CERTIN - PS1

TEAM NUMBER: 65

THE PROBLEM WITH LARGE NUMBER OF DEPENDENCIES AND PACKAGES

Auditing all the projects in an organization with thousands of servers proves to be problematic to deal with for the auditor and the auditee. All these applications use different libraries have different dependencies and have multiple tech stacks, and all the servers have different packages installed.

We device a solution to make this information readily available and beyond.

WE DEVIDE OUR APPLICATION INTO TWO COMPONENTS:

CODEBASE SCANNER

Keep track of libraries and dependencies used in a codebase on a per project basis.

SERVER AGENT

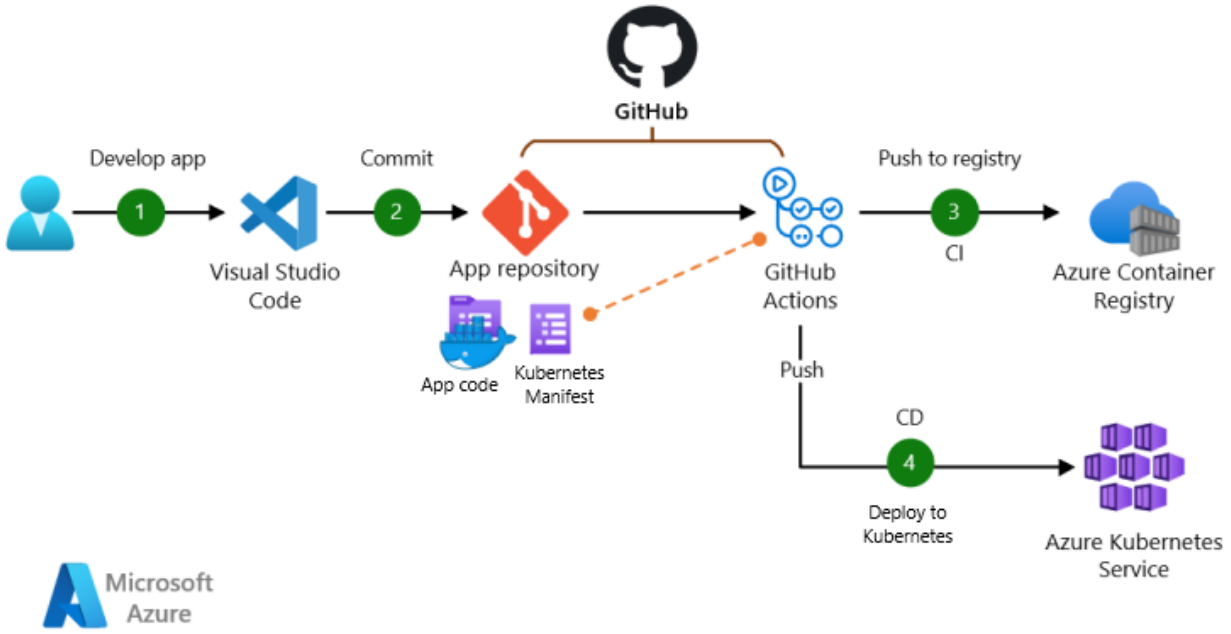
Keeps track of packages and their versions installed across different servers.

CODEBASE SCANNER

WORKING : Scanning projects for dependency files like pom.xml, package.json, requirements.txt etc. for lisiting dependencies on a per project basis.

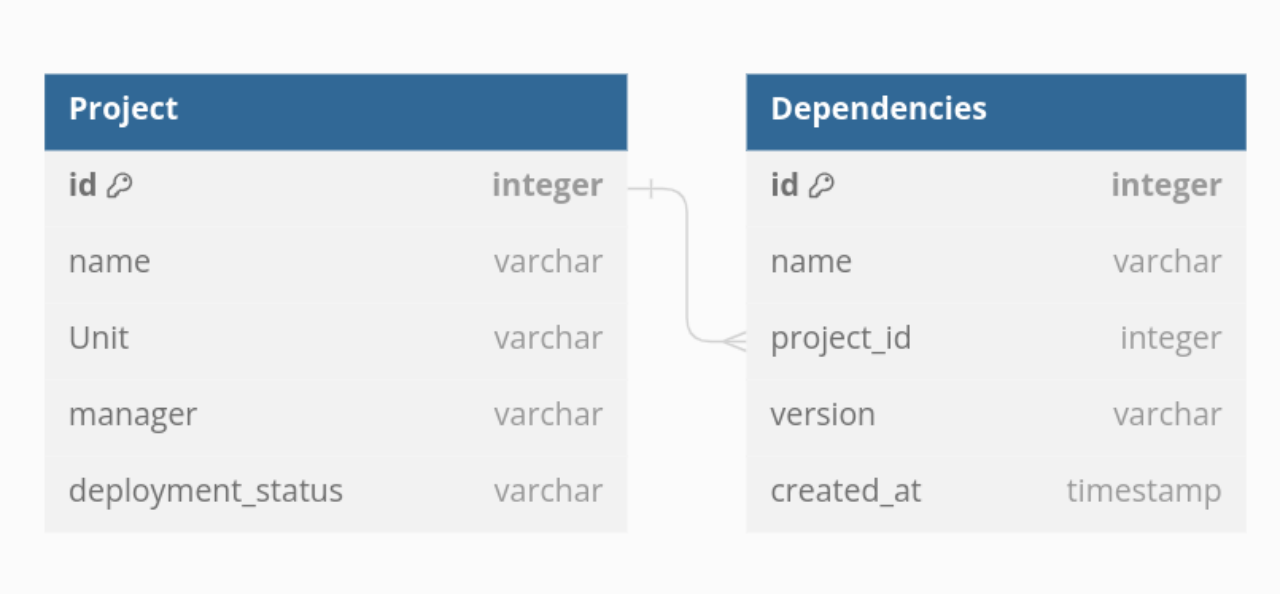
CODEBASE SCANNER

INTEGRATION : The application is integrated in the code-upload pipeline where it is triggered after a successful code push.



CODEBASE SCANNER

INFORMATION: We maintains a centralized database for all projects and maps a project and its version to a set of dependencies in the form of libraries and packages.

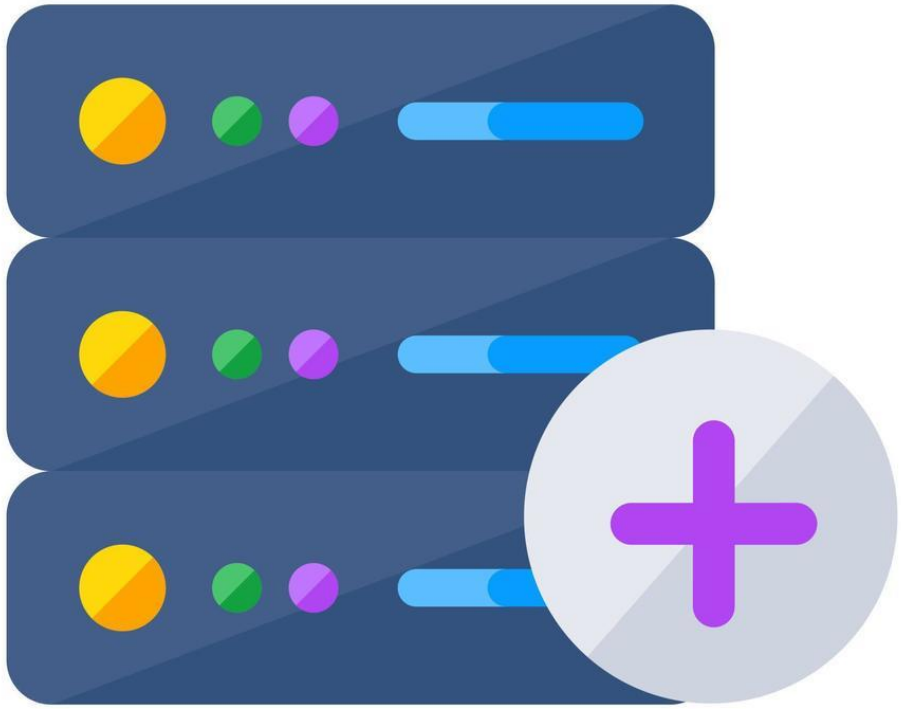


SERVER AGENT

WORKING : Periodic scan of deployment servers using package managers(like DNF, aptitude etc.) and all binaries in user path(like /bin,/sbin etc.) and update database.

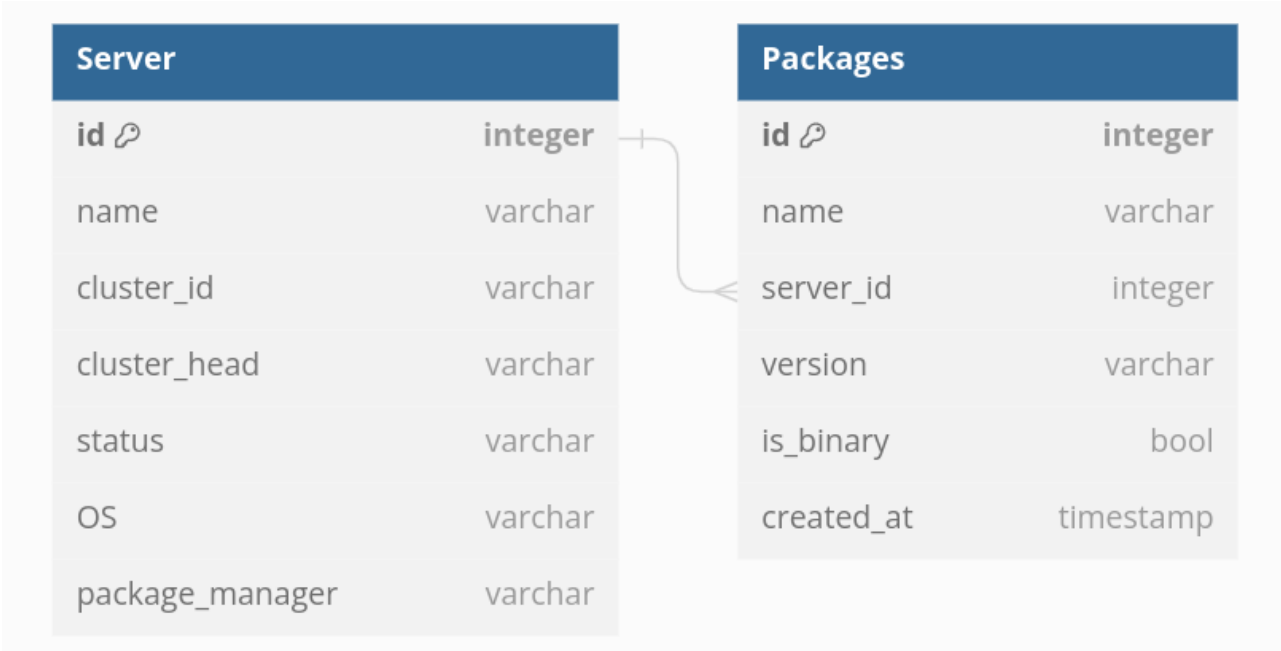
SERVER AGENT

INTEGRATION : The application is integrated in the devops server setup pipeline where the application is added as a service on every newly spawned server which runs scans periodically.



SERVER AGENT

INFORMATION: We maintain a centralized database for all servers and map them to a list of packages installed in them with their versions.



BEYOND PASSIVE DATA COLLECTION

ACTIVE THREAT SCAN

We can integrate public databases like the CVE database by MITRE for finding out vulnerable packages/dependencies in our systems.

ACTIVE THREAT SCAN

Trigger: A database trigger on updating values in our dependencies or package table can check the CVE database for known vulnerabilities in our updated entries.

We also run this periodically for all database entries.

CERTIN – PS2

TEAM NUMBER: 65

APPROACH FOR REDUCING VULNERABILITIES IN SOFTWARE APPLICATIONS.

Henceforth we will discuss a security base guideline to be followed in application development process for reducing the possibilities of any exploits by any malicious agent.

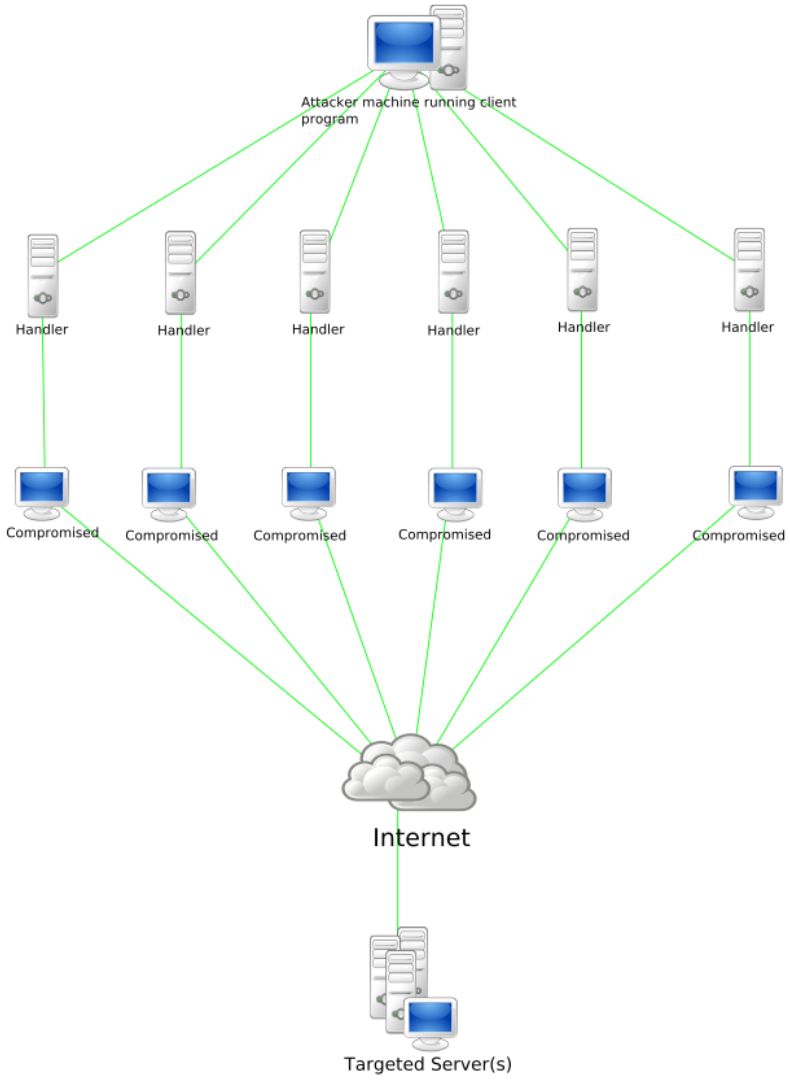
We will divide these into two categories:

- Development guidelines.
- Deployment guidelines.

DEVELOPMENT GUIDELINES

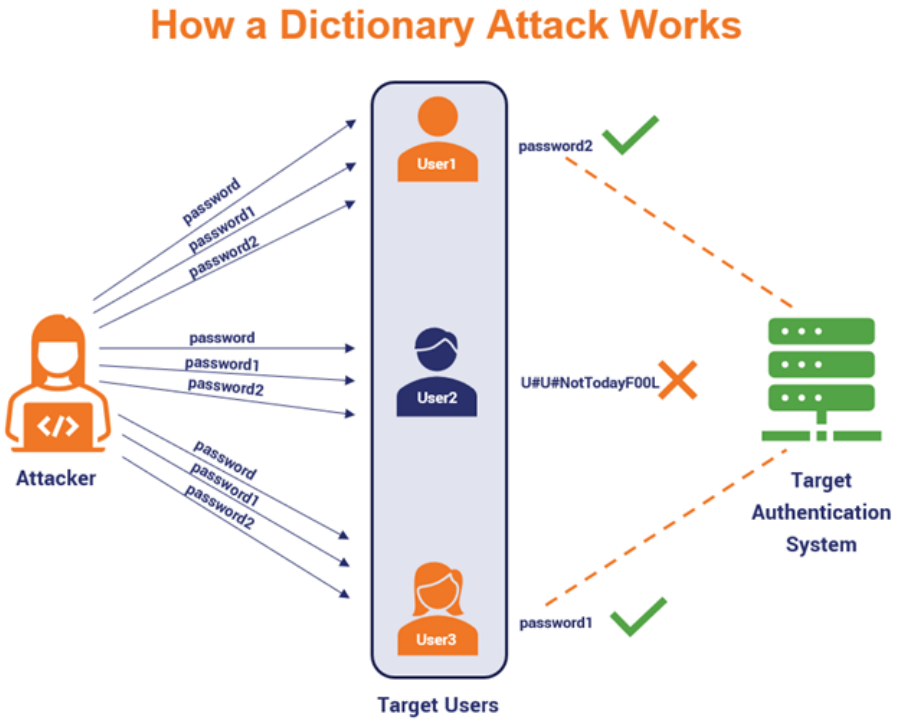
1. STATELESS SERVERS

A stateless server does not keep any user or session related data in memory. This is important to prevent servers from DoS attacks which overwhelm the server with large number of open sessions.



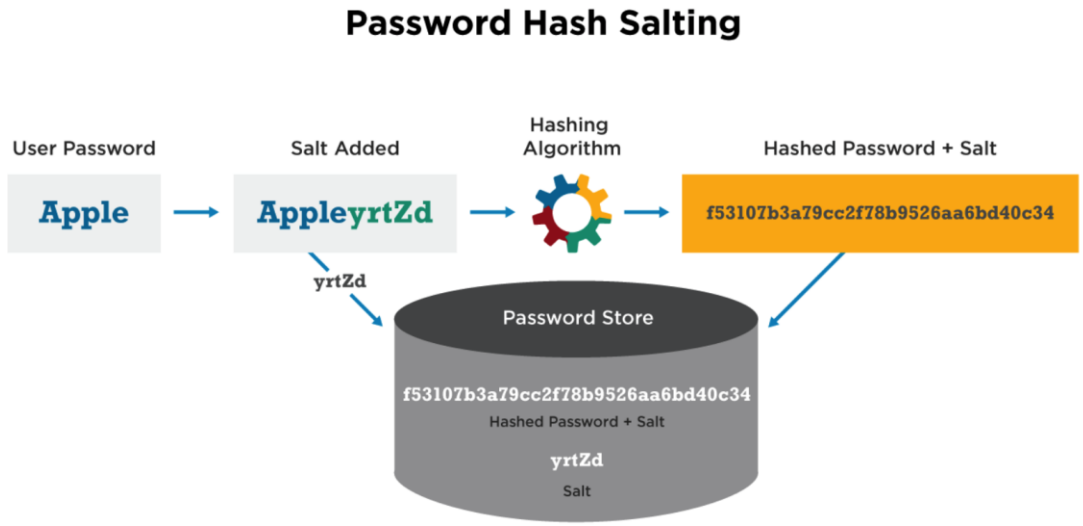
2. AUTHENTICATION RATE LIMITERS

Authentications request API calls must be limited on a per user per unit time rate limit and a per IP per unit time rate limit to prevent against brute force attacks like dictionary attacks and password spraying.



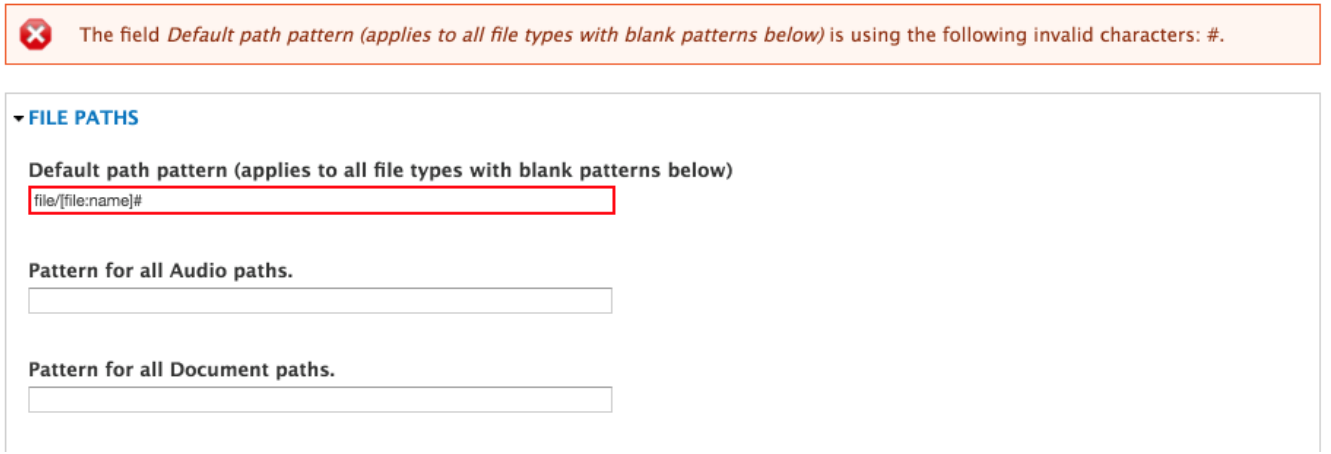
3. SECURE CREDENTIAL STORAGE

Any form of passwords stored in a database must be stored in hash form, only matching of hashes is required for authentication. This makes sure that in a data breach, your actual user's passwords don’t get leaked.



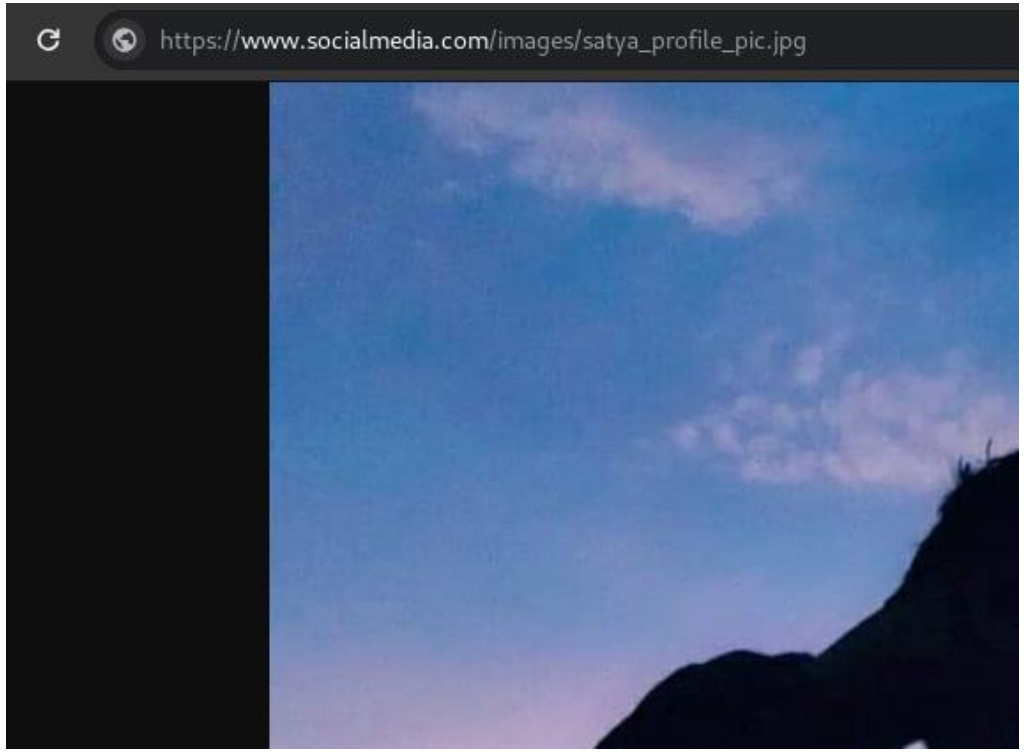
4. INPUT FIELD VALIDATION

Validating user inputs in the backend for allowed character combinations and size helps prevent several attacks like remote code execution, Local File Inclusion and Buffer Overflow vulnerabilities.



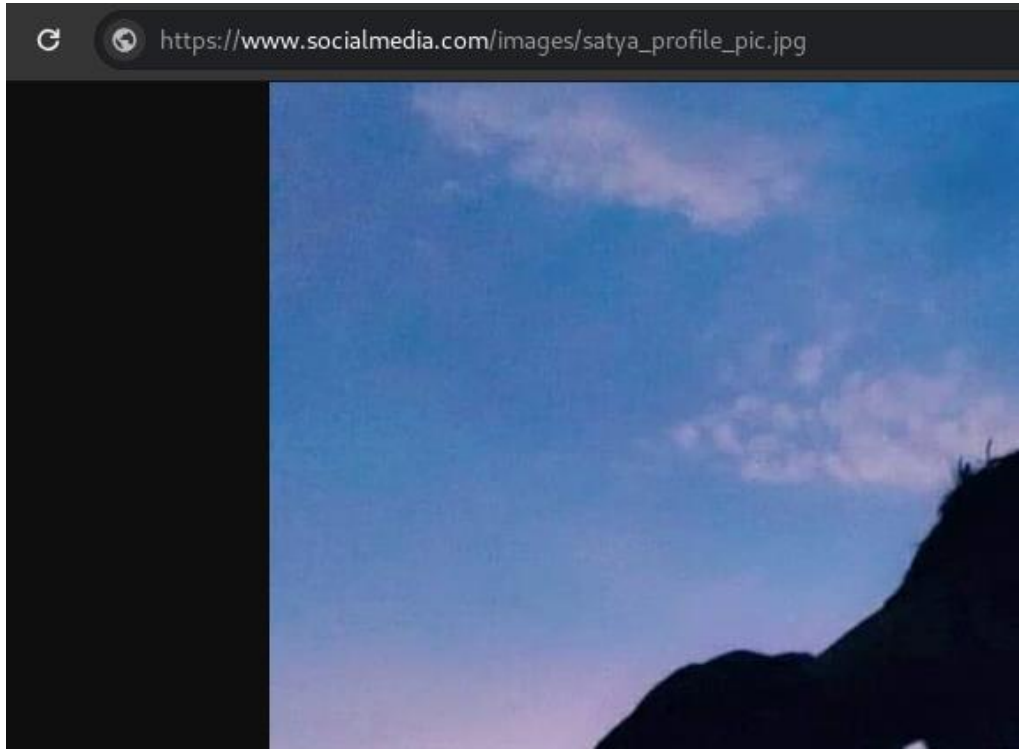
5. RANDOM ENTITY ID

Every time a data entry is created in a database (like a new user or new post) their unique id must be randomly generated and not in a predictable order. Thus, making it difficult to guess the ID of the next entity, this prevents against attack vectors that use known entity IDs, or relation in these IDs.



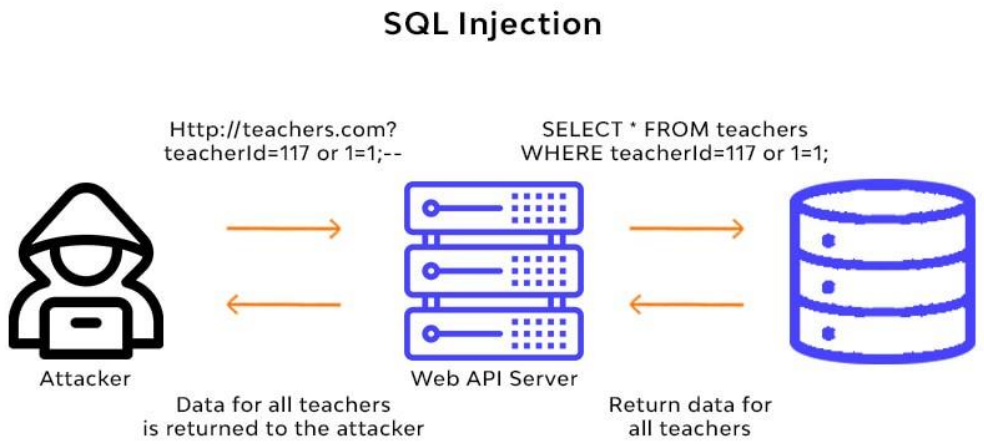
6. ADEQUATE AUTHORIZATION

After user authentication, data should also be authorized if it is accessible by the user. All requests by a user requesting data which he does not have access to must fail.



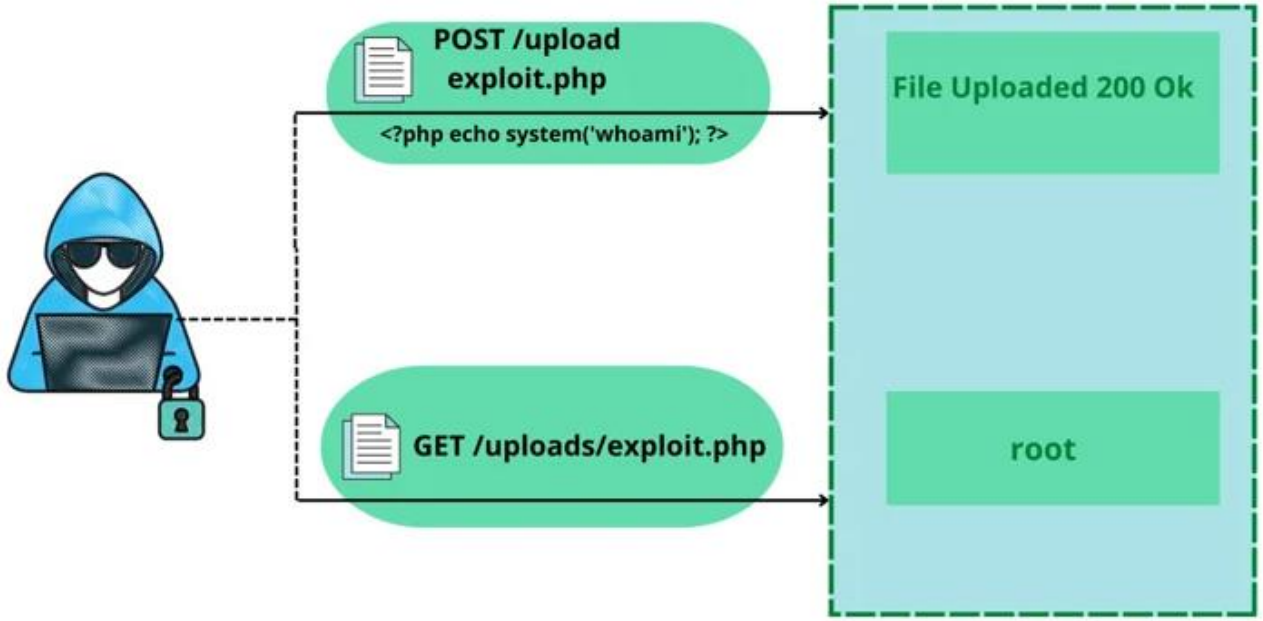
7. DATABASE QUERIES

To prevent SQL injection, use of input filtering methods like query builders, ORMs etc. must be mandated for querying the database. They help filter user given data in the query statement to prevent unwanted behavior.



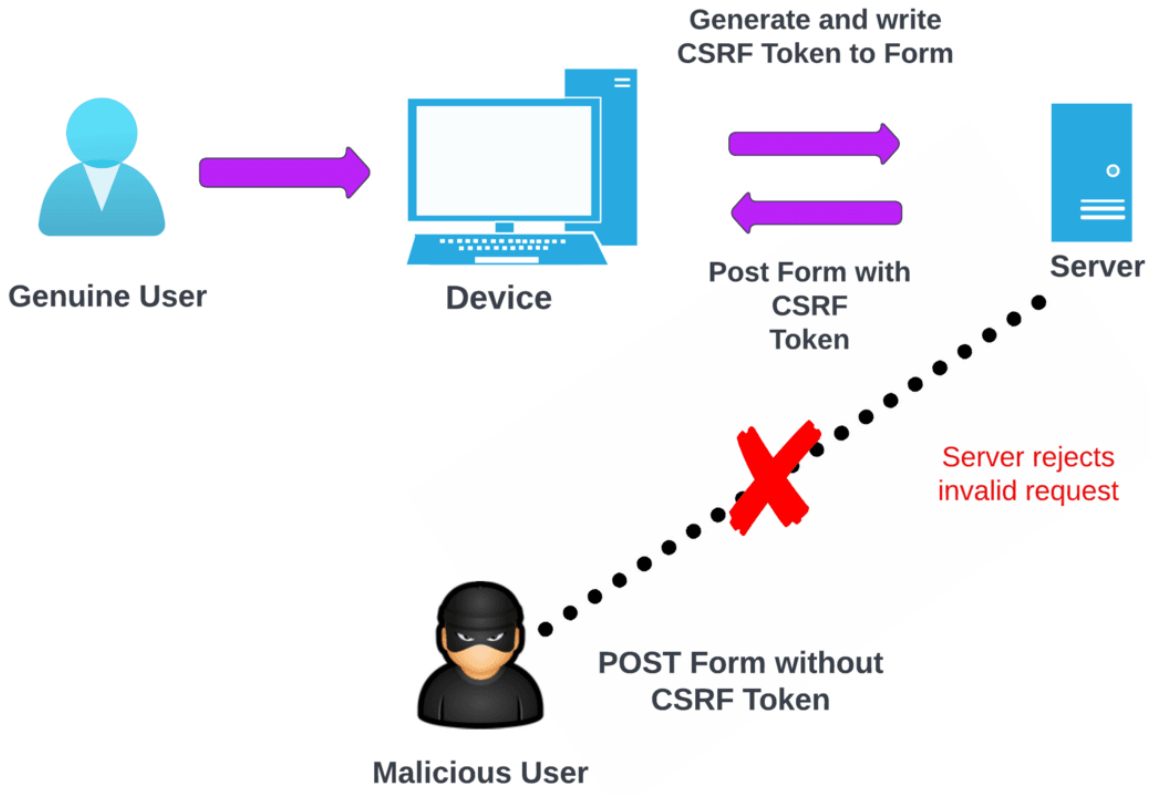
8. FILE UPLOAD

All file uploads must be validated and given a new name(preferably random)and the extension must be verified at backend. Example: rev_shell.php.png will be renamed to ss123.png. This prevents against file upload vulnerabilities.



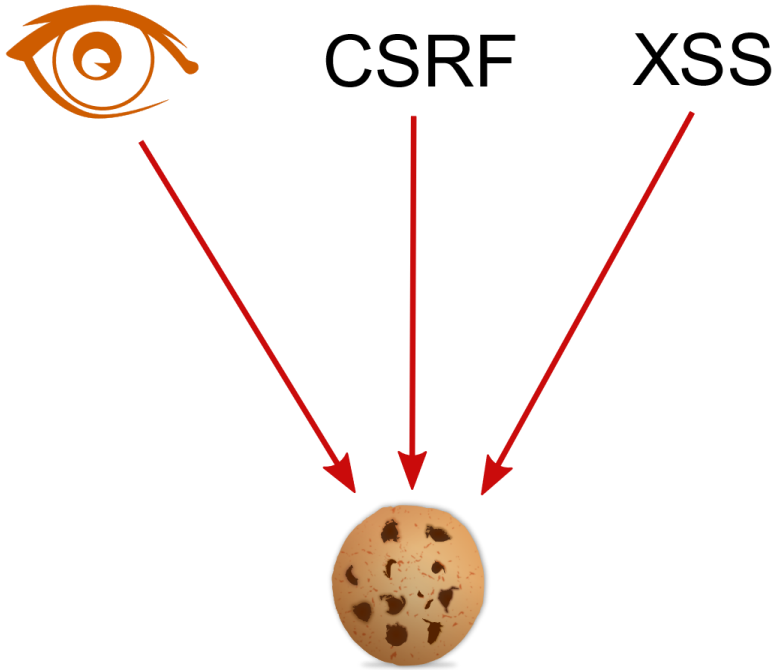
9. USE CSRF TOKENS

To prevent Cross-Site Request Forgery all requests must be accompanied by a CSRF token. One way to do it is using HTTP only cookies and creating one cookie per session which must be in URL params of every subsequent request. For validation URL token and cookie token must match.



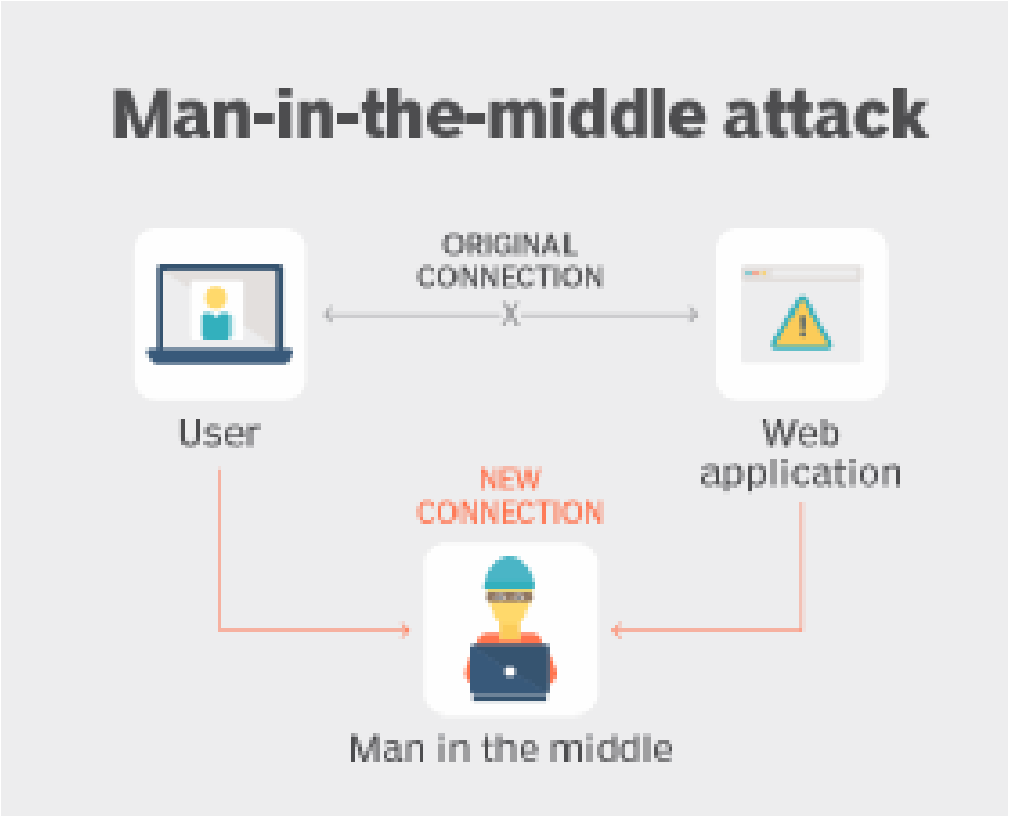
10. COOKIE POLICIES

Cookies must always be HttpOnly and Secure. HttpOnly cookies cannot be accessed via the page's javascript hence renders Cross-site scripting useless and Secure cookies are only sent over https connections, so they prevent against network and sniffing attacks.



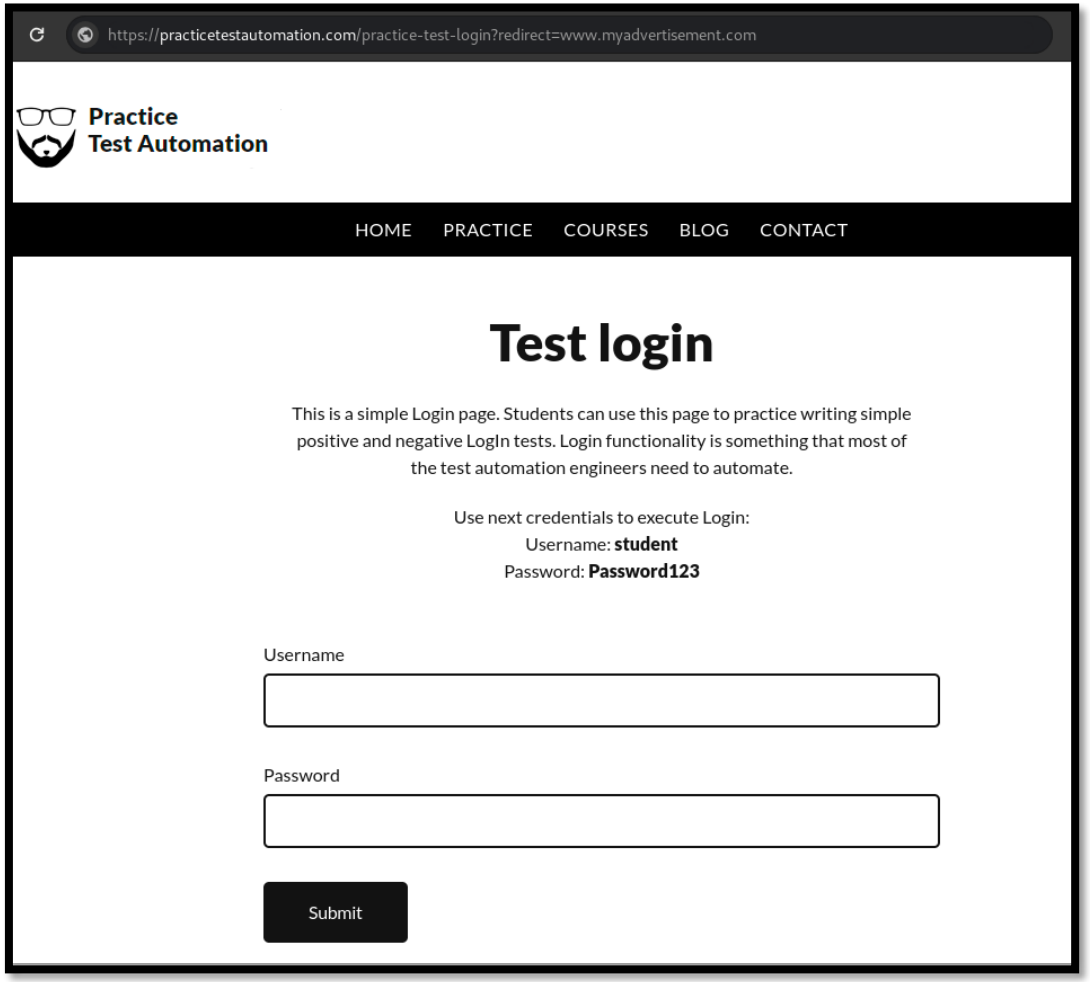
11. ENCRYPTION AND SIGNING

Any data being kept at the client side should always be signed to prevent unauthorized tampering. All sensitive data must be stored in encrypted form to prevent against sniffing attacks.



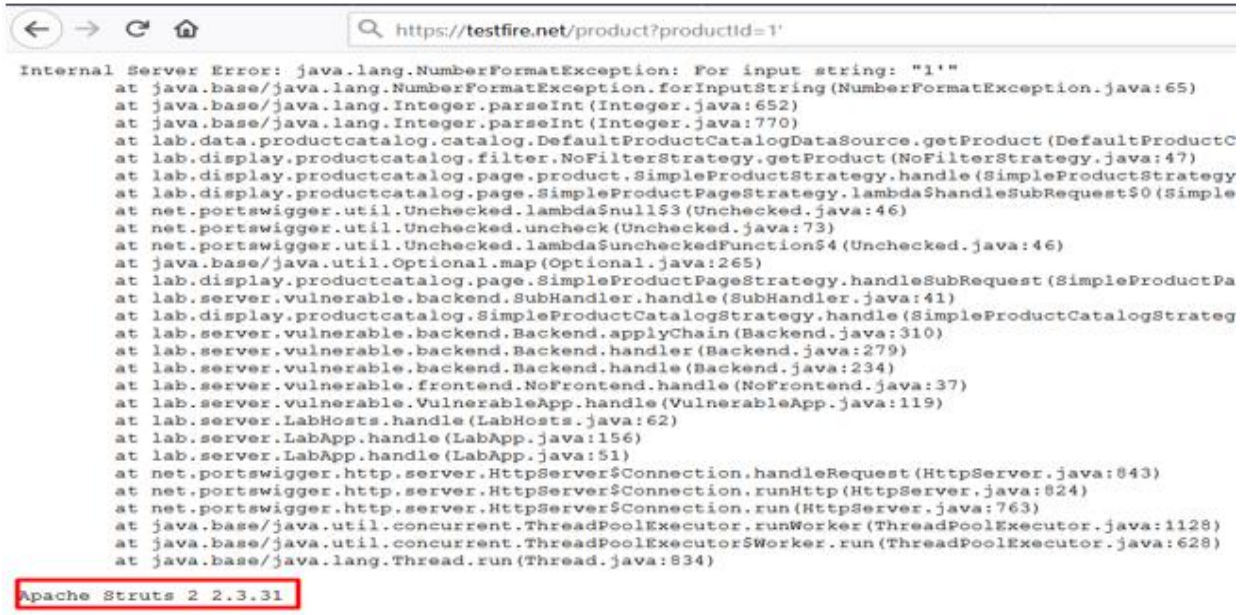
12. PARAMETER AUTHENTICATION

Unauthenticated parameters can lead to unwanted redirects and XSS attacks. These parameters can be given values that are executable code snippets or redirect to malicious page after some action. Hence, it is important to validate parameters.



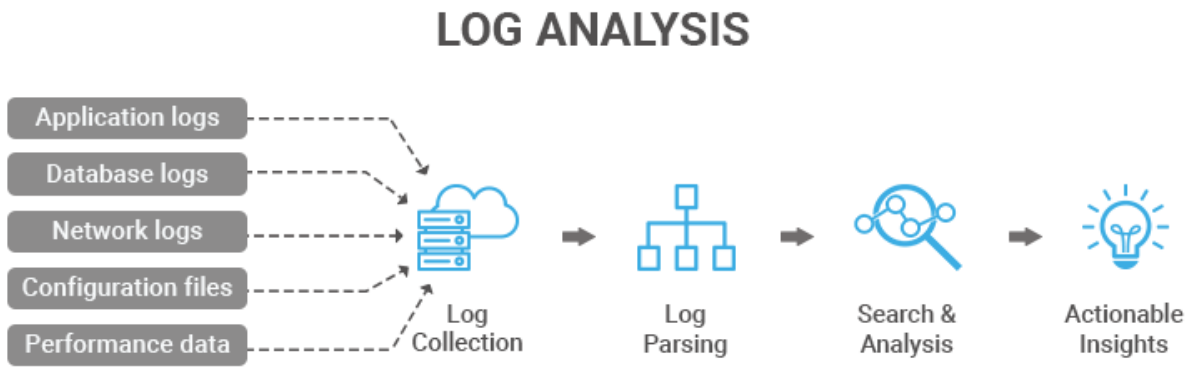
13. PROPER ERROR HANDLING

All exceptions must be handled to prevent data leakage from errors and DoS attacks which crash the server repeatedly. A global catch block(in Java) or an error catching middleware(in Node) at the end of your request line should always be present to ensure that uncaught exceptions do not crash your server instance.



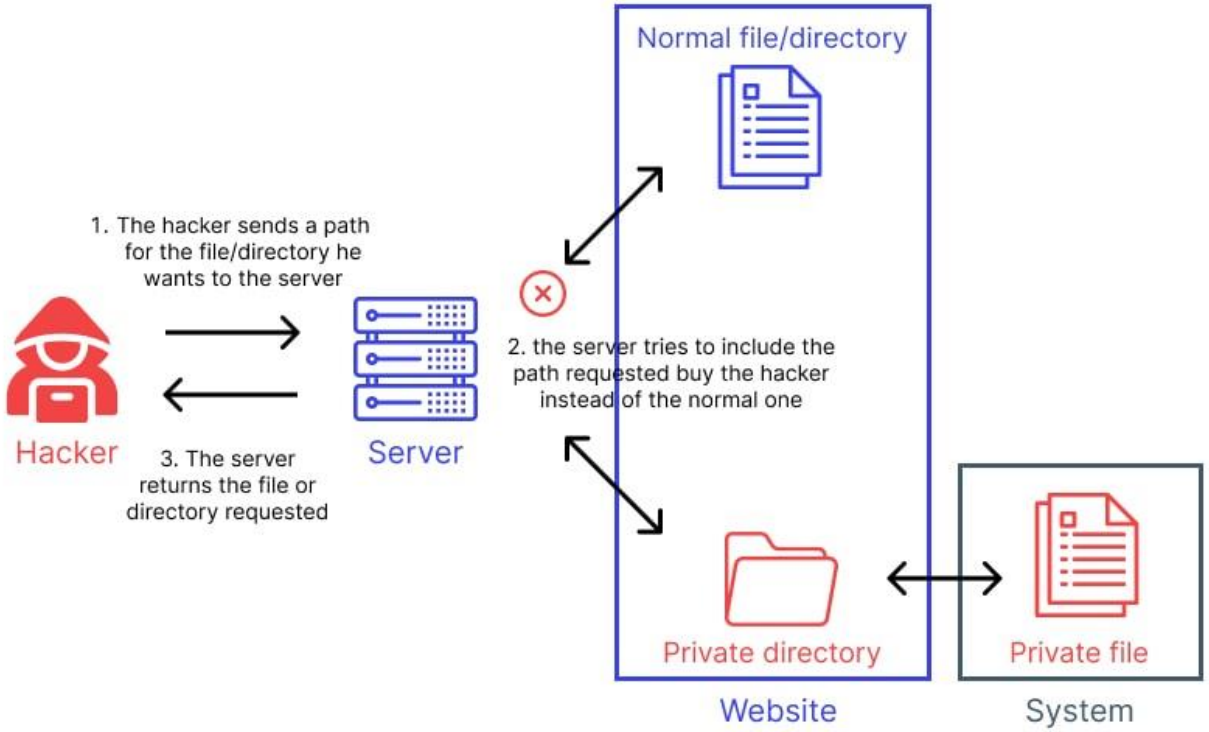
14. LOGGING

Proper logging with extensive details must be put in place for server activities so that active threat detecting models(like SIEM) and security auditors can effectively detect attacks and take preventive measures.



15. STATIC SERVING

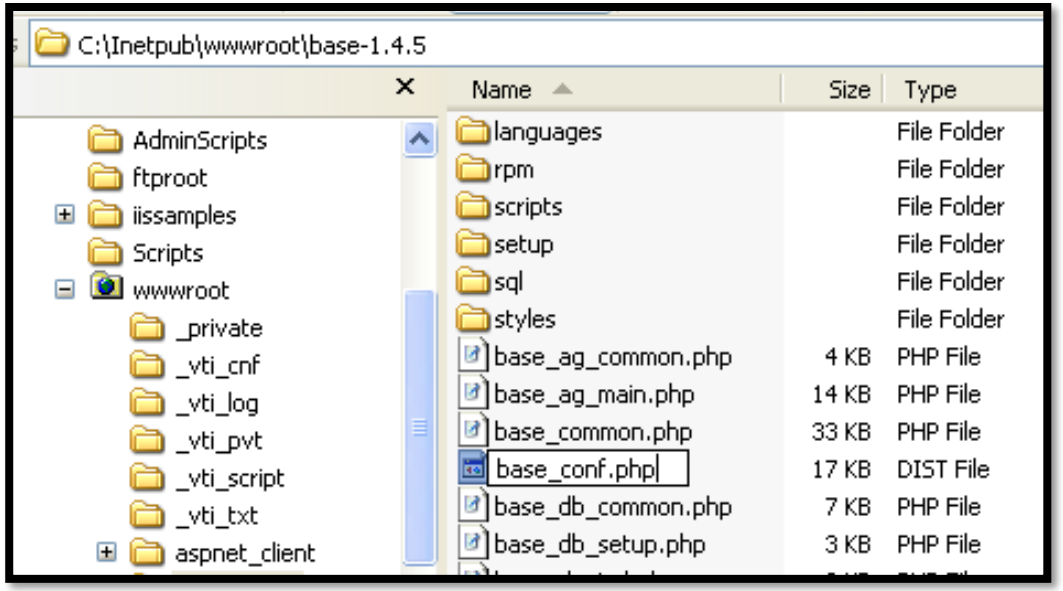
Serving static directories is discouraged and use of static file serving is encouraged. This mitigates directory traversal as only authorized file are served and prevents code upload vulnerabilities as the uploader does not know where the uploaded files are stored on the server.



DEPLOYMENT GUIDELINES

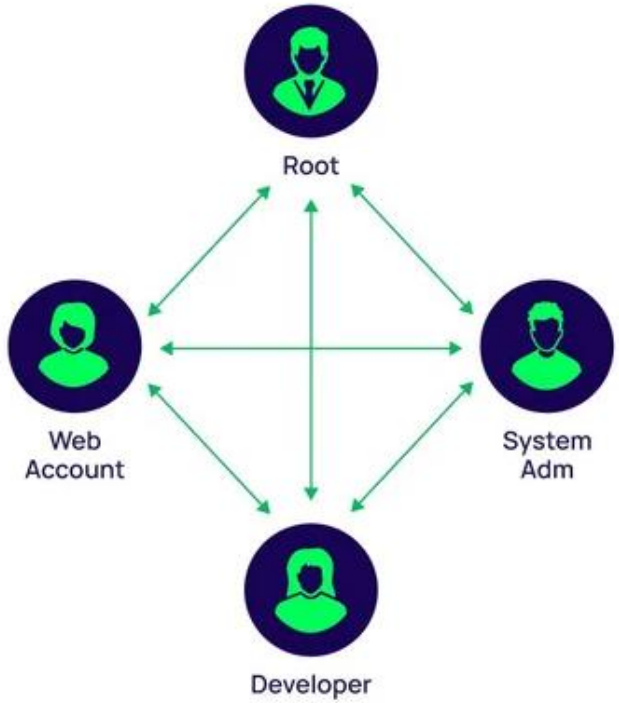
1. SECURE DEFAULTS

All your server and OS configurations must be the strictest policies defined in the base.conf(or any other configuration file) file which also serves as the default config file, all other config files overwrite the config parameters as required.



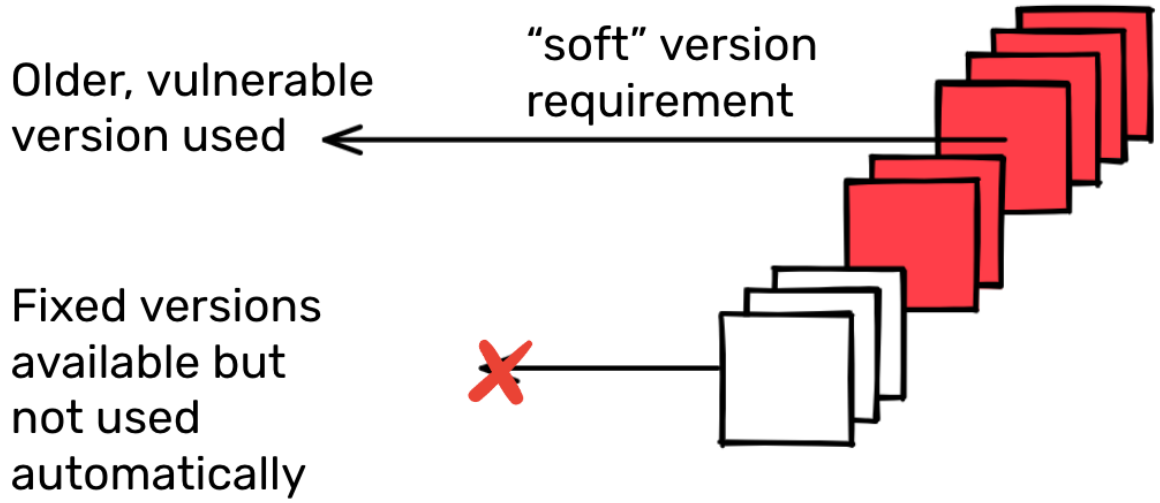
2. LEAST PRIVILEGE

Server applications must be run with the least privilege required for it to function properly and serve the business logic. This helps control damage in case of breach and helps against privilege escalation and limits remote code execution.



3. UP-TO-DATE DEPENDECIES

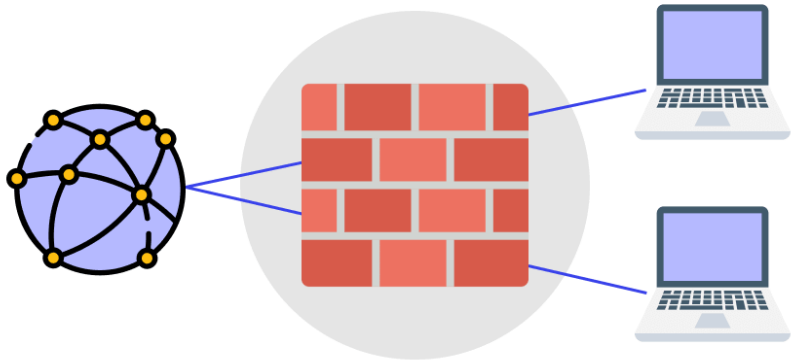
Application dependencies must periodically be updated for security patches and bug fixes. Updating OS packages is also a crucial part of it.



4. USE OF FIREWALLS

Firewalls protect servers against DoS attacks like flooding and can help prevent remote file inclusion, buffer overflow etc. A firewall serves a lot of purposes including defending internal networks and resources from illegal access.

The concept of a firewall

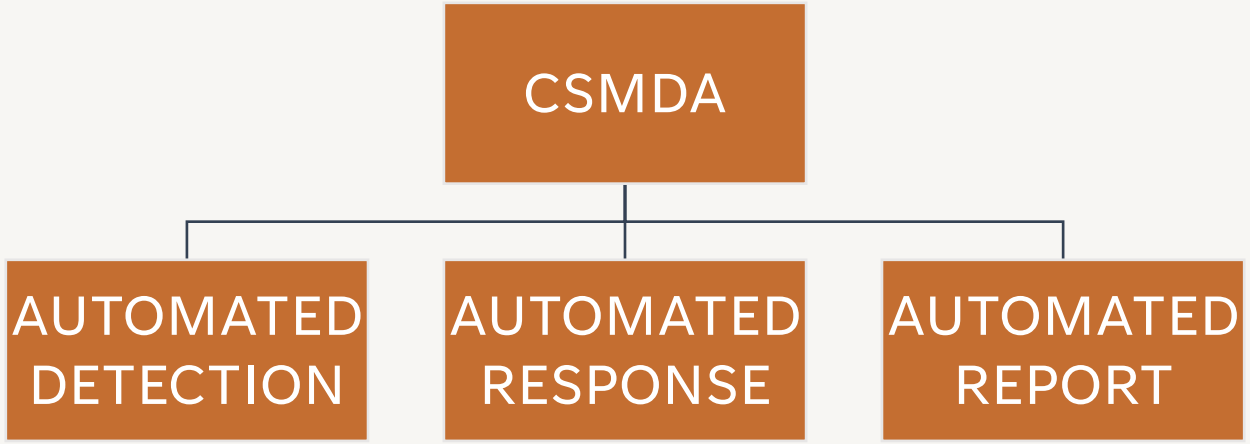


5. OS HARDENING

OS hardening simply refers to removing unnecessary programs and services on the server, these help prevent against unwanted risk factors.

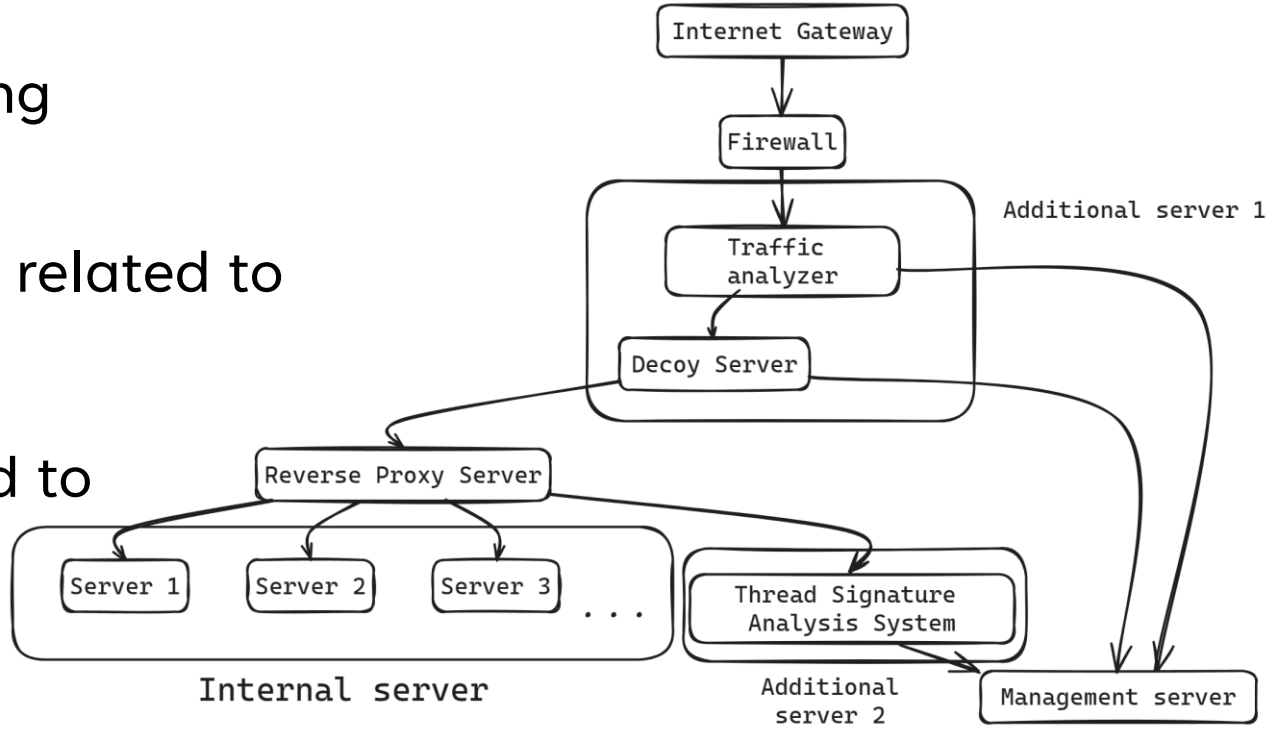
CENTRALIZED SECURITY MONITORING AND DETECTION APPLICATION

APPLICATION SUMMARY



APPLICATION ARCHITECTURE

- It utilizes 2 extra agents with existing infrastructure .
- First agent generally detects attack related to network.
- Second agent detects attack related to application.



AUTOMATED DETECTION:

TRAFFIC ANALYZER & DECOY SYSTEM

- Traffic analyser Identifies suspicious patterns in network traffic.
- Common attack vector like brute forcing can be detected before they hit, using rules and anomaly detection.
- The Fake decoy server will lures attackers, revealing tactics and triggering alerts.

THREAT SIGNATURE ANALYSIS

- It is an automated system that will continuously monitor log data from various sources like network devices, applications, and security tool.
- The data is verified with the Symantec's database for any known attack.

INTEGRITY IN CI/CD PIPELINE

- The application will calculate and store hashes of code change in CI/CD.
- Further during each stage the code hashes are verified.
- This ensure traceability and accountability of all code.

TECHNIQUES TO DETECT PHISING ATTACK

- Suspicious links and resources can be blocked and the user/application will be provided with a warning.

AUTOMATED RESPONSE:

AUTOMATED RESPONSES

1. **Incident in Traffic Analyzer:**
    - Firewall blocks abnormal IP behaviour.
2. **Incident in traffic analyser and Decoy Server:**
    - Extracts attacker's tactics from logs.
    - Destroys and deploys additional decoy servers.
3. **Incident in Development Pipeline:**
    - Isolates compromised systems to prevent lateral movement.
    - Halts pipeline execution until issues are addressed.

AUTOMATED REPORTING:

AUTOMATED REPORTING

- After detection from the respective agent, CSMDA starts preparing the report using log data like Ip and time stamps from the traffic analyzer, malware, and attack vectors from the decoy server.
- Using these logs, it will correlate them with Symantec's attack signature database.
- Along with it will also send emergency notifications to security personal.

THANK YOU