|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CVE Numbers | CWE Name | |  | | --- | |  | | Description | |  |  | | --- | |  | | Impact | |  | | --- | | Solutions/  Mitigations |  |  | | --- | |  | | Phase | |  | | --- | | MITRE  ATT&CK |  |  | | --- | |  | |
| |  | | --- | | CVE-2023-  27579 |  |  | | --- | |  | | CWE-697: Incorrect Comparison | |  | | --- | | The product incorrectly  Compares two entities  in a security-relevant  context, potentially  leading to weaknesses. |  |  | | --- | |  | | |  | | --- | | -Incorrect  Authentication,  - Incorrect  authorization,  - information  leakage |  |  | | --- | |  | | Use complete comparisons, proper validation, thorough testing | Implementation | |  | | --- | | T1078:  Valid Accounts |  |  | | --- | |  | |
| CVE-2023-25801 | CWE-415: Double Free | |  | | --- | | The product calls free()  twice on the same  memory address,  potentially leading  to modification of  unexpected memory  locations. |  |  | | --- | |  | | Modify Memory; Execute Unauthorized Code or Commands | |  | | --- | |  |   Ensure each allocation is freed only once, set pointer to NULL after freeing, use static analysis tools | Implementation | T1499: Resource Consumption |
| |  | | --- | | CVE-2023-  25676 |  |  | | --- | |  | | CWE-476: NULL Pointer Dereference | The product dereferences a pointer that it expects to be valid but is NULL. This can lead to crashes or unintended behaviors. | - DoS: Crash, Exit, or Restart Execute Unauthorized Code or Commands Read  Memory Modify Memory | - Check all pointers for NULL before use<br>- Use a programming language that prevents NULL dereferences<br>- Verify function return values before using them<br>- Initialize variables properly | |  | | --- | | Implementation |  |  | | --- | |  | | T1078: Valid Accounts |
| |  | | --- | | CVE-2023-  25675 |  |  | | --- | |  | | CWE-697: Incorrect Comparison | The product compares two entities in a security-relevant context, but the comparison is incorrect, which may lead to resultant weaknesses. | Technical Impact: Varies by Context | Ensure all relevant factors are included in comparisons • Use secure comparison functions • Validate all input data before comparison • Conduct thorough code reviews to identify incorrect | |  | | --- | | Implementation |  |  | | --- | |  | | T1211: Exploitation for Defense Evasion |
| CVE-2023-25674  CVE-2023-25674  (NAME)  (name issue )   |  | | --- | |  | | |  | | --- | | CWE-  476:  Null  Pointer  Derefe  rence | |  | |  |  |  | | --- | |  | | The product dereferences a pointer that it expects to be valid but is NULL. | **Availability**: DoS: Crash, Exit, or Restart. **Integrity**: Execute Unauthorized Code or Commands; Read Memory; Modify Memory. **Confidentiality**: Read Memory; Modify Memory. | **mplementation**: Check all pointers for NULL before dereferencing them. **Requirements**: Select a programming language that is not susceptible to these issues. **Implementation**: Check the results of all functions that return a value and verify that the value is non-null before acting upon it. | **Implementation** **Requirements** **Architecture and Design** | Not specifically |
| **Page 2**  CVE-2023-25668   |  | | --- | |  |  |  | | --- | |  | | CWE-122: Heap-based Buffer Overflow | A heap overflow condition is a buffer overflow where the buffer that can be overwritten is allocated in the heap portion of memory, generally using a routine such as malloc() | **Availability**: Buffer overflows can cause crashes and excessive resource consumption. **Integrity**: Can lead to arbitrary code execution or memory modification. **Confidentiality**: Risk of unauthorized memory access. **Access Control**: Potential for bypassing security mechanisms. | **Pre-design**: Use languages or compilers with automatic bounds checking. **Architecture and Design**: Employ abstraction libraries to reduce risky API usage. **Operation**: Utilize buffer overflow detection tools and implement bounds checking. **Build and Compilation**: Apply features like ASLR and PIE for memory randomization. | **Implementation Architecture and Design Operation Build and Compilation** | Not specifically |
| |  | | --- | | CVE-2023-  25667 |  |  | | --- | |  |  |  | | --- | |  |  |  | | --- | |  | | CWE-190: Integer Overflow or Wraparound | The product performs a calculation that can produce an integer overflow or wraparound when the logic assumes the resulting value will always be larger than the original value. This occurs when an integer value is incremented to a value that is too large to store in the associated representation | - Availability: DoS (Crash, Exit, Restart); Resource Consumption (Memory); Instability <br> - Integrity: Modify Memory <br> - Confidentiality: Execute Unauthorized Code; Bypass Protection Mechanism <br> - Other: Alter Execution Logic; DoS (CPU) | - **Requirements:** Ensure strict protocol definitions and conformance. <br> - **Language Selection:** Use languages or compilers with automatic bounds checking. <br> - **Architecture and Design:** Utilize vetted libraries or frameworks for safe integer handling. | Requirements Implementation - Architecture and Design | Not directly mapped |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CVE-2023-25664:   |  | | --- | |  |  |  | | --- | |  | | CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') | A buffer overflow condition occurs when data is copied to a buffer without ensuring that the buffer can hold it, leading to potential overwriting of adjacent memory. This can result in various issues such as crashes or unauthorized code execution. | **Phase: Requirements** <br> **Strategy: Language Selection** <br> Use a language that prevents buffer overflows or provides constructs to avoid them. <br> **Phase: Architecture and Design** | **Integrity**: Modify Memory; Execute Unauthorized Code or Commands <br> **Availability**: Modify Memory; DoS: Crash, Exit, or Restart; DoS: Resource Consumption (CPU) |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CVE-2023-25661   |  | | --- | |  |  |  | | --- | |  | | Cwe-20  Improper  Input  validation | Input validation is a frequently-used technique for checking potentially dangerous inputs to ensure they are safe for processing. When software does not validate input properly, an attacker can craft inputs that are unexpected, leading to unintended input being processed, resulting in altered control flow, resource control, or code execution. Input validation can be applied to raw data and metadata. Properties that need validation include size, type, syntax, consistency, conformance to rules, and more. Errors in deriving properties contribute to improper validation. Distinctions between input validation and output escaping are important |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | |  |  |  | | --- | |  | | CVE-2024-  37032 | |  |  | | --- | |  | | Improper Input Validation | Large language model (LLM) management tool does not validate the format of a digest value. | |  | | --- | | Path traversal |  |  | | --- | |  | | Ensure proper validation of all input formats, use secure coding practices. | Implementation | T1190 - Exploit Public-Facing Application |
| |  | | --- | | CVE-2022-  45918 |  |  | | --- | |  |  |  | | --- | |  |  |  | | --- | |  | | Improper Input Validation | Learning management tool debugger uses external input to locate session logs without proper path validation. | |  | | --- | | Path traversal |  |  | | --- | |  | | Validate and sanitize all input paths, implement access controls to limit file system access. | Implementation | T1059.001 - Command-Line Interface |
| CVE-2021-30860   |  | | --- | |  |  |  | | --- | |  | | Improper Input Validation | Integer overflow in mobile OS due to improper input validation. | Arbitrary code execution | Implement proper input validation, use integer overflow checks, and handle exceptions appropriately. | Implementation | T1210 - Exploitation of Remote Services |
| |  | | --- | | CVE-2021-  22205 |  |  | | --- | |  |  |  | | --- | |  |  |  | | --- | |  | | Improper Input Validation | Bypass of a validation step leading to eval injection. | |  | | --- | | Code injection |  |  | | --- | |  | | Use strong input validation, avoid use of eval-like functions, implement least privilege principles. | Implementation | T1059.006 - Command and Scripting Interpreter |
| CVE-2021-21220   |  | | --- | |  |  |  | | --- | |  | | Improper Input Validation | Insufficient input validation in browser allows heap corruption. | |  | | --- | | Memory  corruption |  |  | | --- | |  | | Perform strict input validation, utilize secure coding practices to avoid heap corruption vulnerabilities. | Implementation | T1068 - Exploitation for Privilege Escalation |
| |  | | --- | | CVE-2020-  9054 |  |  | | --- | |  |  |  | | --- | |  |  |  | | --- | |  | | Improper Input Validation | Improper validation of username parameter leads to OS command injection. | |  | | --- | | Command  injection |  |  | | --- | |  | | Implement strict input validation, sanitize user inputs, use parameterized queries. | Implementation | T1059.003 - Command-Line Interf |
| //  CVE-2022-41902   |  | | --- | |  |  |  | | --- | |  | | Out-of-bounds Write | This vulnerability involves writing data past the end of the intended buffer, which can lead to arbitrary code execution, system crashes, or corruption of data. | Memory Corruption: | Apply security patches provided by the vendor, conduct rigorous input validation, use safe memory management practices, and perform code audits. | Exploitation | T1068: Exploitation for Privilege Escalation |
| CVE-2022-36004   |  | | --- | |  |  |  | | --- | |  | | Reachable Assertion | An assertion failure occurs in the XYZ software when handling malformed or unexpected input, causing the software to crash. | Denial of Service (DoS) | Improve input validation to handle unexpected or malformed data gracefully and avoid relying on assertions for critical checks. | |  | | --- | | Implementation |  |  | | --- | |  | |  |
| CVE-2022-35970   |  | | --- | |  |  |  | | --- | |  | | |  | | --- | |  |  |  | | --- | | Improper  Input  Validation | | The product receives input or data but does not validate or incorrectly validates the input, leading to unsafe or incorrect processing. | Denial of Service (DoS): Crash, excessive resource consumption (CPU, memory)  - Confidentiality: Reading sensitive data  - Integrity: Modifying data or executing unauthorized code | - **Architecture and Design**: Use language-theoretic security techniques to define acceptable inputs with formal "recognizers"  - **Architecture and Design**: Utilize input validation frameworks such as OWASP ESAPI  - **Implementation**: Assume all input is malicious, use whitelisting  - **Implementation**: Validate input after combining data from multiple sources  - **Implementation**: Validate input when crossing language boundaries  - **Implementation**: Convert input types directly and validate after conversion  - **Implementation**: Ensure proper decoding and canonicalization of inputs | - **Architecture and Design**: Ensure client-side checks are replicated server-side to avoid CWE-602  - **Implementation**: Ensure consistent character encoding between components |  |
| CVE-2022-23591   |  | | --- | |  |  |  | | --- | |  | | Uncontrolled Recursion | The product does not properly control the amount of recursion, leading to excessive consumption of resources, such as memory or the program stack. | Denial of Service (DoS): Resource consumption (CPU, memory, stack memory)  - Confidentiality: Potential leakage of application data if the process/thread is killed and reports errors | Denial of Service (DoS): Resource consumption (CPU, memory, stack memory)  - Confidentiality: Potential leakage of application data if the process/thread is killed and reports errors | **Implementation**: Always test recursion depth and handle errors properly | -- |
| CVE-2022-29216   |  | | --- | |  |  |  | | --- | |  | | Improper Control of Generation of Code ('Code Injection') | The product constructs a part of code using externally-influenced input but does not properly neutralize special elements that could modify the syntax or behavior of the code. | - **Bypass of Protection Mechanisms**: Can control authentication.  - **Privilege Escalation or Identity Assumption**: Access to resources that the attacker should not access.  - **Execution of Unauthorized Code**: Leads to data integrity issues and execution of arbitrary code.  - **Hiding Activities**: Actions performed by injected code may go unlogged. | - **Refactor Code**: Avoid dynamically generating code.  - **Isolated Environment**: Use sandboxes or "jails" to restrict code execution.  - **Input Validation**: Treat all inputs as malicious, use strict allowlists.  - **Static Analysis**: Use tools to detect vulnerabilities in code.  - **Dynamic Testing**: Employ tools for fuzz testing, robustness testing, and fault injection.  - **Environment Hardening**: Use techniques like automatic taint propagation. | **Architecture and Design**: Refactor code, use isolated environments.  - **Implementation**: Input validation, environment hardening.  - **Testing**: Static and dynamic analysis.  - **Operation**: Use secure compilation practices and hardened environments. | - **T1071**: Application Layer Protocols  - **T1203**: Exploitation for Client Execution |
| CVE-2022-23584   |  | | --- | |  |  |  | | --- | |  | | Use After Free | The product reuses or references memory after it has been freed. This can lead to accessing invalid memory if the freed memory has been reallocated and used by another part of the program. | - **Memory Corruption**: Previously freed memory might corrupt data if it’s allocated elsewhere.  - **Denial of Service (DoS)**: Crashes or restarts when invalid data is used.  - **Arbitrary Code Execution**: If memory is reallocated and contains function pointers, this might allow execution of arbitrary code. | **Language Selection**: Use languages with automatic memory management.  - **Set Pointers to NULL**: After freeing memory, set pointers to NULL to prevent use-after-free.  - **Defensive Programming**: Avoid reusing pointers after freeing memory.  - **Memory Safety Tools**: Use tools or libraries that detect use-after-free errors | - **Architecture and Design**: Select appropriate programming languages and design to avoid memory management issues.  - **Implementation**: Set pointers to NULL after freeing.  - **Testing**: Employ memory safety tools and rigorous testing. | - **T1071**: Application Layer Protocols  - **T1203**: Exploitation for Client Execution |
| |  | | --- | |  |  |  | | --- | |  | |  |  |  |  |  |  |
| |  | | --- | |  |  |  | | --- | |  | |  |  |  |  |  |  |
| |  | | --- | |  |  |  | | --- | |  | |  |  |  |  |  |  |
| |  | | --- | |  |  |  | | --- | |  | |  |  |  |  |  |  |
| |  | | --- | |  |  |  | | --- | |  | |  |  |  |  |  |  |