Insecure Cryptographic Storage

Troy Hunt
troyhunt.com

pluralsight
hardcore developer training

Outline

- How OWASP views the risk
- Performing an attack against weak password storage
- Understanding password storage and hashing
- How salt helps protect hashes (and where it’s still vulnerable)
- Creating stronger hashes
- Asymmetric encryption, symmetric encryption and DPAPI
- What *isn’t* cryptographic storage – myths and misconceptions

OWASP overview and risk rating

Threat Agents
—
Consider the users of your system. Would they like to gain access to protected data they aren't authorised for? What about internal administrators?

Understanding password storage

- **When we talk about cryptographic storage in web applications, it's most commonly about password storage**
- **There are three common password storage mechanisms:**
    - Plain text (no cryptography)
    - Encrypted
    - Hashed

Hashing != encryption

- **Although the terms are used interchangeably, they're fundamentally different**
- **Encryption is a reversible process**
  - For password storage, it usually involved a single private key to both encrypt and decrypt (symmetric encryption)
- **Hashing is a one-way process**
  - The ciphertext of a hashed password cannot be un-hashed

Understanding hashing

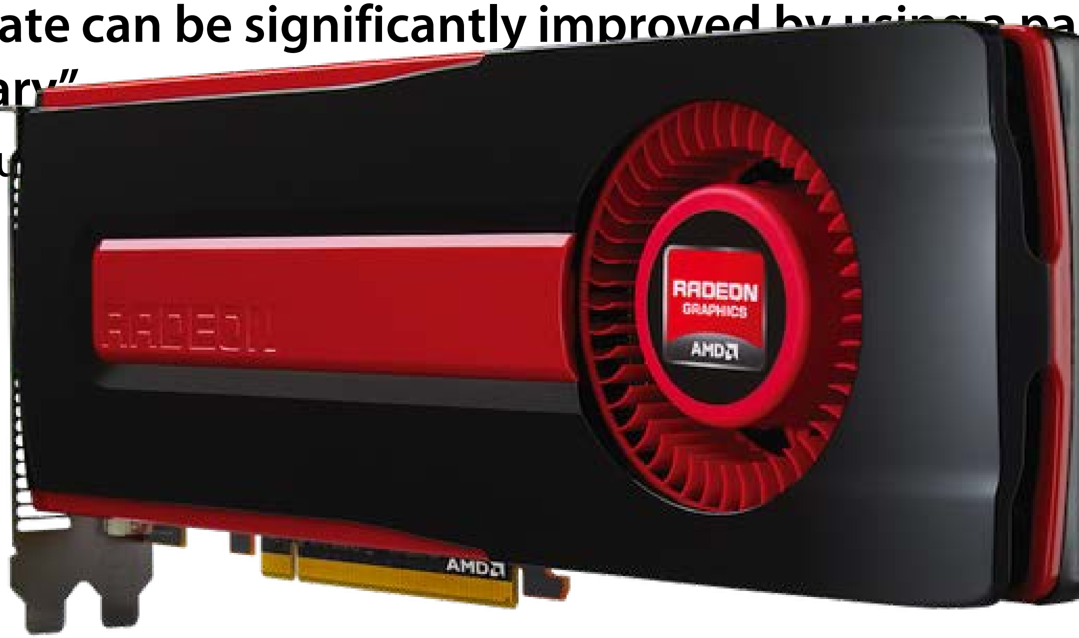
- **A hash is a keyless, one-way, deterministic algorithm**
  - Every time the same algorithm is used on the same password it produces the same result
  - When an account is created, the password is hashed and stored
  - When a user logs on, the provided password is hashed and compared to the one in the database
  - The password is never “un-hashed” *and never can be*
- **Hashing algorithms themselves are not usually cracked**
  - When hashed password is “cracked”, it’s normally because the plain text version has been hashed and compared via brute force
  - This may happen billions of times per second on consumer hardware

Understanding salt

- **The problem with a deterministic algorithm is that once the hash of a password is generated, it's *very* easy to compare it to a breached hash**
  - You can usually just Google it
- **Rainbow tables were invented to pre-compute hashes and rapidly compare them to breached accounts**
  - They're effective, but very large and slow to generate
- **A "salt" is a sequence of random bytes that can be added to a password before it's hashed**
  - It means the ciphertext of the same password is different when unique random salts are applied
  - The salt is stored in the database alongside the hashed password so that the process can be repeated when the user logs on

Brute forcing salted hashes

- **Salt makes it much harder to “crack” a hash, but it’s always just a matter of time**
- **If hashes can be created *fast* enough, hashed passwords may still be cracked quickly**
  - Modern GPUs are particularly effective at calculating hashes
  - How about 7.5 *billion* hashes per second on consumer hardware?!
- **The hit-rate can be significantly improved by using a password “dictionary”**
  - Althou U



Other password hashing considerations

- **Slowing down hashing is great for defending passwords…**
  - …but can cause adverse behaviour on the server
- **Finding a balance between performance and overhead may work well today…**
  - …but may not in years to come as computing power increases
  - By Moore's law, cracking hashes will be 8 times faster in 6 years time
- **Web servers hash with the CPU which is slow…**
  - …but attackers hash with the GPU which is fast
- **It's always possible to upgrade the hashing algorithm later…**
  - …but you still need existing passwords to be hashed using the old algorithm
- **These are all factors that need to be considered on a case by case basis, there's no "one size fits all" answer**

Understanding symmetric and asymmetric encryption

- **Symmetric encryption involves using a single private key for both encryption and decryption**
  - It's frequently used for encrypting data within an application where keys never need to be distributed externally
- **Asymmetric encryption has both a public and a private key**
  - Often this is referred to as public key encryption
  - We see asymmetric encryption being used in implementations such as SSL
- **In both cases, the encryption is reversible so unlike a hashing algorithm, the data can be retrieved**
  - Encryption is a two-way algorithm
  - Hashing is not encryption

The challenge of key management

- An encryption scheme will be rendered useless if the private key is disclosed
- Attacks on websites may lead to the compromise of the source files of the web server
- Frequently, private keys are improperly stored, such as in plain text in the web.config
- This is one of the major value propositions of hashing; there's no key management as it's not reversible

What *isn't* cryptographic storage?

- Frequently, simple string transformation processes are mistaken as a means of cryptography
- For example, ROT13:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
```

```
NOPQRSTUVWXYZABCDEFGHIJKLMnopqrstuvwxyzabcdefghijklm
```

Encoding isn't cryptography

- **Base64 is often mistaken as a suitable means of password cryptography**
  - It's simply a binary to text encoding scheme *and its reversible!*
- A password hash or salt may be Base64 encoded simply to allow it to be represented using ASCII characters…
  - …but this is after it has been securely hashed
- If in doubt, refer to Kerckhoffs's principle:

> *"One ought design systems under the assumption that the enemy will immediately gain full familiarity with them."*

Summary

- **Cryptographic storage is the last line of defence in a system**
  - It’s what saves us after a risk such as SQL injection is exploited
- **Password hashing is all about trying to slow the process down in order to increase the time and cost of cracking**
  - It’s not about being 100% secure, it’s about increasing difficulty
  - Creating higher workloads through approaches such as PBKDF2 and bcrypt is the best defence
  - Salt is still important, but not as useful as many people think
- **The problem with encryption remains key management**
  - DPAPI makes it easy to solve this problem (but introduces other problems)
- **Character rotation and encoding aren’t cryptographic!**
  - Remember Kerckhoffs's principle; are you happy for the enemy to view your implementation?