# CWE Detail – CWE-798

## Description

The product contains hard-coded credentials, such as a password or cryptographic key.

## Extended Description

There are two main variations: Inbound: the product contains an authentication mechanism that checks the input credentials against a hard-coded set of credentials. In this variant, a default administration account is 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. It can also be difficult for the administrator to detect. Outbound: the product connects to another system or component, and it contains hard-coded credentials for connecting to that component. This variant applies to front-end systems that authenticate with a back-end service. The back-end service may require a fixed password that can be easily discovered. The programmer may simply hard-code those back-end credentials into the front-end product.

## Threat-Mapped Scoring

Score: 3.0

Priority: P2 - Serious (High)

## Observed Examples (CVEs)

**•** CVE-2022-29953: Condition Monitor firmware has a maintenance interface with hard-coded credentials

**•** CVE-2022-29960: Engineering Workstation uses hard-coded cryptographic keys that could allow for unathorized filesystem access and privilege escalation

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

**•** CVE-2022-30997: Programmable Logic Controller (PLC) has a maintenance service that uses undocumented, hard-coded credentials

**•** CVE-2022-30314: Firmware for a Safety Instrumented System (SIS) has hard-coded credentials for access to boot configuration

**•** CVE-2022-30271: Remote Terminal Unit (RTU) uses a hard-coded SSH private key that is likely to be used in typical deployments

**•** 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

**•** CVE-2012-3503: Installation script has a hard-coded secret token value, allowing attackers to bypass authentication

**•** CVE-2010-2772: SCADA system uses a hard-coded password to protect back-end database containing authorization information, exploited by Stuxnet worm

**•** CVE-2010-2073: FTP server library uses hard-coded usernames and passwords for three default accounts

**•** CVE-2010-1573: Chain: Router firmware uses hard-coded username and password for access to debug functionality, which can be used to execute arbitrary code

**•** CVE-2008-2369: Server uses hard-coded authentication key

**•** CVE-2008-0961: Backup product uses hard-coded username and password, allowing attackers to bypass authentication via the RPC interface

**•** CVE-2008-1160: Security appliance uses hard-coded password allowing attackers to gain root access

**•** CVE-2006-7142: Drive encryption product stores hard-coded cryptographic keys for encrypted configuration files in executable programs

**•** CVE-2005-3716: VoIP product uses hard-coded public credentials that cannot be changed, which allows attackers to obtain sensitive information

**•** CVE-2005-3803: VoIP product uses hard coded public and private SNMP community strings that cannot be changed, which allows remote attackers to obtain sensitive information

**•** CVE-2005-0496: Backup product contains hard-coded credentials that effectively serve as a back door, which allows remote attackers to access the file system

## Related Attack Patterns (CAPEC)

* CAPEC-191
* CAPEC-70

## Attack TTPs

**•** T1078.001: Default Accounts (Tactics: defense-evasion, persistence, privilege-escalation, initial-access)

**•** T1552.001: Credentials In Files (Tactics: credential-access)

## Modes of Introduction

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

## Common Consequences

**•** Impact: Bypass Protection Mechanism — Notes:

**•** Impact: Read Application Data, Gain Privileges or Assume Identity, Execute Unauthorized Code or Commands, Other — Notes:

## Potential Mitigations

**•** Architecture and Design: For outbound authentication: store passwords, keys, and other credentials 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 [REF-7]. In Windows environments, the Encrypted File System (EFS) may provide some protection. (Effectiveness: N/A)

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

**•** Architecture and Design: If the product must contain hard-coded credentials or they cannot be removed, perform access control checks and limit which entities can access the feature that requires the hard-coded credentials. 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 using passwords: apply strong one-way hashes to 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 handling an incoming password during authentication, take the hash of the password and compare it to the saved hash. Use randomly assigned salts for each separate hash that is generated. 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 or keys that 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 or keys 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."

**•** The cryptographic key is within a hard-coded string value that is compared to the password. It is likely that an attacker will be able to read the key and compromise the system.

**•** 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: The Taxonomy\_Mappings to ISA/IEC 62443 were added in CWE 4.10, but they are still under review and might change in future CWE versions. These draft mappings were performed by members of the "Mapping CWE to 62443" subgroup of the CWE-CAPEC ICS/OT Special Interest Group (SIG), and their work is incomplete as of CWE 4.10. The mappings are included to facilitate discussion and review by the broader ICS/OT community, and they are likely to change in future CWE versions.