2a.

Web application and web service vulnerabilities exist in memory-safe languages due to design, implementation, and configuration flaws. One of the problems is input validation failures: unsanitized user inputs can lead to SQL injection because of poor parameterization that executes malicious SQL, or command injection because of poor sanitization leveraged by attackers to run OS commands. Things go wrong when developers trust user input without validation.

Authentication and authorization flaws may allow unauthorized access or privilege elevation. Predictable session tokens can be exploited to conduct session hijacking, while weak access controls enable access to restricted areas by unauthorized users. For example, there could be attempts by the user to manipulate the URLs for viewing admin pages or data that they shouldn't see.

Misconfigured security settings lead to vulnerabilities. The exposure of sensitive functions, such as admin interfaces or directory traversal exploits, is usually a result of a default or incorrect setup. Not securing directories, disabling unneeded features, or changing default credentials leaves applications vulnerable.

Data exposure is caused by mishandling of sensitive information, such as user credentials. The best examples are plaintext password storage and failure to encrypt data during transmission. Also, verbose error messages may expose internal details, helping attackers create exploits.

Insecure dependencies complicate web application security. Vulnerable third-party libraries or APIs can bring risks to secure applications, as was recently shown by the Log4Shell vulnerability in Apache Log4j.

Logic flaws in business workflows can also allow an attacker to bypass security or perform unintended actions. Examples of this include race conditions that exploit timing gaps and poorly designed workflows that allow skipping critical steps.

Cross-Site Scripting (XSS) is a critical vulnerability resulting from poor user-input sanitization. It allows an attacker to inject and display malicious scripts to users, either stored in a database or reflected in URLs. These types of attacks take advantage of flaws in server-side data handling.

Improper API design and usage exposes web applications to risks. APIs without rate limiting or access controls can enable brute force attacks, denial of service, or data exposure. Similarly, permissive APIs may expose sensitive operations without adequate protections, further increasing risk.

It exploits the trust web applications have in a browser, tricking it into making unauthorized requests. This happens when applications don't use CSRF tokens to validate requests, which allows an attacker to do actions on behalf of users, like changing passwords or transferring funds. More dangerous root causes of vulnerabilities include improper validation, mismanagement of authentication, poor business logic design, outdated dependencies, and lack of secure development practices. These problems emphasize the importance of strong security practices and defense-in-depth strategies to reduce risks, even when using memory-safe languages.

**2b.**

Input Validation and Output Encoding

* Mechanism: This would validate and sanitize user input so as to prevent malicious payloads from being accepted. It also encodes output to ensure it is properly represented in its intended context, such as properly escaping HTML entities to avoid XSS.

Parameterized Queries

* Mechanism: The use of prepared statements and parameterized queries to prevent SQL injection. This separates SQL logic from input data.

Authentication and Session Management Best Practices

* Mechanism: Enforces strong authentication schemes, such as multi-factor authentication, coupled with strong session management, like secure session tokens with short expiration times.

Access Control Enforcement

* Mechanism: It enforces strict authorization policies, ensuring users can only access resources and operations that are allowed. The effectiveness is in mitigating unauthorized access and privilege escalation. The limitation is that it is vulnerable to logic flaws or misconfigured policies.

Security Headers

* Mechanism: Utilize HTTP headers like Content-Security-Policy (CSP), X-Frame-Options, and X-Content-Type-Options to prevent some classes of attack

CSRF Tokens

* Mechanism: Inserts unique tokens inside forms or requests to validate the origin of actions taken by users to avoid CSRF attacks.

Dependency Management

* Mechanism: It includes the process of updating third-party libraries and frameworks with security patches for known vulnerabilities. This can be automated with tools like dependency checkers.

API Security Measures

* Mechanism: Authentication, rate limiting, and input validation of APIs to prevent their abuse and unauthorized access.

WAF

* Mechanism: Monitors and filters HTTP traffic to block malicious requests.

Error and Logging Management

* Mechanism: Configures applications to log errors securely and suppress sensitive information in user-facing error messages.

A graph of different colored bars

Description automatically generated with medium confidence

The effectiveness is represented on the scale from 0-3 with 3 being the most effective.

**2c.**

No, none of the solutions provides complete protection against all web application vulnerabilities. Even with source code changes and recompilation, vulnerabilities can still arise due to design flaws, implementation gaps, or evolving attack vectors.

Input Validation and Encoding can be highly effective against injection attacks like SQL injection and XSS, but ineffective against logic flaws or insecure dependencies. Parameterized Queries can fully mitigate SQL injection but doesn’t address other forms of input manipulation or application-layer vulnerabilities. Authentication and access control reduces risks of unauthorized access but relies on correct configuration and consistent enforcement. Weaknesses in session management can still lead to exploitation. Security headers and CSRF tokens strongly mitigate specific classes of attacks, such as XSS, CSRF, and clickjacking, but do not address other vulnerabilities like API abuse or data leakage. Dependency management mitigates known vulnerabilities in third-party libraries but cannot prevent issues from zero-day vulnerabilities in dependencies. Web Application Firewalls (WAFs): Additional layer of security but cannot completely stop exploitation due to unknown attack vectors or logical flaws.

**3a.**

Let's explore two key approaches to establishing trust in digital systems: Trust-On-First-Use (TOFU) and Trusted-Third-Party (TTP).

Trust-On-First-Use is exactly what it sounds like: the system establishes trust when you first connect and remembers that connection for future interactions. Think of it like meeting someone for the first time-you might be cautious initially, but once you've established that first connection, you recognize them in future meetings. The beauty of TOFU lies in its simplicity. You don't need any complicated infrastructure or external verification systems, making it particularly useful in straightforward, lower-risk scenarios where you can reasonably secure that first interaction.

However, TOFU is not without its own set of risks. The most concerning factor is that first connection-if anyone intercepts it, what is called a Man-in-the-Middle attack-they could pose as the legitimate party. Once that trust has been established, there's no built-in mechanism to help you verify whether you're talking to the right person anymore. If an attacker manages to replace your stored trust information, in some cases, you might never notice.

On the other side is the Trusted-Third-Party model, which much better resembles getting a reference of someone from a mutual friend. In other words, you come to know someone through a common friend before actually meeting the person. In the online world, this role can be played by Certificate Authorities. This model provides very strong security because in this case, you are not dependent on that first exchange alone; there is a chain of trust already. It works particularly well in large systems where many participants need to trust each other, and it's good protection against those irritating Man-in-the-Middle attacks.

But TTP has its own set of problems. Establishing and maintaining such a trust infrastructure is neither easy nor inexpensive. You're also putting a lot of faith in that central authority-if they get compromised, it affects everyone who relies on them. There's also the very practical challenge of what happens when trust needs to be revoked-like when a certificate is compromised. Getting the word out about revoked certificates can be slow and sometimes incomplete, leaving windows of vulnerability.

The choice between these models often depends on your specific needs, weighing security needs against practical considerations such as cost and complexity.

Scenario 1: Personal Communication Devices

TOFU is Appropriate: When you're setting up personal devices, like configuring SSH between your laptop and home server, TOFU makes perfect sense. Since you physically control both devices during the initial setup, you can be confident that first connection is secure. It's like introducing two friends who are both standing right in front of you.

TTP is Not Ideal: For these personal setups, getting and maintaining certificates would be overkill. It's like hiring a notary to verify that you're talking to your own devices - unnecessary hassle and expense for something you can verify yourself.

Scenario 2: E-commerce Websites

TTP is Appropriate: For online shopping, having a trusted third party like a Certificate Authority is crucial. When customers visit an e-commerce site to make purchases, they need solid assurance that they're on a legitimate website, especially when entering credit card information or personal data. It's like having a trusted institution vouch for a business's legitimacy.

TOFU is Not Ideal: Most users won't (and shouldn't have to) manually verify a website's authenticity on their first visit. Without third-party verification, attackers could easily intercept these initial connections, potentially exposing sensitive information. It's too risky for financial transactions.

Scenario 3: Internal Corporate Systems

TOFU is Appropriate: In a small company with its own controlled network, TOFU can work well. When employees connect to internal systems for the first time, they can trust what they see because the environment is already secure and managed. It's like working in a building where access is already controlled at the front door.

TTP is Not Ideal: Building out a TTP infrastructure for internal systems would be like installing bank-level security in a family home - the added complexity and cost doesn't match the need when you're already working in a trusted environment.

**3b.**

What the Unified Model Would Look Like

This unified approach would be applied in layers, starting with the most basic first connection and then adding more detailed security as time progressed. In this, when two systems first connect, they establish initial trust just like in TOFU, storing the public key or identifier from that first interaction. Behind the scenes, though, the system would double-check with a trusted authority-such as a Certificate Authority or blockchain-to check that stored information for an authenticity check. To maintain security over longer terms, the system would intermittently recheck these credentials, ensuring nothing has been compromised.

Problems Solved by the Unified Solution

The integration of this approach offers solutions for various serious security problems: Third-party verification is added against the classical Man-in-the-Middle vulnerability affecting TOFU at the first connection. The method also comes out more cost-effective compared to a pure TTP system since you would not be required to have pre-trust every single device and interaction. It easily accommodates updating of security credentials using the TTP for verification, while still retaining the ease of use of the TOFU relationship. Most importantly, it works well in mixed environments where you have both internal systems (which work well with TOFU) and external services (which need TTP-level security).

Disadvantages of the Combined Solution

There are some disadvantages in this integrated solution. This will make the implementation far more complicated, as a system needs to manage both local storage of trust and the processes of verification with external entities. By incorporating a trusted third party, we add a single point that could be compromised, thus partly undermining the independence of TOFU. There is also a performance overhead: those periodic checks with the TTP require more CPU and network bandwidth, which might be prohibitive in resource-constrained settings. Finally, there is a potential security gap if TTP validation either fails or is delayed, the system might be falling back to trusting an initial key that does not have recent verification.

**3c.**

SSH Connection Security for Remote Servers

A combined trust approach can help when developers and system administrators need to connect to remote servers via SSH. In the first place, the connection is done with Trust-On-First-Use: The SSH client simply remembers the server's public key, so the initial setup is fast and easy. But here's where it gets interesting-after that initial connection, the system checks the server's key with a trusted authority-such as a Certificate Authority or an internal PKI system-to make sure nothing fishy has happened. In this way, you get the best of both worlds: ease of quick setup, combined with the security of verified trust.

Enhancing the Security of IoT Devices within Smart Homes

Smart home security is a unique challenge that is handled particularly well by a unified trust model. When you first set up an IoT device, it uses TOFU to establish trust with your home router - simple and user-friendly. But the security doesn't stop there. Your router then works behind the scenes to verify the device's firmware signatures and certificates with the manufacturer's trust system. The router keeps checking these credentials periodically to ensure your device stays secure. This approach makes setup easy for homeowners while maintaining robust security against tampering or compromise.

Email Encryption and Secure Messaging

For secure communication services such as PGP or Signal, this unified model does particularly well. When users exchange the first public keys or share contact information, they employ the simple TOFU approach-just accept and store the key. But then, the system actually checks those keys against a trusted source, such as a certificate directory or blockchain ledger, for authenticity. This setup makes it easy for users to start communicating securely while protecting against impersonation attempts. It's like having the convenience of trusting someone at first sight, but then quietly verifying their credentials through a trusted source.

**4a.**

AEAD, or Authenticated Encryption with Associated Data, is a cryptographic scheme to provide confidentiality and integrity of data, besides authenticity for secure communication over an untrusted network. That basically brings integrity protection along with encryption of the data, so encryption and integrity verification of data are guaranteed.  
AEAD includes three primary components:

1. Encryption (Confidentiality)
2. Integrity Checking (Integrity)
3. Authentication (Authenticity)

Importance of AEAD Components (CIA)

Encryption (Confidentiality)

Encryption can be thought as a secure envelope for your message; its primary job is to keep your information private from prying eyes. Without encryption, it's like sending a postcard instead of a sealed letter; anyone along the way can read your sensitive information. For instance, an attacker monitoring network traffic-e.g., someone watching mail go by-could easily capture sensitive data like passwords or personal information through simple eavesdropping techniques.

Integrity Checking (Integrity)

Integrity checking provides that a tamper-evident seal on your message is kept in such a way that it gets assured that what arrives is the same as what was sent; modifications anywhere in between are detected. Eliminating integrity checking allows active attackers to modify the contents of messages unnoticed--say, by intercepting the letter and making changes before delivery. Through bit-flipping attacks or Man-in-the-Middle techniques, attackers could modify commands or data in transit, potentially causing serious harm to systems or users.

Authentication (Authenticity)

Authentication confirms the identity of the party who has created the message, the same as the signature on a letter would provide. If this doesn't exist, anyone can impersonate anybody. Think of it this way: suppose an e-mail has come from your bank, but there's nothing that guarantees it is from your bank. The message will allow replay attacks - whereby an old message may be replayed as a new message, or spoofing attacks in which the attacker may pose as legitimate users or systems.

Why Are All Three Components Necessary?

Security in communication is like a three-legged stool: take away any leg, and the whole thing becomes unstable. Each component of AEAD addresses a different security concern, and they work together to provide comprehensive protection. Without confidentiality, your private conversations become public. Without integrity, messages can be altered in transit. Without authenticity, you can't trust who you're talking to. Only by providing all three components can we truly secure the communication channel over untrusted networks.

**4b.**  
 A very important security property, enabling protection of the past communications in case of later compromise of long-term keys, is forward security or forward secrecy. It relies on the usage of ephemeral session keys, generated independently of the long-term private key, for encryption. If an attacker manages to obtain the long-term key, he cannot decrypt the previously captured sessions of communication.

TLS Implementation

TLS achieves forward secrecy with two widely deployed ephemeral key exchange algorithms: First, Diffie-Hellman Ephemeral (DHE) produces fresh key pairs for every session, and the shared session key is computed via the Diffie-Hellman protocol. Second, Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) is a more efficient variant of DHE; instead of using modular arithmetic, it employs elliptic curve cryptography for these ephemeral exchanges.

The importance of forward security cannot be overstated. Without it, an attacker who compromises a server's private key would gain the ability to decrypt all previously recorded communications. This poses significant risks for sensitive data such as login credentials, financial transactions, and classified communications.

Scenarios Where Forward Security May Not Be Appropriate

Despite its benefits, forward security may not be suitable in certain contexts:

Performance-critical systems may find this computational overhead of generating ephemeral keys for each session prohibitive, particularly in resource-constrained environments like embedded devices or high-traffic servers. Legal and compliance requirements of certain industries would demand the ability to decrypt past communications for auditing purposes. An organization dealing in finance or legal operations may be required by law to maintain decrypted records for regulatory purposes.

Low-Sensitivity Data: Ephemeral social media updates or short-lived chat messages do not require the additional security of forward secrecy, as the information has no long-term sensitivity.

**4c.**   
Authentication methods can be categorized into three fundamental approaches, each using distinct mechanisms to verify identity claims. Here's an analysis of each method:

Something You Know (Knowledge-Based Authentication) This method relies on secret information like passwords or PINs. The verification process uses cryptographic hashing - the system stores hashed passwords and validates authentication by comparing hash values. To enhance security, salted hashing is employed to prevent rainbow table attacks. While this method is straightforward to implement and widely understood, it can be vulnerable to brute force attacks, phishing, and database breaches, particularly when passwords are weak or poorly managed.

Something You Have (Possession-Based Authentication) This approach verifies identity through possession of physical or digital tokens. It employs public-key cryptography, where users hold private keys that correspond to public keys known to the system. Time-Based One-Time Passwords (TOTP) are also common, generating synchronized temporary codes for verification. The authentication process involves presenting and validating these tokens. While this method is more resistant to remote attacks, it can be compromised if the token is stolen or duplicated.

Something You Are (Biometric Authentication) Biometric authentication uses unique physical or behavioral characteristics for identification. It involves sophisticated feature extraction and matching processes, with secure multi-party computation ensuring protected comparisons. The system processes biometric samples into templates and performs matches against stored data using confidence thresholds. While this method is difficult to fake and can't be shared, it faces challenges with spoofing attacks and the permanence of biometric data - once compromised, biometric identifiers cannot be easily changed.

Combined Approaches Multi-Factor Authentication (MFA) strengthens security by combining multiple authentication methods. By requiring different types of proof (such as a password plus a token), MFA significantly increases the difficulty of unauthorized access. This layered approach compensates for the weaknesses of individual methods while leveraging their respective strengths.

**5.**

The preparation phase would begin with thorough reconnaissance of Alice and Bob to understand their communication patterns, devices, and network infrastructure. This would involve mapping their internet providers and communication routes through passive surveillance and open-source intelligence. Device profiling would identify Alice and Bob’s operating systems and versions through metadata analysis. Supply chain vulnerabilities would need to be investigated, along with potential zero-day exploits in the communication tools being used. This phase would require cybersecurity experts, OSINT analysts, and reverse engineers, taking 2-3 months and substantial funding.

**KEEP IN MIND THAT I HAVE NO IDEA HOW MUCH THIS WOULD COST REALLY SO THESE ARE NUMBERS I PULLED FROM INSIDE MY HEAD.**

**Resources required:**

* People: Cybersecurity experts, OSINT analysts, reverse engineers.
* Time: 2-3 months for detailed reconnaissance and zero-day acquisition.
* Budget: $500,000+ (for zero-day purchases, surveillance infrastructure, and expertise).

The attack phase would focus on exploiting identified vulnerabilities through various vectors. Man-in-the-Middle attacks might target implementation flaws, while device exploitation could involve phishing or malicious updates. Active network intrusion and key exchange manipulation would be attempted to capture traffic for offline analysis. This would require red team operators and exploit developers working over several months.

**Resources Required**:

* **People**: Red team operators, network penetration testers, exploit developers.
* **Time**: Weeks to months, depending on the complexity of the attack.
* **Budget**: $1,000,000+ (for zero-day delivery, malware deployment, and network access).

The post-exploitation phase would concentrate on maintaining persistent access to Alice and Bob’s communication while avoiding detection. This would involve backdoor installation, traffic analysis, monitoring of key updates, and secure data exfiltration. Ongoing monitoring and analysis would be needed, requiring dedicated cyber analysts and malware operators.

**Resources Required**:

* **People**: Cyber analysts, malware operators.
* **Time**: Ongoing (requires continuous monitoring).
* **Budget**: $500,000 annually (to maintain infrastructure and cover operational costs).

However, significant challenges would make this operation difficult. Modern encryption standards make direct decryption nearly impossible. Geographic separation of targets complicates physical access. Zero-day vulnerabilities may not exist or could be quickly patched. Detection risk remains high throughout.