CVE Short: 2020-5229

By Trent Chismar, Ryan Brockway, Angela Castronuovo, Joshua Cominelli

Short description of the vulnerability and the product/system affected.

Opencast is open source software used for “automated video capture, management, and distribution”.[1] In versions of Opencast before 8.1, Opencast used the MD5 hash algorithm to store passwords. This is an issue because MD5 is not a secure algorithm, as collision attacks can be performed on MD5, and the MD5 hash can be broken quickly.[2] Opencast also salted the passwords with the user’s username, which would result in collisions with users with common usernames and passwords. If an attacker was able to gain access to the database where the password hashes are stored, the attacker could then figure out the user’s password by using a rainbow table and seeing if any of the calculated hashes match the password hashes from the database. It matters that this product has a password hashing vulnerability because an adversary could potentially gain access to a user’s account if they have access to the database where the password hashes are stored. All user’s of Opencast are affected by this vulnerability because they are all at risk of having their password stolen. This could then lead to other security issues for the user. If the user uses the same password across multiple sites, and the attacker gains access to it through this password hashing vulnerability, the user could have their other accounts compromised as well.

Who can launch the attack?

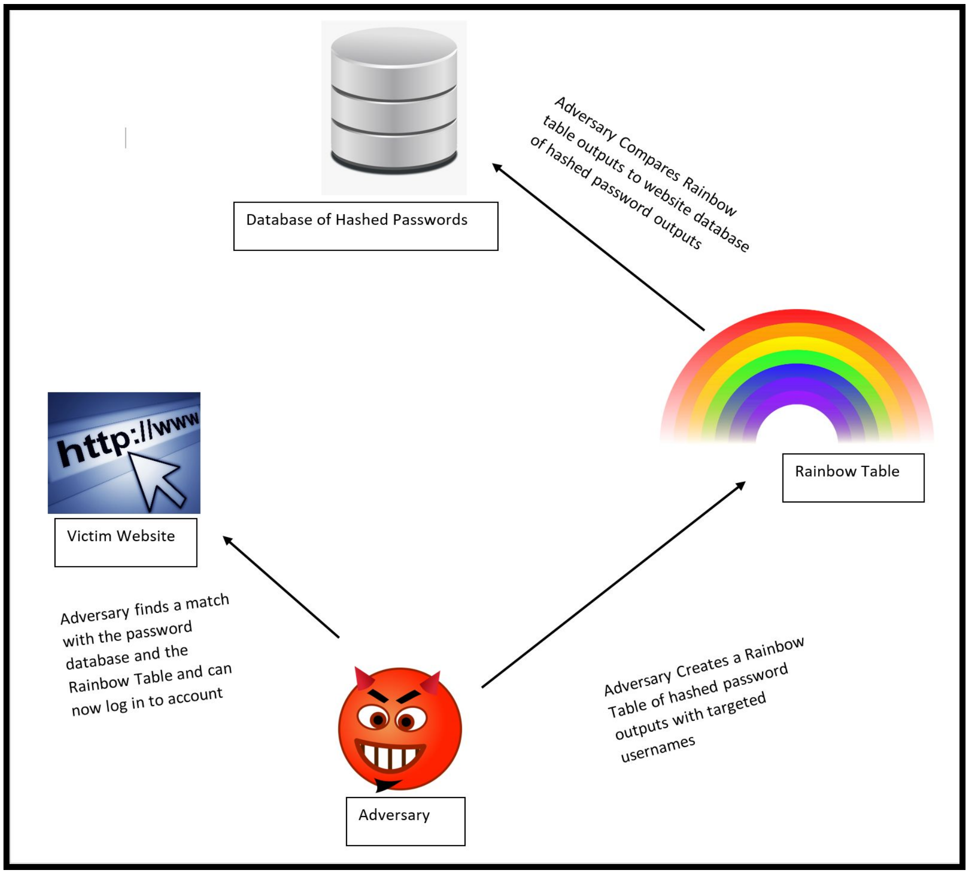
The attacker would need to somehow gain access to the file containing Opencast’s password hashes.[3] Since the MD5 code is readily available, it is easy for the attacker to create dictionaries of possible hashes salted with chosen usernames, either using popular usernames like ‘admin’ or targeted at a particular known username. Knowing that the password hashes are salted with the corresponding username makes the attacker’s job significantly easier, since he only has to create one dictionary for each particular username. The attacker could also use one of the many databases of reversed MD5 hash tables available for free online, meaning if the hash is already in these reversed hash tables, the adversary could execute this attack without having to do any coding themselves.

Threat Model:

[1] https://opencast.org

[2] https://en.wikipedia.org/wiki/MD5#Overview_of_security_issues

[3] https://nvd.nist.gov/vuln/detail/CVE-2020-5229


Database of Hashed Passwords
Adversary Compares Rainbow table outputs to website database of hashed password outputs
http://www
Victim Website
Rainbow Table
Adversary finds a match with the password database and the Rainbow Table and can now log in to account
Adversary Creates a Rainbow Table of hashed password outputs with targeted usernames
Adversary

The victim website uses a hash function for user passwords where the salt is the user's username instead of a random salt. This makes it easier for the adversary to perform a brute force attack on the database of passwords. To do this, the Adversary can create a Rainbow table[4], which is a table used for caching the outputs of hash functions. The adversary can then target certain usernames such as the admin user or certain big names like Bill Gates or Jeff Bezos[5] and use that username for the salt in the rainbow table. The Adversary then compares the outputs of the website's database of hashed passwords to the outputs in the rainbow table to find a match. Once a match is found, the adversary now has access to that account's username and password.

In the case of obtaining the admin's password, the adversary would gain a lot of control over the website. While, in the case of an adversary targeting someone like Bill Gates, the adversary could potentially obtain Bill Gates' bank password and transfer money out of the account.

[4] https://www.geeksforgeeks.org/understanding-rainbow-table-attack/

[5] https://smerity.com/articles/2012/salting_with_usernames.html#:~:text=By%20using%20usernames%20as%20salts,target%20system%20has%20been%20compromised.