# CWE Detail – CWE-259

## Description

The product contains a hard-coded password, which it uses for its own inbound authentication or for outbound communication to external components.

## Extended Description

There are two main variations of a hard-coded password: Inbound: the product contains an authentication mechanism that checks for a hard-coded password. Outbound: the product connects to another system or component, and it contains a hard-coded password for connecting to that component.

## Threat-Mapped Scoring

Score: 3.0

Priority: P2 - Serious (High)

## Observed Examples (CVEs)

**•** CVE-2022-29964: Distributed Control System (DCS) has hard-coded passwords for local shell access

**•** CVE-2021-37555: Telnet service for IoT feeder for dogs and cats has hard-coded password [REF-1288]

**•** CVE-2021-35033: Firmware for a WiFi router uses a hard-coded password for a BusyBox shell, allowing bypass of authentication through the UART port

## Modes of Introduction

**•** Implementation: REALIZATION: This weakness is caused during implementation of an architectural security tactic.

**•** Architecture and Design: N/A

## Common Consequences

**•** Impact: Gain Privileges or Assume Identity — Notes: If hard-coded passwords are used, it is almost certain that malicious users can gain access through the account in question.

**•** Impact: Gain Privileges or Assume Identity, Hide Activities, Reduce Maintainability — Notes: A hard-coded password typically leads to a significant authentication failure that can be difficult for the system administrator to detect. Once detected, it can be difficult to fix, so the administrator may be forced into disabling the product entirely.

## Potential Mitigations

**•** Architecture and Design: For outbound authentication: store passwords outside of the code in a strongly-protected, encrypted configuration file or database that is protected from access by all outsiders, including other local users on the same system. Properly protect the key (CWE-320). If you cannot use encryption to protect the file, then make sure that the permissions are as restrictive as possible. (Effectiveness: N/A)

**•** Architecture and Design: For inbound authentication: Rather than hard-code a default username and password for first time logins, utilize a "first login" mode that requires the user to enter a unique strong password. (Effectiveness: N/A)

**•** Architecture and Design: Perform access control checks and limit which entities can access the feature that requires the hard-coded password. For example, a feature might only be enabled through the system console instead of through a network connection. (Effectiveness: N/A)

**•** Architecture and Design: For inbound authentication: apply strong one-way hashes to your passwords and store those hashes in a configuration file or database with appropriate access control. That way, theft of the file/database still requires the attacker to try to crack the password. When receiving an incoming password during authentication, take the hash of the password and compare it to the hash that you have saved. Use randomly assigned salts for each separate hash that you generate. This increases the amount of computation that an attacker needs to conduct a brute-force attack, possibly limiting the effectiveness of the rainbow table method. (Effectiveness: N/A)

**•** Architecture and Design: For front-end to back-end connections: Three solutions are possible, although none are complete. The first suggestion involves the use of generated passwords which are changed automatically and must be entered at given time intervals by a system administrator. These passwords will be held in memory and only be valid for the time intervals. Next, the passwords used should be limited at the back end to only performing actions valid for the front end, as opposed to having full access. Finally, the messages sent should be tagged and checksummed with time sensitive values so as to prevent replay style attacks. (Effectiveness: N/A)

## Applicable Platforms

**•** None (Class: Not Language-Specific, Prevalence: Undetermined)

## Demonstrative Examples

**•** This is an example of an external hard-coded password on the client-side of a connection. This code will run successfully, but anyone who has access to it will have access to the password. Once the program has shipped, there is no going back from the database user "scott" with a password of "tiger" unless the program is patched. A devious employee with access to this information can use it to break into the system. Even worse, if attackers have access to the bytecode for application, they can use the javap -c command to access the disassembled code, which will contain the values of the passwords used. The result of this operation might look something like the following for the example above:

**•** Every instance of this program can be placed into diagnostic mode with the same password. Even worse is the fact that if this program is distributed as a binary-only distribution, it is very difficult to change that password or disable this "functionality."

**•** This Java example shows a properties file with a cleartext username / password pair.

**•** Multiple vendors used hard-coded credentials in their OT products.

## Notes

**•** Maintenance: It might be appropriate to split this entry into an inbound variant and an outbound variant. These variants are likely to have different consequences, detectability, etc., although such differences are not suitable for a split. More importantly, from a vulnerability theory perspective, they might be characterized as different behaviors. The difference is in where the hard-coded password is stored - on the component performing the authentication, or the component that is connecting to the external component that requires authentication. However, as with many weaknesses, the "vulnerability topology" should not be regarded as important enough for splits. For example, separate weaknesses do not exist for client-to-server buffer overflows versus server-to-client buffer overflows.

**•** Other: In the Inbound variant, a default administration account may be created, and a simple password is hard-coded into the product and associated with that account. This hard-coded password is the same for each installation of the product, and it usually cannot be changed or disabled by system administrators without manually modifying the program, or otherwise patching the product. If the password is ever discovered or published (a common occurrence on the Internet), then anybody with knowledge of this password can access the product. Finally, since all installations of the product will have the same password, even across different organizations, this enables massive attacks such as worms to take place. The Outbound variant can apply to front-end systems that authenticate with a back-end service. The back-end service may require a fixed password that can be discovered easily. The programmer may simply hard-code those back-end credentials into the front-end product. Any user of that program may be able to extract the password. Client-side systems with hard-coded passwords pose even more of a threat, since the extraction of a password from a binary is usually very simple.