# CWE Detail – CWE-328

## Description

The product uses an algorithm that produces a digest (output value) that does not meet security expectations for a hash function that allows an adversary to reasonably determine the original input (preimage attack), find another input that can produce the same hash (2nd preimage attack), or find multiple inputs that evaluate to the same hash (birthday attack).

## Extended Description

A hash function is defined as an algorithm that maps arbitrarily sized data into a fixed-sized digest (output) such that the following properties hold: The algorithm is not invertible (also called "one-way" or "not reversible") The algorithm is deterministic; the same input produces the same digest every time Building on this definition, a cryptographic hash function must also ensure that a malicious actor cannot leverage the hash function to have a reasonable chance of success at determining any of the following: the original input (preimage attack), given only the digest another input that can produce the same digest (2nd preimage attack), given the original input a set of two or more inputs that evaluate to the same digest (birthday attack), given the actor can arbitrarily choose the inputs to be hashed and can do so a reasonable amount of times What is regarded as "reasonable" varies by context and threat model, but in general, "reasonable" could cover any attack that is more efficient than brute force (i.e., on average, attempting half of all possible combinations). Note that some attacks might be more efficient than brute force but are still not regarded as achievable in the real world. Any algorithm that does not meet the above conditions will generally be considered weak for general use in hashing. In addition to algorithmic weaknesses, a hash function can be made weak by using the hash in a security context that breaks its security guarantees. For example, using a hash function without a salt for storing passwords (that are sufficiently short) could enable an adversary to create a "rainbow table" [REF-637] to recover the password under certain conditions; this attack works against such hash functions as MD5, SHA-1, and SHA-2.

## Threat-Mapped Scoring

Score: 3.25

Priority: P2 - Serious (High)

## Observed Examples (CVEs)

**•** CVE-2022-30320: Programmable Logic Controller (PLC) uses a protocol with a cryptographically insecure hashing algorithm for passwords.

**•** CVE-2005-4900: SHA-1 algorithm is not collision-resistant.

**•** CVE-2020-25685: DNS product uses a weak hash (CRC32 or SHA-1) of the query name, allowing attacker to forge responses by computing domain names with the same hash.

**•** CVE-2012-6707: blogging product uses MD5-based algorithm for passwords.

**•** CVE-2019-14855: forging of certificate signatures using SHA-1 collisions.

**•** CVE-2017-15999: mobile app for backup sends SHA-1 hash of password in cleartext.

**•** CVE-2006-4068: Hard-coded hashed values for username and password contained in client-side script, allowing brute-force offline attacks.

## Related Attack Patterns (CAPEC)

* CAPEC-461
* CAPEC-68

## Attack TTPs

**•** T1553.002: Code Signing (Tactics: defense-evasion)

## Modes of Introduction

**•** Architecture and Design: COMMISSION: This weakness refers to an incorrect design related to an architectural security tactic.

## Common Consequences

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

## Potential Mitigations

**•** Architecture and Design: Use an adaptive hash function that can be configured to change the amount of computational effort needed to compute the hash, such as the number of iterations ("stretching") or the amount of memory required. Some hash functions perform salting automatically. These functions can significantly increase the overhead for a brute force attack compared to intentionally-fast functions such as MD5. For example, rainbow table attacks can become infeasible due to the high computing overhead. Finally, since computing power gets faster and cheaper over time, the technique can be reconfigured to increase the workload without forcing an entire replacement of the algorithm in use. Some hash functions that have one or more of these desired properties include bcrypt [REF-291], scrypt [REF-292], and PBKDF2 [REF-293]. While there is active debate about which of these is the most effective, they are all stronger than using salts with hash functions with very little computing overhead. Note that using these functions can have an impact on performance, so they require special consideration to avoid denial-of-service attacks. However, their configurability provides finer control over how much CPU and memory is used, so it could be adjusted to suit the environment's needs. (Effectiveness: High)

## Applicable Platforms

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

## Demonstrative Examples

**•** This code relies exclusively on a password mechanism (CWE-309) using only one factor of authentication (CWE-308). If an attacker can steal or guess a user's password, they are given full access to their account. Note this code also uses SHA-1, which is a weak hash (CWE-328). It also does not use a salt (CWE-759).

**•** At least one OT product used weak hashes.

**•** The vulnerable code shows an incorrect implementation of the HMAC authentication where it only uses the least significant 32 bits of the secret message for the authentication (the remaining 480 bits are hard coded as zeros). As a result, the system is susceptible to brute-force attacks where the attacker only needs to determine 32 bits of the secret message instead of 512 bits, weakening the cryptographic protocol.

## Notes

**•** Maintenance: Since CWE 4.4, various cryptography-related entries including CWE-328 have been slated for extensive research, analysis, and community consultation to define consistent terminology, improve relationships, and reduce overlap or duplication. As of CWE 4.6, this work is still ongoing.