Buffer Overflow

Buffer overflows are among the most well-known types of software vulnerabilities. A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold, or when a program attempts to put data in a memory area outside of the boundaries of a buffer. The simplest type of error, and the most common cause of buffer overflows, is the "classic" case in which the program copies the buffer without restricting how much is copied. Other variants exist, but the existence of a classic overflow strongly suggests that the programmer is not considering even the most basic of security protections.

The most popular web app languages (e.g., Java) protect against this type of security vulnerability. But many mobile apps are written in C/C++, a language family that’s famously vulnerable to buffer overflow and incorrect buffer size calculation.

**How to Prevent:**

* Avoid C/C++. Those programming languages prone to buffer overflow. If possible, use another language such as Java, Python or COBOL that do not allow direct memory access.
* Address space layout randomization (ASLR) randomly arranges the location of different parts of the program in memory, working together with virtual memory management.
* Avoid standard library function that are not bounds checked. Bounds checking in abstract data type libraries can reduce the occurrence and impact of buffer overflows.
* Scan your code for buffer overflow vulnerabilities with Static Application Analysis tooll.

Issues with logging

If you control access to your log files (e.g., the ones on your web server), thorough logging is a plus. It can help you detect an attack and determine its scope and potential damage after the fact. But sometimes you don’t control access to the log files (e.g., the ones on your users’ devices). In that case, the “more is more” paradigm doesn’t necessarily apply. All the information that helps you trace security vulnerabilities can help an attacker do it too especially if it’s unencrypted.

* **Insufficient Logging & Monitoring:**

Exploitation of insufficient logging and monitoring is the bedrock of nearly every major incident.  
Attackers rely on the lack of monitoring and timely response to achieve their goals without being detected. This issue is one of the Top 10 security weakness.

* **Insecure Data Storage:**

Insecure data storage vulnerabilities occur when development teams assume that users or malware will not have access to a mobile device’s filesystem and subsequent sensitive information in data-stores on the device. Filesystems are easily accessible. Organizations should expect a malicious user or malware to inspect sensitive data stores. Usage of poor encryption libraries is to be avoided. Rooting or jailbreaking a mobile device circumvents any encryption protections. When data is not protected properly, specialized tools are all that is needed to view application data.

**How to Prevent:**

* Ensure all login, access control failures, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts and held for sufficient time to allow delayed forensic analysis.
* Ensure that logs are generated in a format that can be easily consumed by a centralized log management solution.  
  - Ensure high-value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append-only database tables or similar.
* Establish effective monitoring and alerting such that suspicious activities are detected and responded to in a timely fashion.

Failure to protect sensitive data

Loss of connectivity threatens customers temporarily. But loss of sensitive data threatens customers for the rest of their lives—and can have severe consequences for your business. Admittedly, protecting data in transit is hard. But at least you can use strong encryption on all data at rest (i.e., stored data).

* **Sensitive Data Exposure:**Rather than directly attacking crypto, attackers steal keys, execute man-in-the-middle attacks, or steal clear text data off the server, while in transit, or from the user’s client, e.g. browser. A manual attack is generally required. Previously retrieved password databases could be brute forced by Graphics Processing Units (GPUs). Over the last few years, this has been the most common impactful attack. The most common flaw is simply not encrypting sensitive data. When crypto is employed, weak key generation and management, and weak algorithm, protocol and cipher usage is common, particularly for weak password hashing storage techniques. For data in transit, server-side weaknesses are mainly easy to detect, but hard for data at rest.

Failure frequently compromises all data that should have been protected. Typically, this information includes sensitive personal information (PII) data such as health records, credentials, personal data, and credit cards, which often require protection as defined by laws or regulations such as the EU GDPR or local privacy laws.

**How to Prevent:**

* Classify data processed, stored or transmitted by an application. Identify which data is sensitive according to privacy laws, regulatory requirements, or business needs.
* Apply controls as per the classification.
* Don’t store sensitive data unnecessarily. Discard it as soon as possible or use PCI DSS compliant tokenization or even truncation. Data that is not retained cannot be stolen.
* Make sure to encrypt all sensitive data at rest.
* Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key management.
* Encrypt all data in transit with secure protocols such as TLS with perfect forward secrecy (PFS) ciphers, cipher prioritization by the server, and secure parameters. Enforce encryption using directives like HTTP Strict Transport Security.
* Disable caching for response that contain sensitive data.
* Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor).