# CWE Detail – CWE-760

## Description

The product uses a one-way cryptographic hash against an input that should not be reversible, such as a password, but the product uses a predictable salt as part of the input.

## Extended Description

This makes it easier for attackers to pre-compute the hash value using dictionary attack techniques such as rainbow tables, effectively disabling the protection that an unpredictable salt would provide. It should be noted that, despite common perceptions, the use of a good salt with a hash does not sufficiently increase the effort for an attacker who is targeting an individual password, or who has a large amount of computing resources available, such as with cloud-based services or specialized, inexpensive hardware. Offline password cracking can still be effective if the hash function is not expensive to compute; many cryptographic functions are designed to be efficient and can be vulnerable to attacks using massive computing resources, even if the hash is cryptographically strong. The use of a salt only slightly increases the computing requirements for an attacker compared to other strategies such as adaptive hash functions. See CWE-916 for more details.

## Threat-Mapped Scoring

Score: 3.25

Priority: P2 - Serious (High)

## Observed Examples (CVEs)

**•** CVE-2008-4905: Blogging software uses a hard-coded salt when calculating a password hash.

**•** CVE-2002-1657: Database server uses the username for a salt when encrypting passwords, simplifying brute force attacks.

**•** CVE-2001-0967: Server uses a constant salt when encrypting passwords, simplifying brute force attacks.

**•** CVE-2005-0408: chain: product generates predictable MD5 hashes using a constant value combined with username, allowing authentication bypass.

## Modes of Introduction

**•** Implementation: REALIZATION: This weakness is caused during implementation of 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)

**•** Implementation: If a technique that requires extra computational effort can not be implemented, then for each password that is processed, generate a new random salt using a strong random number generator with unpredictable seeds. Add the salt to the plaintext password before hashing it. When storing the hash, also store the salt. Do not use the same salt for every password. (Effectiveness: Limited)

## Notes

**•** Maintenance: As of CWE 4.5, terminology related to randomness, entropy, and
 predictability can vary widely. Within the developer and other
 communities, "randomness" is used heavily. However, within
 cryptography, "entropy" is distinct, typically implied as a
 measurement. There are no commonly-used definitions, even within
 standards documents and cryptography papers. Future versions of
 CWE will attempt to define these terms and, if necessary,
 distinguish between them in ways that are appropriate for
 different communities but do not reduce the usability of CWE for
 mapping, understanding, or other scenarios.