.
- Sharing data between applications or storing app settings.
- Many software applications use XML to store configuration settings.
- XML is commonly used for SOAP (Simple Object Access Protocol) web services to exchange data between servers.

Example

```
<user>
   <id>1</id>
   <name>John Doe</name>
   <email>johndoe@example.com</email>
</user>
```

Types of Data Stores

LDAP(Lightweight Directory Access Protocol) Specialized for storing directory data and managing **access control** for users and **resources** in a network.

Each entry has important details attached, like:

- Name
- Email
- Role in the company (e.g., employee, manager)
- Access rights (e.g., can they log in to certain systems?)

Example

```
ldapsearch -x -b "dc=example,dc=com" "(uid=john.doe)"
```

Interpreted vs. Compiled Languages

- **Interpreted languages**: These are executed using an interpreter, which translates and runs the code line by line at runtime. Examples include SQL, PHP, and Python.
- **Compiled languages**: These are converted into machine-readable code (binary) beforehand and then executed directly by the computer. Examples include C and C++.

Types of SQL Injection

1. **In-band SQLi (Classic)**
2. **Inferential SQLi (Blind)**
3. **Out-of-band SQLi**

1. In-band SQL Injection (Classic SQLi)

Most common & easiest to exploit.

Attack and results use the **same communication channel**.

Error-based SQLi:

Exploits error messages from the database to gather information (e.g., database schema, table names).

Example: If the application shows SQL errors, attackers can use malformed queries to extract sensitive information.

Union-based SQLi:

Uses the UNION SQL operator to combine results of multiple SELECT statements into a single output.

Example: A vulnerable query could expose additional data like user passwords by combining unauthorized tables.

2. Inferential SQL Injection (Blind SQLi)

No direct data returned; attacker observes application behavior.

Takes longer but can be just as dangerous.

Boolean-based SQLi:

- Sends queries that return TRUE/FALSE results.
- Application response changes based on the query result.

Example: If a query like 1=1 results in a page loading normally but 1=2 does not, attackers can deduce whether the condition is true or false.

Time-based SQLi:

- Queries cause delays to indicate TRUE/FALSE results.
- Measures response time to infer results.

Example: Injecting a command like IF(condition, SLEEP(5), 0) helps attackers infer database behavior based on response delay.

Out-of-band SQL Injection

Less common; relies on database features (e.g., DNS or HTTP requests).

Useful when:

- Same-channel attacks aren’t possible.
- Server responses are unstable for inferential attacks.

Examples:

Microsoft SQL Server: xp_dirtree for DNS requests.

Oracle: UTL_HTTP for HTTP requests.

NoSQL Databases

NoSQL databases differ from traditional relational databases by using **key/value pairs** and allowing data to be stored in **flexible, hierarchical structures** (unlike the rigid tabular format in SQL databases).

Key: user123
Value: {"name": "John", "email": "john@example.com", "orders": [101, 102, 103]}

Common examples of NoSQL databases include **MongoDB**.

Flexible, and able to handle large volumes of **unstructured** or **semi-structured data**.

Injecting into MongoDB

```
$m = new Mongo();
$db = $m->cmsdb;
$collection = $db->user;
$js = “function() {
 return this.username == ‘$username’ & this.password == ‘$password’; }”;
$obj = $collection->findOne(array(‘$where’ => $js));
if (isset($obj[“uid”]))
{
 $logged_in=1;
}
else
{
 $logged_in=0;
}
```

demonstrates a login authentication system that uses MongoDB to store user information and check the username and password during login

In JavaScript, // is used to create a single-line comment. An attacker can input a value like Marcus'// as the username.

function() { return this.username == 'Marcus' // ' & this.password == 'aaa’; }

XPath

- XPath (XML Path Language) is a query language used to navigate through eleme
  and attributes in an XML document.

```
<addressBook>
<address>
<firstName>William</firstName>
<surname>Gates</surname>
<password>MSRocks!</password>
<email>billyg@microsoft.com</email>
<ccard>5130 8190 3282 3515</ccard>
</address>
<address>
```

To retrieve all email addresses, you could use

```
//address/email/text()
```

To retrieve details of a specific user (e.g., Gates)

```
//address[surname/text()='Dawes']
```

Xpath (Subverting Application Logic)

consider an XPath query used to verify user credentials and retrieve sensitive information like a credit card number:

```
//address[surname/text()='Dawes' and password/text()='secret']/ccard/text()
```

This query checks the surname and password fields and, if they match, retrieves the user's credit card number.

If we provide this ' or 'a'='a

```
//address[surname/text()='Dawes' and password/text()=''' or 'a'='a']/ccard/text()
```

This alters the query's logic by making the password/text() condition always true ('a'='a' is always true), thus retrieving the credit card information for all users with surname dawes instead of just the targeted user.

Informed XPath Injection

Condition Testing

' or 1=1 and 'a'='a': This query will return data because the condition 1=1 is always true.

' or 1=2 and 'a'='a': This query will return nothing because 1=2 is false.

similar to boolean-based SQL injection.

Injecting into LDAP

- LDAP Search Filters

Simple match condition: Searches for a specific attribute value.

Example: (username=daf) (This searches for the username "daf")

Disjunctive query: Searches for any one of several conditions.

Example: (|(cn=searchterm)(sn=searchterm)(ou=searchterm)) (Searches for "searchterm" in multiple fields)

Conjunctive query: Requires all conditions to match.

Example: (&(username=daf)(password=secret)) (Searches for "daf" with the password "secret")

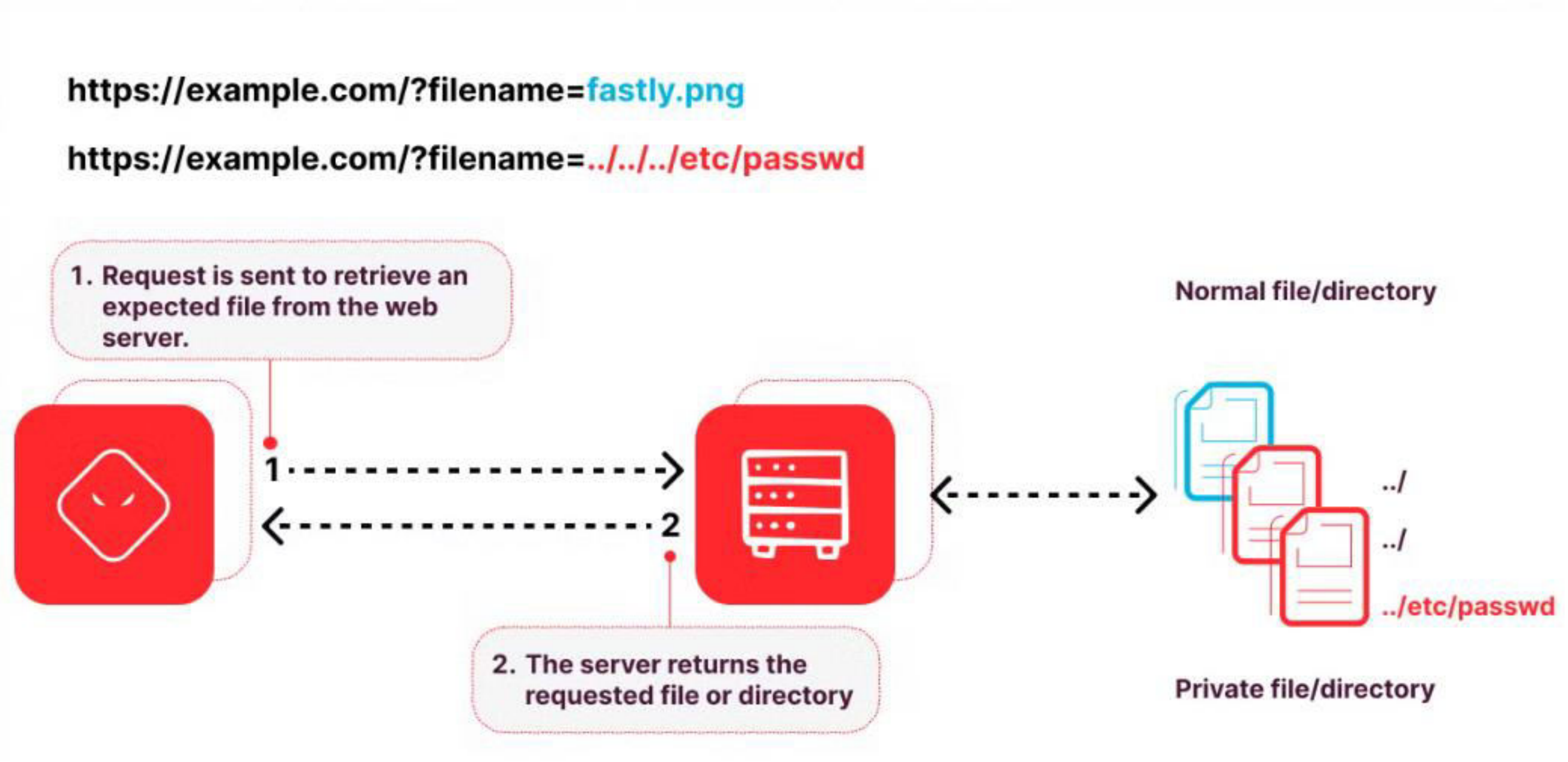
Why LDAP Injection is Less Dangerous Than SQL Injection

- **Logical Operators**: LDAP queries often use operators like AND or OR before user input, making it harder for attackers to use simple attacks like OR 1=1.
- **Fixed Data Retrieval:** The data returned is often fixed, so attackers can’t change what data is retrieved by manipulating the input.
- **Limited Error Messages:** LDAP systems rarely provide helpful error messages, making it harder for attackers to figure out if their attack is working ("blind" exploitation).

Directory Traversal Attack

- A directory traversal attack, also known as a path traversal attack, is a type of web application vulnerability that allows an attacker to access restricted directories and files stored on a web server outside the intended directory.

Directory Traversal Attack



https://example.com/?filename=fastly.png

https://example.com/?filename=../../../etc/passwd

1. Request is sent to retrieve an expected file from the web server.

2. The server returns the requested file or directory

Directory Traversal Attack

- Imagine a website that allows file downloads through a URL like:

https://example.com/download?file=report.pdf

- The application reads the file path directly from the file parameter without sanitizing the input.
- Malicious Input:The attacker modifies the input to: https://example.com/download?file=../../../../etc/passwd

Directory Traversal Attack

- Here, ../../ moves up the directory tree, allowing access to files like /etc/passwd (Linux) or C:\Windows\System32\config\SAM (Windows).

- This file (/etc/passwd) contains user account information on Linux, and exposing it can lead to further attacks.