Project Planning	Name(s): Dale Warburton

Detail what **hardware** and **software** you will be using and why.

Hardware:

Raspberry Pi 4 B –
Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
8GB LPDDR4-3200 SDRAM, 2.4 GHz and 5.0 GHz IEEE, 802.11ac wireless, Bluetooth 5.0, BLE, Gigabit Ethernet, 2 x USB 3.0 Ports, 2 x USB 2.0, Ports, 2 x micro-HDMI, 40 pin GPIO header, micro-sd slot, Power – 5v DC (min 3A) over USB-C or GPIO header, Cisco Meraki Router, Netgear 8 Port Switch.

Raspberry Pi 4 B: The Raspberry Pi is a single-board computer (SBC) built onto one circuit board. It includes, CPU, GPU, RAM, Network hardware, USB and HDMI ports, Storage Interface and GPIO pins. It is a lightweight computer and is suitable enough to power small projects. It is the choice for a lot of personal projects due to its capabilities and has a large support network. For the project planning, all hardware and software components will be broken down to find out how they function, perform and the level of security required when used within a LAMP stack E-commerce web platform.

Broadcom BCM2711: This is the central processing unit (CPU) within that drives the Raspberry Pi. This chip acts as the brain of the system handling every system task from operating system to web server operations. It runs in parallel with the Quad core Cortex-A72 (ARM v8). For processing around 100-150 simultaneous dynamic requests this chip would process requests sufficiently.

The successor to the BCM2711 is the BCM2712, it features a quad core ARM Cortex-A76 cluster running at a clock speed of up to 2.4 GHz. This would result in x2 x3 times the performance in CPU-intensive tasks and could potentially manage significantly more users (closer to 1,000 users making simultaneous dynamic requests) It supports faster memory, enhanced caching and improved GPU.

The BCM2711 has been identified with several security flaws including a speculative code store bypass vulnerability (CVE-2021-26313) that could lead to data leakage. This is a medium-severity vulnerability that can lead to attackers being able to leak data through a side channel attack potentially breaching the data protection act. The fix involves applying a security patch from AMD to update the microcode on the affected CPU.

To keep the device's operating system and drivers up to date patching and following guidance from the Raspberry Pi Foundation help keep the system secure.

Quad Core: The quad core is made up of four cores that each handle its own processes or thread. In this system the application server can handle multiple simultaneous requests effectively. In the LAMP web server application typically, a core will work like this.

Core 1 – Operating System and Background Tasks (Linux)

Core 2 – Web Server (Apache)

Core 3 – App Logic (PHP)

Core 4 – Database Queries and I/O (MySQL)

This separation prevents any one process from choking the entire system under a moderate load. For the E-commerce website if multiple users are online and make numerous requests the quad core provides an effective process of dealing with those requests simultaneously by spreading the tasks across each core.

SoC: This stands for system on chip which includes the Cortex series, Broadcom BCM27111, memory controller and other essential circuits, I/O, USB and networking.

Cortex-A72: Provides the CPU architecture and is responsible for the logic that defines how the processor (Broadcom BCM2711) executes instructions. It is a three-way decode out of order superscalar pipeline that is designed for complex tasks in mobile and embedded systems. It can be configured with up to four cores, which is our set-up. It also includes its own L1 instructions and data caches, and a shared L2 cache.

In our E-commerce store caching could be a great way to improve performance. It works by creating a cache and storing recurring web requests into memory (RAM). This method keeps the pages available quicker and avoids constantly re-fetching them, saving memory and processing power. In a live E-commerce environment this strategy would be a great way to save on memory and improve performance which is one of Cortex-A72 supported features.

The natural upgrade would be its successor Cortex-A73 which offers significant improvements in performance and power by around 30% and provides a L3 cache upgrade.

Within this project if you are looking at supporting more than 150 simultaneous dynamic requests, when you upgrade the CPU to BCM2712, you would also need a compatible Cortex-A76 to support running at 2.4GHz.

The Cortex-A72 has a side channel vulnerability (CVE-2020-13844) that enables a malicious actor to improperly gather small bits of sensitive data from privileged memory like the CPU cache or DRAM. Fix, update relevant software & versions & 'secure stacks.'

Another side channel vulnerability is the Spectre-BSE which can allow local attackers to exfiltrate sensitive data by gaining a weak form of control over the processors branch history. (CVE-2024-10929). Updating systems software can help breach of data and potential conflict with the data protection act.

ARM v 8 64-bit: ARM v 8 is the underlying instruction set for the Cortex-A72 and defines how the processor "speaks" machine code. It provides modern desktop-class instruction set and is efficient for multi-tasking.

64-bit describes how wide the processors data pathways and internal registers are. It is the size of the chunks of data the CPU can handle in a single operation. Bits are binary data which is 1s and 0s which can be processed in 64-bit chunks. Meaning that it when it adds numbers it can handle up to 64 bits of numeric data in one go. When it loads memory, it grabs 64-bit word (8 bytes) at once and when it moves data between registers, each register can hold 64 bits. Registers are high speed storage inside the CPU (each 64-bits wide). 64-bit is generally considered better for tasks that require large amounts of memory and dealing with large data sets. Whereas 32-bit systems are limited to 4 GB of RAM. A 64-bit system could theoretically use up to 16 billion GB. 64-bit is considered mainstream and for general use, If the website required more users than around 100-150, let's imagine the online shop went viral and thousands of users required simultaneous connections, a 128-bit / 256-bit would be a more powerful option to be able to process a higher number of requests.

From a security standpoint the ARMv8 adds modern security features like hardware exception levels which define four main privilege levels called Exception Levels (EL).

These separation levels are important, when running a web app on a Raspberry Pi it will run these privileges based on requests. For e.g. if it needs to send data it will ask the kernel (EL1) through a system call.

EL0: User Mode – Who Runs on this level: Apps (PHP, Browser)

This level is the lowest privilege, and programs can not directly access hardware or system memory. They must request help from the OS. This means a malicious app can't crash or hijack the system.

EL1: Kernel Mode – Who runs on this level: Operating system (Linux Kernel)

Has full control over memory, processes, and hardware. Manages what user apps are allowed to do. Provides isolation, if one process is hacked it cannot easily infect another.

EL2: Hypervisor – Who runs on this level: Virtualization layer

Used if running virtual machines (like on cloud servers) not applicable in this scenario.

EL3: Secure Monitor – Who runs on this level: Secure world / firmware

Used for "trusted" code such as secure boot or firmware-level operations. Very high privilege. Security-critical firmware is isolated from the normal OS, if Linux is compromised the firmware stays safe.

The ARMv8 has security flaws that include a stack underflow vulnerability in v8-M TrustZone processors (CVE-2020-16273) this can allow non-secure software to manipulate secure stacks if they are not properly managed. To fix this vulnerability software developers must implement "stack sealing" in the secure software running on affected Armv8-M processors with the Security Extension (TrustZone). This is a software-only mitigation, and no hardware changes are required.

A bug in the Privileged Access Never (PAN) security feature (CVE-2020-16273) and a more recent issue with the Memory Tagging Extension (MTE) that was bypassed through execution to leak sensitive information have been found. This vulnerability has been described as potentially unfixable and affects all ARM CPU's that use MTE.

8 GB LPDDR4-3200 SDRAM: This is the RAM, random access memory for the computer which is used to store temporary data and the instructions for the program. 8 GB means 8 gigabytes of RAM.

LPDDR4 stands for Low-Power DDR4 memory which is energy efficient for embedded systems / general use computers / computers designed for a single task.

3200 is the memory speed, 3200 MT/s (mega transfers per second) this is how quickly data moves between CPU and RAM.

Within this stack the RAM will store operating system data, PHP processes, MySQL buffer/caching and application data.

In terms of performance, it can handle small to medium dynamic workloads. 8 GB would be split like so.

2-3 GB to MySQL buffer pool

2 GB Linux operating system

4 GB for PHP, caching (if implemented) and temporary buffers.

Any heavy database writes or in memory caching that could exceed the RAM would trigger a swap to disk memory which would cause slower performance. More users would take up more RAM, for 100-150 users this may be okay but anything more could require an upgrade to either 16GB or 32GB would allow space for more memory.

Modern DRAM chips (DDR and LPDDR) are affected by a vulnerability in deployment of internal mitigations against RowHammer attacks known as Target Row Refresh. To exploit

this vulnerability the attacker needs to create certain access patterns to trigger bit flips on affected memory modules. Attackers may be able to conduct privilege-escalation attacks against the kernel. Advanced mitigation is required that includes, SoftTRR a software approach that protects page tables, other techniques are available, without going into more detail, REGA, DDR5 and RFM.

Gigabit Ethernet: This would be the preferred method to provide a wired network interface that connects directly to LAN using an ethernet cable. It can support speeds of up to 100 megabits per second and is known for its stability, high speed, low latency and is ideal for hosting web services. The Gigabit ethernet will be easily able to process hundreds of dynamic requests and medium to heavy levels of traffic. Low latency and reliable connection would be the benefits of this connection, and 24-hour access could be available and this and would be critical for user experience.

It is known as hard wired and offers less exposure to attacks as it offers a physical connection, it can be paired with HTTPS (SSL & TLS) to ensure that web traffic is encrypted end-to-end. The connection can be placed behind a firewall or router for strong segmentation and security. In our case we will have the Cisco Meraki firewall sat in front as an extra layer of security.

With this being a physical connection, back-ups systems off premises via the cloud or another isolated server could be required for disaster recovery purposes or if data or system access is compromised by a malicious actor. Physical threats occur from an insider threat or malicious actors if they access the ethernet cable they could cut the power supply. Mitigation includes considering an Uninterrupted Power Supply and restricted access areas.

USB 3.0 and USB 2.0 Ports: The Raspberry Pi includes the four ports which could be useful to attach external SSD (solid state drive) for faster storage or simply for connecting other hardware. The MySQL data is most likely going to be responsible for taking up the majority of memory / storage resources. Presuming that the database is on premises, the ports could be used for external hard drives along with an SSD to offer extra storage for the large amounts of data that MySQL would produce.

Using USB devices creates multiple security risks such as implementing malicious devices carrying, worms, viruses, ransomware or allow unauthorized access to the network, insider threats, man-in-the-middle attacks, if the device is compromised. Theft of the USB could lead to loss of data, putting you at risk of breaching of the data protection act. Encryption must be used to help mitigate along with limiting physical access and using file permissions. Only trusted devices must be used.

Micro-HDMI: The Micro-HDMI is used to connect devices such as TV's monitors and small cameras.

The Micro-HDMI is vulnerable to Electromagnetic Eavesdropping (TEMPEST Attacks): Researchers have demonstrated methods to capture and reconstruct the video feed being

transmitted through HDMI cables by intercepting electromagnetic radiation. Mitigation employs physical shielding measures for cables and equipment to minimize leakage.

It can also be prone to side channel attacks and malware transmission attacks, insider threats, man-in-the-middle, espionage, measures to stop these attacks include, using encrypted connections, being vigilant with connections and using physical protection measures such as protecting port damage and restricted access.

Power — 5V DC (min 3A) over USB-C or GPIO: The Raspberry Pi 4 B requires 5 volts direct current (DC) as its power input with a minimum of 3 amperes (3A). The standard way to power this is via USB-C port. GPIO header is an alternative method to power the board, usually used with custom or embedded set-ups.

The USB-C power supply includes power surge protection and over current protection which protects the circuit board from damage.

Incorrect wiring within the GIPO Header can cause a physical security risk. Always use proper connectors and wiring.

In terms of performance this is a standard for light to medium web servers processing a few hundred requests per minute. Anything beyond a few hundred dynamic requests and it would require an upgrade.

Potential for Denial-of-Service by unplugging. Consider implementing uninterrupted power supply (UPS) and restricted access to areas where the power supply is.

Cisco Meraki Router: Cisco is one of the world's largest telecommunications companies that provides hardware such as the Meraki Router. It is a cloud-managed network security and routing device. It routes packets between devices on the network, protects the network with a firewall, by filtering and blocking traffic / malicious or unauthorised requests. Has an in-built VPN for secure remote access through encrypted tunnels along with cloud management via the Meraki dashboard.

Models range from 250Mpbps to over 1GPS enabling speeds to support 50-100 clients. The router comes with configuration and automatic firmware updates ideal for uptimes and consistency.

The slight risk comes from its cloud connectivity meaning that your dependent on Cisco servers not going down. Cisco Meraki is expensive and it's hard to migrate from. For our purpose depending on model, it is a great choice.

If an attacker gets access to the local network the Cisco Meraki router is vulnerable to AnyConnect VPD DoS attack where a remote hacker can cause a Denial –of-Service attack on MX and Z series devices with Cisco AnyConnect VPN enabled. It causes the server to

restart and all active VPN sessions will fail preventing new connections until the system is restored. Cisco has released software updates to address this issue. To mitigate keep the router software updated.

Netgear 8 Port Switch: The Netgear switch is responsible for connecting computers, servers and networked devices with a LAN so they can communicate efficiently. It provides 8 Ethernet ports for the Raspberry Pi and expands the local network so other devices can connect. In our network it will sit between the router and the Raspberry PI. The firewall will sit in front of the Netgear switch adding a layer of protection to the system. It is a physically wired connection adding more security than wireless as it is less vulnerabilities to unauthorized access. It relies on secure protocols being used (HTTP, SSH, VPN (Cisco) for data protection.

It requires minimum configuration but comes with risks as any device plugged into the switch could gain access, insider threats or malicious actors who gain access to the device could insert a device that downloads malware, spyware or any type of malicious downloadable software viruses. To mitigate you could use physical security and VLAN, virtual area network to create virtual boundaries, preventing a security breach in one network segment from effecting other parts of the network.

2.4 GHz and 5.0 GHz IEEE 802.11ac wireless (Wi-Fi): These are two wireless frequency bands that support the Raspberry Pi. This feature is not needed as the website would be hosted online via a secure and reliable physical ethernet cable connection. The wireless access could cause major security flaws by allowing hackers to gain unauthorized access to the network. For all intents and purposes these two wireless access points would be disabled, stopping all malicious attempts to compromise the application / network.

Software:

The website is running on a LAMP stack. The Linux version is Ubuntu 22.04 Command Line Interface. It is an LTS version of Linux. The website also uses Flask, Authd and Python.
The software is running on the LAMP Stack: Linux OS, Apache Web Server, MySQL, PHP.

Linux Version Ubuntu 22.04 Command Line Interface:

Codename is Jammy Jellyfish, was released in 2022. It offers (TLS) long term support until 2032 with updates. The function of the Ubuntu version of Lunix is that it will be the underlying operating system. Linux Ubuntu supports the integration of the other software. Apache, MySQL and PHP will all be installed and configured on top of the Linux Ubuntu 22.04 version.

Ubuntu's support model means you get patched security updates; it has pre-built in security features like AppArmor which is an application-level firewall. It supports full-disk encryption for sensitive data. Is great at integrating with Apache, MySQL and supports modern day

DevOps technologies such as Docker, Kubernetes and containerization. It has a great community support and is compatible with a wide range of hardware including ARM, which is within our hardware set up.

The Linux Ubuntu version has been discovered to have several vulnerabilities including Network traffic control, (CVE-2025-37797, CVE-2025-38083) a malicious actor could exploit a vulnerability to access the network and carry out malware, ransomware or Denial-of-Service attacks. To mitigate an update to the kernel packages is required to patch.

Flask: is a web framework for Python, A framework is a collection of prewritten code, libraries and tools designed to simplify and speed up development. Python is pre-installed on Ubuntu. Flask is open source and web apps, REST API's and microservices are use cases. It is a flexible and lightweight framework that only installs core essentials and does not come with the full weight of Django. Another Python Framework.

Flask is vulnerable to cross-site-scripting attacks, which occurs when user input is displayed in a webpage without proper escaping. <script>alert('hack')</script>. This script could download malicious code unless proper validation and use of escaping variables, using output encoding, HTML sanitization and framework security is in place. It is also susceptible to SQL injections, use of ORMS (SQLAlchemy) or parameterized queries instead of raw SQL to mitigate attacks. HTTPS must always be used to avoid data exposure breaching the data protection act.

AuthD: is an authentication daemon for Ubuntu. Its purpose is to integrate Ubuntu machines with cloud-based identity providers so that logins on Ubuntu can leverage other identity systems such as Microsoft ID or Google IAM. It works in tandem with Ubuntu's login/SSH identity stack. It provides modern auth flow with Mult-factor authentication.

Vulnerabilities have been reported such as privilege escalation flaw when a new user logs in via SSH leading to root group assessment. Proper configuration for SSH, PAM and local cashing is important to avoid bypass.

Theoretical System Analysis / Performance

@1.5GHz: This is the speed which the CPU processes. 1.5 GHz equals 1.5 billion cycles per second per core. By running 4 cores there are 6 billion cycles per second. When making a web request all cycles will not run exactly as expected due to other factors such as the operating system, I/O, memory and storage/network all taking up cycles.

Making a request on the product page of the website requires CPU cycles for:

1. Receiving the request (Network stack, Linux) Core
2. Routing and parsing (Apache) Core

3. Executing backend logic (PHP) Core
4. Querying the database (MySQL) Core
5. Assembling Response
6. Sending response back to client

Some of these requests are CPU bound meaning that they are processed solely by the CPU and some are I/O bound (database, network etc). This means you cannot be 100% accurate when breaking down the exact cycle requests per second, but you can get very close by approximating.

Let's look at theoretically maxing the capacity of the above hardware to find its breaking point. That way we can accurately gauge how the hardware will perform and how many users it can cater for.

The LAMP stack E-commerce website will use dynamic requests such as transactional requests, dynamic data requests and other dynamic content requests.

CPU frequency is measured in cycles per second (Hz). To calculate CPU cycles per request, you must find the number of requests occurring each second.

The 4 cores running @ 1.5GHz produce 1.5 billion cycles per second and together 4 cores x 1.5 billion cycles per second = 6 billion cycles per second. <<< CPU Max processing power.

Dynamic Pages

Lamp stack PHP page with 1 database query: 20-40 milliseconds (we will use the average of 0.03 seconds = 30 milliseconds as the consistent dynamic database query in this example) CPU time per request.

Each core: 1 request / 0.03 milliseconds (dynamic data request) = 33 requests per second per core.

4 cores: 33 x 4 = **132 requests per second**

Per minute: 132 x 60 = **7,900 requests/minute**

The above shows that one dynamic database request will be processed by the CPU at 0.03 seconds (30 milliseconds). To find out how many requests a core can handle we take that 1 request and divide it by how long it took which is 0.03 seconds (30 milliseconds) which equals 33 request per second. You then take the 33 rec/sec and times it by 4 because there are four cores to get 33 x 4 which is equal to 132 requests per second. To find how many requests

would be in a minute we times that by 60 seconds in a minute. You get 132 requests per second x 60 = 7,900 request per minute.

We now know that 4 cores can process 132 dynamic web requests per second.

Each dynamic web request takes 0.03 seconds (30 milliseconds) time per core.

CPU Frequency: 1.5GHz = 1.5 billion cycles/sec per core.

In the next step we need to identify how many CPU cycles each dynamic request takes. To do this you need to multiply the CPU frequency by the time per request.

CPU Cycles 15000000000 (1.5 billion) x 0.03s = 45,000,000 cycles requests per second per core.

Now we know that each dynamic request takes 45,000,000 CPU cycles on one core.

To find the total CPU cycles per second we need to multiply the requests per second the four cores can handle which is 132 and times that by how many cycles per second each request uses per core which is 45,000,000.

132 dynamic requests x 45,000,000 cycles per request = 5.94 billion cycles per second.

We know from earlier that:

4 cores running @ 1.5GHz produce 1.5 billion cycles per second and together 4 cores x 1.5 cycles billion = 6 billion cycles.

132 dynamic requests per second takes up 5.94 billion cycles per second. The CPU capacity per second is 6 billion cycles. This is very close to max capacity. When dynamic requests per second is equal to the total CPU available, the CPU will be at full load and will not be able to handle more requests.

Performance Analysis Conclusion: Based on the accuracy of the above calculations, if 132 users all made a dynamic web request on the website at the same time, the computers CPU would almost be at full capacity leaving only 60 million cycles per second as the remaining headroom.

The computer would be running at 99% utilisation with 1% CPU headroom left. This is very close to maxing out. The remaining capacity would only cater for a few extra concurrent web users or a few more background processes which would cause a bottleneck. The bottleneck would cause response delays and even request drops.

To improve performance, the processor could be upgraded to a model with a higher clock speed and more cores. A faster CPU would handle each request more quickly, while additional cores would allow more user requests to be processed simultaneously. As traffic increases- particularly beyond 100-150 concurrent visitors – it would also be necessary to

upgrade the system memory from 8GB to 16 GB or 32 GB, ensuring the server can efficiently cache data and handle larger workloads without slowing down.

When upgrading the processor, the system-on-chip architecture, such as the Cortex series, should also be updated to ensure compatibility and optimized communication between the CPU, memory and the other components.

All software updates and patches must be up to date to immediately mitigate security flaws across all devices. Any pre-identified flaws in the security of software and hardware components must have the identified measures in place. The physical hardware must be in a place of restricted access to mitigate malicious actors and isolate insider threats. The current set up can be secured with the above security measures and others such as ModSecurity, AuthD, Cisco Meraki Router Security firewall VPN, Fail2Ban & and security best practises put into place.

What do you **plan** to achieve with your web server? Use the **SMART** method and set out your **objectives**.

This web server is set out to achieving a level of compliance that could be acceptable and comply with the PCI-DSS (Payment Card Industry Data Security Standard).

Every website that handles payment and card information even through a payment gateway must comply with PCI-DSS. It is a global set of standards for any organisation that accepts, processes, stores, or transmits credit card information. Some of the core requirements that need to be covered are: Install and maintain firewall, do not use vendor-supplied defaults, protect cardholder data, maintain secure systems and applications, restrict access to cardholder data, track and monitor access, regular testing. Given the tools we have available I have conducted a SMART strategy to help mitigate vulnerabilities and comply with PCI-DSS

SMART Linux objectives to comply with PCI-DSS

- **Specific:** Keep a secure OS with regular patching and firewall implementation.
- **Measurable:** Latest updates installed and up-to-date firewall config implemented

- **Achievable:** Scheduled updates have been implemented; Cisco Meraki firewall layer added an extra security layer of security.
- **Relevant:** Ensures complies with PCI-DSS.
- **Time:** Implemented with updates scheduled, firewall layer added immediately.

SMART Apache objectives to comply with PCI-DSS

- **Specific:** Disable directory listings, hide banners, enable ModSecurity.
- **Measurable:** Run script tests to make sure each installation meets requirements such as no headers showing system details and ensure that each configuration has been verified.
- **Achievable:** Versions should be done running on the latest and most up-to date software packages ensuring patching. These can be done quickly and without effecting the web sites core processes.
- **Relevant:** Ensures compliance with PCI-DSS
- **Time:** Implemented immediately within 24 hours

SMART MySQL objectives to comply with PCI-DSS

- **Specific:** Install and configure a WAF (e.g., ModSecurity) on Apache to detect and block malicious SQL.
- **Measurable:** WAF actively logs and block SQL injection attempts / attacks.
- **Achievable:** Use OWASP (Open Worldwide Application Security Project) rule set and leverage ModSecurity.
- **Relevant:** Ensures compliance with PCI-DSS.
- **Time:** Complete within 3 days.

SMART PHP objectives to comply with PCI-DSS

- **Specific:** Keep PHP updated, sanitize user inputs to prevent SQLi/XSS

- **Measurable:** 100% of all user-input based queries refactored, all updates and patches installed.
- **Achievable:** In correlation with SQL all front-end search fields can easily be fixed with proper format acceptance. Updates regularly scheduled.
- **Relevant:** Ensures compliance with PCI-DSS
- **Time:** Complete within 3 days or ASAP.
- **Time:** Complete within 3 days or ASAP.

SMART Objectives to prevent SQL Injection attack

SQL injections attacks are used by attackers to implement malicious code typically using the front-end of your website as an entry point. Below are SMART methods to prevent such attacks.

SMART Validate User Input and Sanitize

Specific: Validate all user input within input fields on the front-end of the E-commerce site that leans on the server to ensure only expected data types and formats are excepted.

Measurable: Make sure forms, query parameters such as search fields, and API endpoints have strict validation rules.

Achievable: PHP's filter_var or Python's validators library along with using whitelist expected format.

Relevant: Reduces risk of malicious input reaching SQL execution.

Time: Implement across all areas within 1 week.

SMART Implement Parameterized Queries

- **Specific:** Use parameterized/prepared statements instead of dynamic SQL in all database queries.
- **Measurable:** 100% of all user-input based queries refactored.

- **Achievable:** Use libraries that support prepared statements such as MySQLi/PDO (PHP) or SQLAlchemy (Python)
- **Relevant:** Prevents SQL injection by separating SQL logic from the user input.
- **Time:** Complete within 1 week.

SMART Use Secure Coding Techniques

- **Specific:** Educate developers and train them on best practises on SQL injection prevention, safe database patterns and security best practices.
- **Measurable:** Ensure dev team member's complete course
- **Achievable:** Use workshops, online courses or use OWSAP SQLi prevention guide.
- **Relevant:** Significantly reduces future vulnerabilities within the code and improves coding best practices
- **Time:** Dev must complete training within first 4 weeks or before being given access to coding platforms.

SMART Conduct Security Testing

- **Specific:** Use tools such as BurpeSuite, sqlmap or OWASP to perform automated SQL injection testing on all endpoints.
- **Measurable:** Have by weekly vulnerability scans and patch findings.
- **Achievable:** Schedule scripting tests.
- **Relevant:** Detects vulnerabilities before attackers.
- **Time: Immediately,** by weekly re-occurring.

SMART Monitor Logs and Alerts for suspicious Queries

- **Specific:** Use Fail2Ban to log monitor MySQL for error logs and Apache access logs to detect suspicious code.
- **Measurable:** Alerts generated for any suspicious SQL queries.

- **Achievable:** Use tools Fail2Ban, create custom scripts or System Incident and Event Management Systems.
- **Relevant:** Helps early detection of SQL injection attempts.
- **Time:** Set up within first few weeks or ASAP

SMART Patching LAMP stack Software

This keeps all software updated and ensures that any the stack is compliant with data protection regulations ensuring no known vulnerabilities exist.

- **Specific:** Regularly applying security patches to Linux, Apache, MySQL and PHP reduces vulnerabilities that attackers could exploit.
- **Measurable:** Ensuring that no known CVE's affect the stack by making sure there are zero outstanding upgrades (app list –upgradeable).
- **Achievable:** By using package management tools and automated updates (if preferred) can be reliably completed with minimum disruption to the site.
- **Relevant:** Keeping the stack patched helps keep sensitive customer data and ensures compliance.
- **Time:** scheduled regularly every week or two-week dependant.

SMART Securing Physical Hardware

- **Specific:** Implement restricted physical access to all server hardware using locked racks, CCTV monitoring and authorized personal list.
- **Measurable:** Have access log and review it daily for unauthorized entries. Review CCTV footage of any security breaches or triggered access.
- **Achievable:** Utilize existing infrastructure, and basic security controls, locks cameras and access passes.
- **Relevant:** Prevents theft, tampering of devices and unauthorized configuration changes.
- **Time:** scheduled within fist week.

Describe the specific Apache hardening steps you plan to take to secure the system.

Installing ModSecurity on Ubuntu:

ModSecurity is a powerful, real-time web application firewall (WAF) protection against a wide range of attacks like SQL injection and cross site scripting by inspecting malicious files and blocking malicious HTTP traffic. It acts as a security layer to safeguard web applications and can be configured to log and block threats. Below is a step-by-step installation guide on Linux Ubuntu.

Step 1 Update & Installation:

- Sudo apt update
- Sudo apt install libapache2-mod-security2 -y

Step 2 Enable Apache Module:

- sudo a2enmod security2

Step 3 Restart Web Server:

- sudo systemctl restart apache2

Step 4 Enable the main ModSecurity config file:

- sudo nano /etc/modsecurity/modsecurity.conf
- Find this line SecRuleEngine DetectionOnly – change to – SecRuleEngine On

Save and exit (Ctr+0, Enter, Ctrl+X) This activates blocking mode.

Step 5: Enable the OWASP Core Rule Set (CRS)

- sudo apt install modsecurity-crs –y

Link the CRS rule to ModSecurity config

- sudo ln -s /usr/share/modsecurity-crs/base_rules/* /usr/share/modsecurity-crs/activated_rules/

Or for new versions

- sudo cp /usr/share/modsecurity-crs/crs-setup.conf.example /etc/modsecurity/crs-setup.conf
- sudo ln -s /usr/share/modsecurity-crs/rules/ /etc/modsecurity/rules

Step 6 edit Apache ModSecurity config to load the CRS

- sudo nano /etc/apache2/mods-enabled/security2.conf

Step 7 restart Apache

- sudo systemctl restart apache2

Step 8 Verify ModSecurity is Active

- sudo apachectl -M | grep security – result should return security2_module (shared)

Step 9 Check logs when making a request

- Sudo tail – f /var/log/apache2/error.log or sudo tail -f /var/log/modsec_audit.log

Enable Apache Security Configuration:

Fail2ban provides an automated defence against brute-force attacks by monitoring server logs for suspicious activity and proactively blocking offending IP addresses at the firewall level.

Step 1 Enable Apache security configuration:

- sudo a2enconf security
- sudo systemct1 reload apache2

Update the system:

- sudo apt update
- sudo apt upgrade

Install Fail2Ban:

- sudo apt install fail2ban -y
- sudo systemct1 enable fail2ban
- sudo systemct1 start fail2ban
- sudo systemct1 status fail2ban
- sudo cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local

Edit with:

- sudo nano /etc/fail2ban/jail.local

To apply changes;

- Sudo systemctl restart fail2ban

Remove Banner headers:

Headers contain information that reveals software name, version number and sometimes operating system details. Having these details exposed could lead to malicious actors accessing those details by using a web server scanning tool. This data could be used to identify vulnerabilities within your system and capitalised on by malicious actors to gain unauthorized access or worse. Below is a step-by-step guide to taking preventative measures.

Step 1 Edit the main Apache configuration file:

- cd /etc/apache2/conf-available/security.conf

- sudo nano /etc/apache2/conf-availability/security.conf

Step 2 Find lines:

- ServerTokens OS
- ServerSignature On

Step 3 Change them to remove Apache version info from error pages and directory listings:

- ServerTokens Prod
- ServerSignature Off

Step 4 Enable config if not already: (ubuntu usually does this by default but if not)

- Sudo a2enconf security

Step 5 Reload Apache to apply changes:

- sudo systemctl reload apache2

Step 6 Verify changes by running curl script

- Curl –I http://yourdomainhere.com

You should see returned: Server: Apache
2. Check an error page to check if it displays the Apache version.
Disable Directory Listings:

By default Apache may display a directory listing when an `index` file (such as `index.html` or `index.php`) is missing from a directory. This can lead to exposing sensitive files, config back-ups or scripts that attackers could potentially exploit. By disabling directory listings it is a fundamental security measure to prevent unauthorized access to such files. Below is a step-by-step guide to disable the directory listings depending on accessibility to the files.

Step 1

Apache controls directory listings using the Options directive, specifically the 'Indexes' Option. To disable directory listings, remove or explicitly deny the `Indexes` option in the configuration file located at:

`/etc/apache2/apache2.conf` in Ubuntu

Step 2

If Directory listings should only be disabled for a specific site, modify the corresponding virtual host file, usually found in Ubuntu here `/etc/apache2/sites-available/`

```
<VirtualHost *:80>
    DocumentRoot "/var/www/example.com"
    <Directory "/var/www/example.com">
        Options -Indexes
    </Directory>
</VirtualHost>
```

Step 3

If modifying the global the global configuration is not an option, an `.htaccess` file can be used to disable directory listing on a per –directory basis. Create or edit the `.htaccess` file in the target director and add:

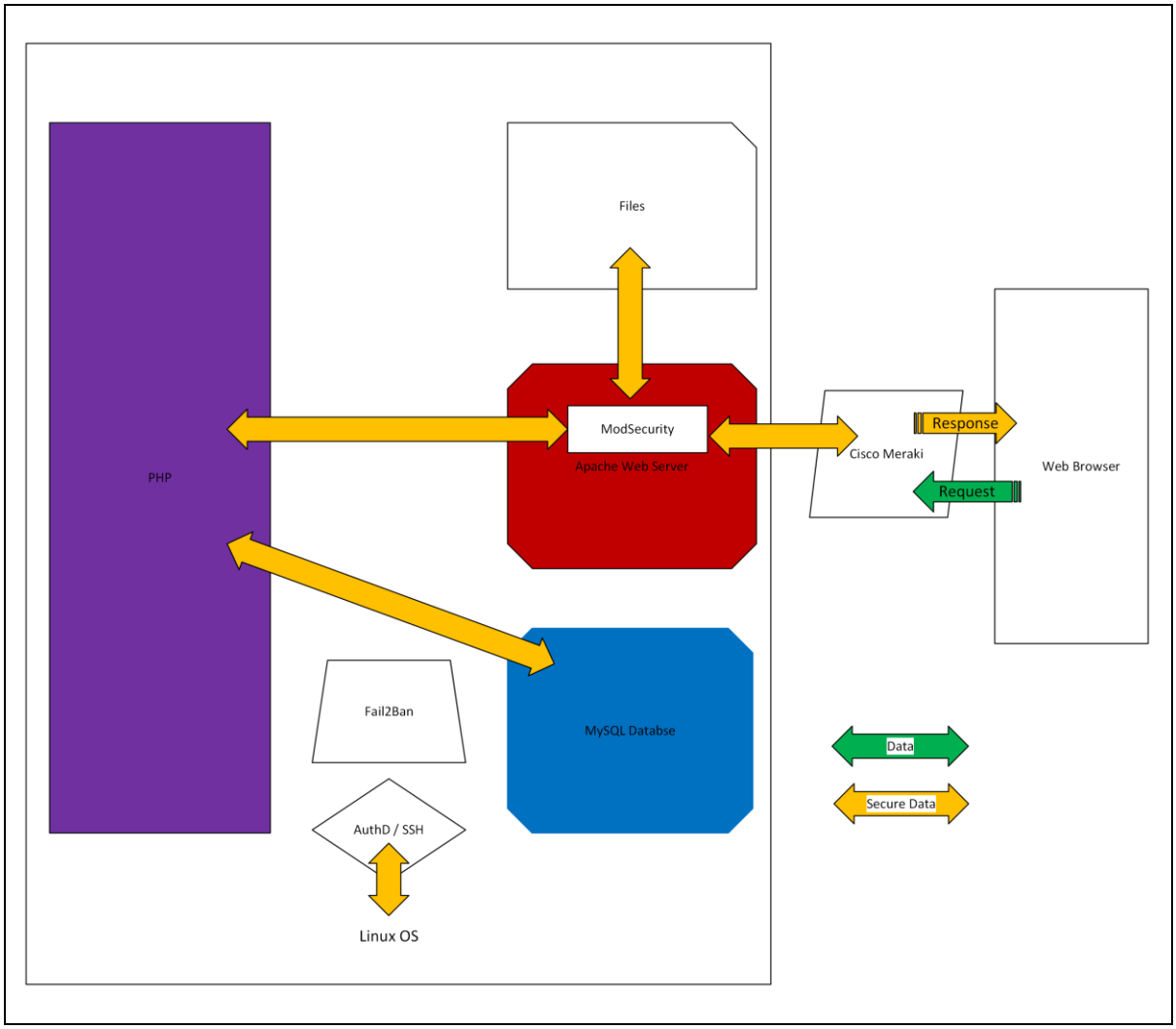
Options –Indexes

Step 4

After modifying the configuration, restart Apache for changes to take effect on Ubuntu:

sudo systemctl restart apache2

Produce a diagram of the **planned security systems** within your **LAMP stack**.


Files
PHP
ModSecurity
Apache Web Server
Cisco Meraki
Response
Request
Web Browser
Fail2Ban
MySQL Databse
Data
Secure Data
AuthD / SSH
Linux OS