Abertay University

An Investigation into the effectiveness of modern-day hashing algorithms

Jack Sime – 1801476@uad.ac.uk

Introduction to Security – CMP101.2019

BSc Ethical Hacking Year 1

2019/2020

.

Abstract

This report is an investigation into the hashing algorithms MD5, SHA3-256, and bcrypt. The hashes will be generated for all 3 algorithms in Python and PHP in Visual Studio Code and will be tested to see if they are crack able using consumer hardware and publicly available tools. From the investigation, it was found that md5 is not advisable to be used and was easily cracked whereas both SHA3-256 and bcrypt are generally quite secure in comparison and difficult for the average consumer to crack.

.

Contents

.

1 Introduction

1.1 Background

There are lots of different algorithms out there that can be used for hashing, all of them function differently, some easier than others to crack and some more widely used. One such algorithm that was very widely used but has become quite outdated in recent times and substantially insecure was md5. MD5 is a stronger version of the md4 algorithm and was created by Ronald Rivest. When md5 was first produced computers of that time would struggle with brute force like attack, but developments in computers severely reduced that time over the years from maybe years to break a password down to a mere few days (md5online, no date). Another fault of MD5 is that it's been proven to have collisions. A collision is when two different words have the same hash, this can lead to problems if you have an account that could theoretically be accessed with an incorrect password (Hawkes, Paddon and Rose, 2004).

A more modern algorithm such as SHA-3 designed in 2015 is considered far superior to md5. The SHA-3 algorithm uses a different sort of structure from md5 and SHA-1 and was handpicked from an open competition (KeccakTeam, No date). Both md5 and the SHA family of algorithms were designed to be able to hash a password quickly. This raises a concern as the faster you hash a password, the weaker the password gets, this occurs as the faster you can hash it also means the faster someone could try to brute force it by hashing millions of passwords a second through the use of every expanding modern technology and the fact that consumer-grade hardware is now at an all-time high in terms of performance. An algorithm known as bcrypt is designed to be the opposite of md5 and SHA, in terms of speed of hashing. Bcrypt is a slow algorithm as it has inherently designed to be used on passwords and so a slow algorithm also means that it's slower for people to try to crack it. Bcrypt also requires a salt to go with the password by default further increasing protection and can be purposely slowed down further to mitigate the effects of ever-increasing the processing speed available to people who may try to break it (D Arias, 2018). These are only a few of the well-known algorithms that could be used, and they all have their upsides and downsides and specific purposes.

Salting is the process or attaching a fixed size of a random set of letters and numbers to the start or end of a password before its hashed to increase the overall security and time to crack. Salting also provides the benefits of every password having a different hashed value as the salts would be different, allowing multiple people to use the same password for a website without having people accessing different accounts that aren't theirs (D Arias, 2018). Salts are usually stored in plaintext along with the hashed password and salt, this allows the salt to be attached to the start or end of the user-entered password and then hashed to compare with the stored value for verification. It's important to produce a new salt for every new password stored due to fact that if the salt is stolen and there is only one salt being used then everyone is at risk compared to the use of multiple where it will reduce the risk and increase the hackers time trying to break several separate passwords (P Menon, 2016).

1.2 Aim

This investigation aims to research and evaluate the hashing algorithms md5, SHA3-256, and bcrypt and determine whether they are still fit for purpose in today's technological world whilst also looking at the salting of a password and how this can be beneficial to the above algorithms. The objectives to achieve this aim would be to test the algorithms using a range of publicly available tools to see if they will/can be broken.

2 PROCEDURE

2.1 OVERVIEW OF PROCEDURE

Within this report, the hashing algorithms MD5, SHA3-256, and Bcrypt are used for testing and comparison. Both the coding languages of Python and PHP were used to produce applications to allow the hashing of all 3 algorithms. For each algorithm both a basic password was used and a more complex one.

In both Languages, each hashing algorithm was used four times to allow for hashing of the basic and complex password. Salts weren't applied to the tested md5 and Sha3-256 due to the software and web-based tools not supporting those hashes being salted. All the results were then tested using hash cracking software where it was applicable.

VS Code (Microsoft, 2020) was used for all the coding and hashing that is present in this investigation. Due to limitations on the software passwords are limited to 6 characters. Both will be tested with 6 characters to provide a fair comparison.

Image 1: Hashsuite interface (Hashsuite, 2020)

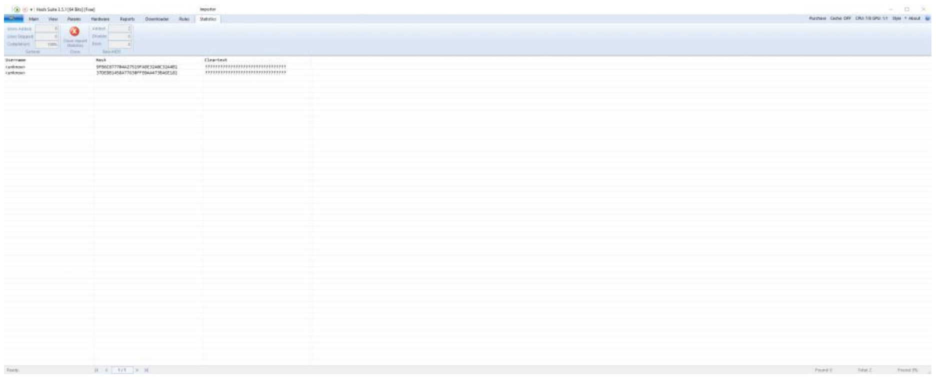
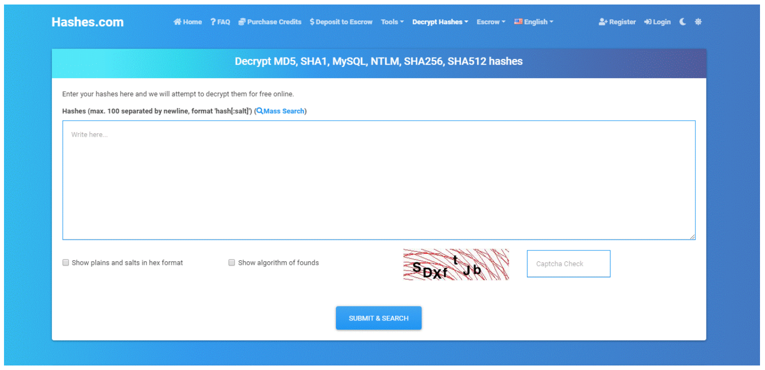


Image 2: Hashes.com Interface



Decrypt MD5, SHA1, MySQL, NTLM, SHA256, SHA512 hashes

Enter your hashes here and we will attempt to decrypt them for free online.

Hashes (max. 100 separated by newline, format 'hash[:salt]') (Mass Search)

Write here...

Show plains and salts in hex format

Show algorithm of founds

Captcha Check

SUBMIT & SEARCH

Image 3: MD5decrypt.net interface

2.2 PROCEDURE PART 1

The basic password in this investigation was defined as “Pass12” and the complex password was defined as “R8azlB”. When using python both MD5 and SHA3-256 were generated using the hashlib library and a random salt for them both was generated using the uuid library. Bcrypt had its library to hash and create salts known as bcrypt.

Due to bcrypt always requiring a salt there is only one version of the hashed bcrypt as its impossible to get an unsalted bcrypt hash. The passwords for MD5 and SHA3-256 were input as strings but due to the way bcrypt works its passwords were input in byte form instead. Each hash was then taken and tested on software known as Hashsuite and a plain brute force attack was used the first time and a dictionary attack was used the second time to test both methods.

Publicly available websites which included hashes.com and MD5decrypt.net were used as a second test to see just how secure the three different algorithms were against accessible software that is available to download or available through a simple website that is open to anyone on the internet.

3 RESULTS

3.1 RESULTS FOR PART 1

After all the testing results were then collated and put into tables for easier reading. Hashed passwords will be identical over both programming languages so Python will be used for results.

See Appendix for hash generation programs.

MD5

Testing the Md5 algorithm was possible both through Hashsuite and through websites. Wordlist used on the application Hashsuite for md5 contained over Ten million different passwords to be tried against the hashed passwords from the investigation.

	Basic	Complex
Hashsuite	Cracked (Brute) Cracked (Dictionary)	Cracked (Brute) Failed (Dictionary)
Web-Based	Cracked	Failed

SHA3-256

Testing of the SHA3-256 algorithm was neither possible on Hashsuite, as that algorithm isn't supported by the application, or on any web-based cracker.

	Basic	Complex
Hashsuite	N/A(Brute) N/A (Dictionary)	N/A(Brute) N/A (Dictionary)
Web-Based	N/A	N/A

Bcrypt

Testing the bcrypt algorithm was not possible on Hashsuite, as the algorithm isn't supported, or on any of the web-based crackers.

	Basic	Complex
Hashsuite	N/A(Brute) N/A (Dictionary)	N/A(Brute) N/A (Dictionary)
Web-Based	N/A	N/A

4 Discussion

4.1 General Discussion

From the results it's clear to see on the first inspection that md5 is incredibly insecure in today's current climate, both SHA and bcrypt held up well in the testing and both remained uncracked at the end of this investigation. The Md5 algorithm was able to be broken in a few seconds on consumer-grade hardware and was relatively simple for the average person to do due to many online guides and tools. Due to the severe insecurity that comes with the algorithm and collision possibility confirmed its clear to see that md5 isn’t fit for purpose in the year 2020. Several other attacks have been done on md5 since collisions were first discovered and have further proven that collisions are apparent in the algorithm (Black, Cochran and Highland, 2006).

SHA3-256 is the most modern algorithm in the investigation and this helped with the inability to crack it as the tools and web-based applications just aren’t available thanks to it being so new. This algorithm has the benefits of being secure and being a fast hashing style of algorithm so it's perfectly suitable to be used in situations where speed is a major factor in the choice of algorithm to use. The migration from md5 to newer algorithms such as the SHA3 family is largely put off due to the sheer amount of systems that would have to be changed or even rewritten and the massive amount of work that would be required to swap everything just isn’t worth the trouble in some companies minds. Bcrypt is the odd one out given that it’s a slow hashing style and so it’s safer from the get-go.

Just like SHA3-256, bcrypt was unable to be broken by the software or web-based systems but unlike SHA3 the bcrypt algorithm isn’t incredibly new. Due to the mixture of slow hashing style and the required inclusion of a salt with every hash request, it makes it incredibly difficult to calculate bcrypt hashes and so it’s difficult for it to be cracked. The distribution of wordlists and rainbow tables which contain huge sets of pre-calculated hashes can be a quick way for cracking of hashes to be done as the hard calculations are already done and the software just has to check to see if any hashes match the pre-calculated ones.

The investigation aimed to evaluate the effectiveness of the 3 hashing algorithms. Each algorithm has been investigated and tested using easily available technology to see if they are breakable and as we see from our results, one of them was breakable using the technology at hand with both an insecure password and a much more complex password.

4.2 Countermeasures

Ways in which the problems raised could be avoided or their impact decreased would primarily relate to salting the MD5 and SHA3 hashes before hashing to provide them with a bit more security, this primarily relates to MD5 considering from testing we have seen how unsafe it can be. Another countermeasure would be to implement newer/more sophisticated algorithms into new systems and applications instead of implementing more unreliable algorithms that will cause trouble in the long run and risk security breaches. In the investigation using web-based tools and it became apparent just how abundant these tools can be, with most algorithms having an online hasher, some an online hash checker, and a few having an actual online hash cracker. To counter these tools using strong passwords made up of random letters, numbers, and symbols seemed to disrupt the tools that were used in this report. These tools will run off pre-determined word tables and so avoiding dictionary words and other commonly used words can help reduce the chances of your password being cracked online. Once again adding a salt to the password before hashing it can throw off most of the basic online tools out there and would more than likely put off the average person trying to crack your hash.

4.3 Conclusions

In conclusion from this investigation, we can see that MD5 should not be used in any new systems where it isn't already embedded or implemented due to the severe exploitability of it in today's environment. SHA3 and bcrypt are far safer and would take a substantial amount of processing power and a modern program to crack even without the implementation of a salt with the hashed password.

4.4 Future Work

With more time and resources more algorithms could have been tested and more substantial testing methodology using better software to allow for much longer passwords and rainbow tables could have provided evidence of other algorithms being able to be cracked. With rainbow tables and more substantial wordlists could have been used to crack the hashes faster and a larger set of passwords could have been used to provide a larger sample size.

References

Why md5 is not safe (No date) [Blog]. Available from: https://www.md5online.org/blog/why-md5-is-not-safe/ [Accessed 1 May 2020].

Hawkes, P., Paddon, M. and Rose, G. 2004. Musings on the Wang et al. MD5 Collision. [Paper]. Available from: https://eprint.iacr.org/2004/264.pdf [Accessed 1 May 2020].

Keccak Team. (No date) *Strengths of Keccak* [Online]. Available from: https://keccak.team/keccak_strengths.html [Accessed 1 May 2020].

Arias, D. 2018. Hashing in Action: Understanding bcrypt [Blog]. Available from: https://auth0.com/blog/hashing-in-action-understanding-bcrypt/ [Accessed 1 May 2020].

Arias, D. 2018. *Adding Salt to Hashing: A better way to store passwords*. [Blog]. Available from: https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/ [Accessed 1 May 2020].

Menon, P. 2016. What is a Salty and how does it make password hashing more secure?. [Blog]. Available from: https://www.skyhighnetworks.com/cloud-security-blog/author/prasida-menon/ [Accessed 1 May 2020]

Microsoft. 2020. Visual Studio Code. [Computer Program]. Available from https://code.visualstudio.com/#alt-downloads [Accessed 1 May 2020]

Hashsuite. 2020. Hashsuite Free. [Computer Program]. Available from https://hashsuite.openwall.net/ [Accessed 1 May 2020]

Black, J., Cochran, M. and Highland, T. 2006. A study of MD5 Attacks: Insights and improvements [Paper]. Available from: https://link.springer.com/content/pdf/10.1007%2F11799313_17.pdf [Accessed 1 May 2020]

APPENDICES

APPENDIX A

Python code:

```python
import hashlib
import uuid
import bcrypt
text= ('Pass12')
textcompllex=('R8azlB')
textb=(b'Pass12')
textbcomplex=(b'R8azlB')
salt = uuid.uuid4().hex
saltb= bcrypt.gensalt()

#MD5 Hashing
hex_dig=hashlib.md5(salt.encode() + text.encode()).hexdigest() + ':' + salt
hex_digcomplex=hashlib.md5(salt.encode() + textcompllex.encode()).hexdigest() + ':' + salt
print(hex_dig) # outputs salted basic plus salt at the end
print(hex_digcomplex) # outputs salted complex plus salt at the end
print (text)
print(hashlib.md5(text.encode()).hexdigest()) # Outputs the hash of basic
print(textcompllex)
print(hashlib.md5(textcompllex.encode()).hexdigest()) # outputs hash of complex
print("----------------")

# SHA3_256
hex_dig1=hashlib.sha3_256(salt.encode() + text.encode()).hexdigest() + ':' + salt
hex_digcomlex1=hashlib.sha3_256(salt.encode() + textcompllex.encode()).hexdigest() + ':' + salt
print(hex_dig1) # outputs hash plus salt hashed
print(hex_digcomlex1)
print (text)
print(hashlib.sha3_256(text.encode()).hexdigest()) # Outputs the hash of password
print(textcompllex)
print(hashlib.sha3_256(textcompllex.encode()).hexdigest())
print("----------------")

mypassword = bcrypt.hashpw(textb,saltb)
print(mypassword)
print (saltb)
# if bcrypt.checkpw(textb,mypassword):
#     print("True")
# else: print ("False")
```

APPENDIX B

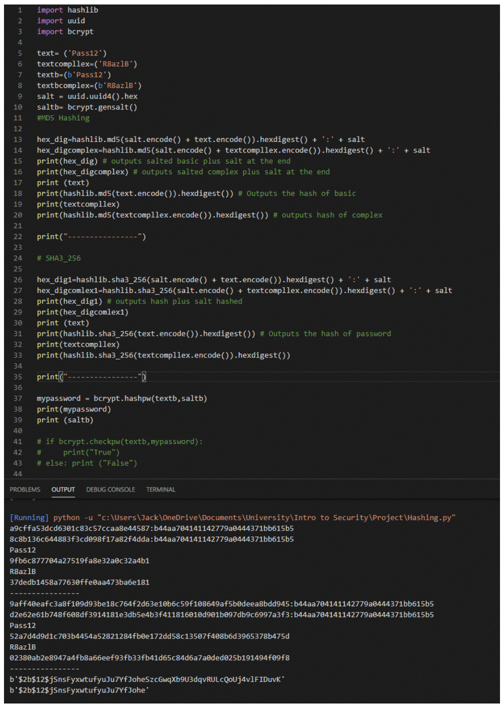
PHP code:

```
<?php
// PHP code to illustrate the working
// of md5(), sha3() and bcrypt()
$str = 'Pass12';
$str1 = 'R8azlB';
$salt = 'b44aa704141142779a0444371bb615b5';
echo sprintf("The md5 hashed password of %s is: %s\n",
                                $str, md5($str.$salt));
echo("-----------------------\n");
echo("md5 hashed password of R8azlB: ");
echo hash('md5',$str1);
echo("\n");
echo("-----------------------\n");
echo("Sha3 hashed password of Pass12: ");
echo hash('sha3-256' , $str);
echo("\n");
echo("-----------------------\n");

echo("Sha3 hashed password of R8azlB: ");
echo hash('sha3-256' , $str1);
echo("\n");
echo("-----------------------\n");

echo("bcrypt hashed password of Pass12: ");
$options = [
    'cost' =>12,
];
echo password_hash($str, PASSWORD_BCRYPT);
echo("\n");

echo("bcrypt hashed password of R8azlB: ");
$options = [
    'cost' =>12,
];
echo password_hash($str1, PASSWORD_BCRYPT);
?>
```

APPENDIX C

Screenshots of both Python and PHP code for recreation.

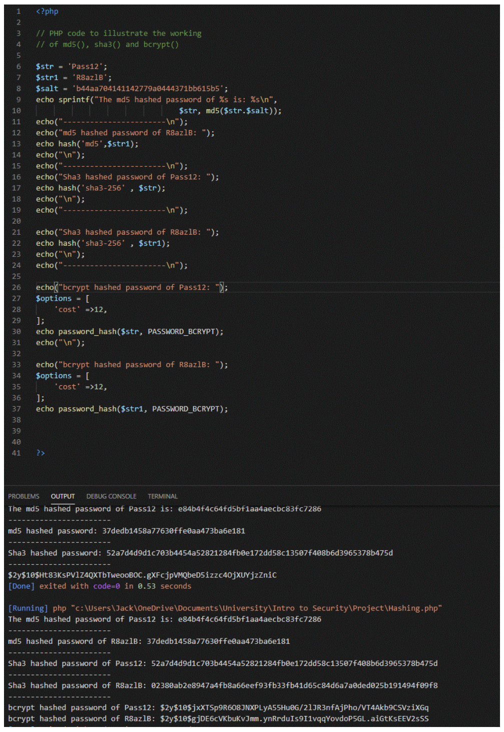
Python:



```python
import hashlib
import uuid
import bcrypt

text= ('Pass12')
textcompllex=('R8azlB')
textb=(b'Pass12')
textbcomplex=(b'R8azlB')
salt = uuid.uuid4().hex
saltb= bcrypt.gensalt()
#MD5 Hashing

hex_dig=hashlib.md5(salt.encode() + text.encode()).hexdigest() + ':' + salt
hex_digcomplex=hashlib.md5(salt.encode() + textcompllex.encode()).hexdigest() + ':' + salt
print(hex_dig) # outputs salted basic plus salt at the end
print(hex_digcomplex) # outputs salted complex plus salt at the end
print (text)
print(hashlib.md5(text.encode()).hexdigest()) # Outputs the hash of basic
print(textcompllex)
print(hashlib.md5(textcompllex.encode()).hexdigest()) # outputs hash of complex

print("---------------")

# SHA3_256

hex_dig1=hashlib.sha3_256(salt.encode() + text.encode()).hexdigest() + ':' + salt
hex_digcomlex1=hashlib.sha3_256(salt.encode() + textcompllex.encode()).hexdigest() + ':' + salt
print(hex_dig1) # outputs hash plus salt hashed
print(hex_digcomlex1)
print (text)
print(hashlib.sha3_256(text.encode()).hexdigest()) # Outputs the hash of password
print(textcompllex)
print(hashlib.sha3_256(textcompllex.encode()).hexdigest())

print("---------------")

mypassword = bcrypt.hashpw(textb,saltb)
print(mypassword)
print (saltb)

# if bcrypt.checkpw(textb,mypassword):
#     print("True")
# else: print ("False")

```

```
[Running] python -u "c:\Users\Jack\OneDrive\Documents\University\Intro to Security\Project\Hashing.py"
a9cffa53dcd6301c83c57ccaa8e44587:b44aa704141142779a0444371bb615b5
8c8b136c644883f3cd098f17a82f4dda:b44aa704141142779a0444371bb615b5
Pass12
9fb6c877704a27519fa8e32a0c32a4b1
R8azlB
37dedb1458a77630ffe0aa473ba6e181
---------------
9aff40eafc3a8f109d93be18c764f2d63e10b6c59f108649af5b0deea8bdd945:b44aa704141142779a0444371bb615b5
d2e62e61b748f608df3914181e3db5e4b3f411816010d901b097db9c6997a3f3:b44aa704141142779a0444371bb615b5
Pass12
52a7d4d9d1c703b4454a52821284fb0e172dd58c13507f408b6d3965378b475d
R8azlB
02380ab2e8947a4fb8a66eef93fb33fb41d65c84d6a7a0ded025b191494f09f8
---------------
b'$2b$12$jSnsFyxwtufyuJu7YfJoheSzcGwqXb9U3dqvRULcQoUj4vlFIDuvK'
b'$2b$12$jSnsFyxwtufyuJu7YfJohe'
```

PHP:



```php
<?php

// PHP code to illustrate the working
// of md5(), sha3() and bcrypt()

$str = 'Pass12';
$str1 = 'R8azlB';
$salt = 'b44aa704141142779a0444371bb615b5';
echo sprintf("The md5 hashed password of %s is: %s\n",
                                $str, md5($str.$salt));
echo("----------------------\n");
echo("md5 hashed password of R8azlB: ");
echo hash('md5',$str1);
echo("\n");
echo("----------------------\n");
echo("Sha3 hashed password of Pass12: ");
echo hash('sha3-256' , $str);
echo("\n");
echo("----------------------\n");

echo("Sha3 hashed password of R8azlB: ");
echo hash('sha3-256' , $str1);
echo("\n");
echo("----------------------\n");

echo("bcrypt hashed password of Pass12: ");
$options = [
    'cost' =>12,
];
echo password_hash($str, PASSWORD_BCRYPT);
echo("\n");

echo("bcrypt hashed password of R8azlB: ");
$options = [
    'cost' =>12,
];
echo password_hash($str1, PASSWORD_BCRYPT);



?>
```

```
The md5 hashed password of Pass12 is: e84b4f4c64fd5bf1aa4aecbc83fc7286
----------------------
md5 hashed password: 37dedb1458a77630ffe0aa473ba6e181
----------------------
Sha3 hashed password: 52a7d4d9d1c703b4454a52821284fb0e172dd58c13507f408b6d3965378b475d
----------------------
$2y$10$Ht83KsPVlZ4QXTbTweooBOC.gXFcjpVMQbeD5izzc4OjXUYjzZniC
[Done] exited with code=0 in 0.53 seconds

[Running] php "c:\Users\Jack\OneDrive\Documents\University\Intro to Security\Project\Hashing.php"
The md5 hashed password of Pass12 is: e84b4f4c64fd5bf1aa4aecbc83fc7286
----------------------
md5 hashed password of R8azlB: 37dedb1458a77630ffe0aa473ba6e181
----------------------
Sha3 hashed password of Pass12: 52a7d4d9d1c703b4454a52821284fb0e172dd58c13507f408b6d3965378b475d
----------------------
Sha3 hashed password of R8azlB: 02380ab2e8947a4fb8a66eef93fb33fb41d65c84d6a7a0ded025b191494f09f8
----------------------
bcrypt hashed password of Pass12: $2y$10$jxXTSp9R6O8JNXPLyA55Hu0G/2lJR3nfAjPho/VT4Akb9CSVziXGq
bcrypt hashed password of R8azlB: $2y$10$gjDE6cVKbuKvJmm.ynRrduIs9I1vqqYovdoP5GL.aiGtKsEEV2sSS
```