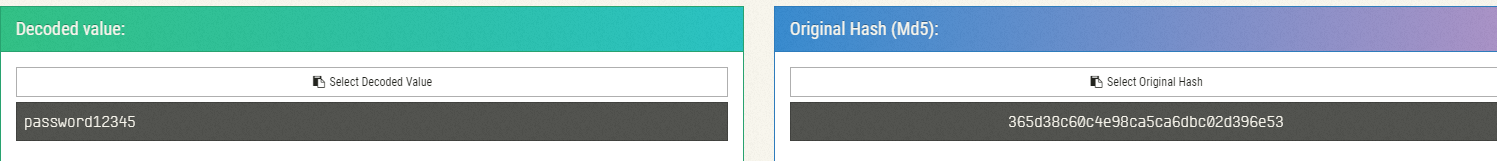
**Opdracht 5: Ensuring Integrity**

**5.1: Terminology En Vakjargon**

|  |  |  |
| --- | --- | --- |
| Term | Explanation | Translation (NL) |
| SPoF | Part of a system that, if it fails, stops the entire system from working. (Unwanted in a system with a high availability/reliability) | Enigste faalpunt |
| Resilience | The ability to protect data and apps from just about any type of issue | veerkracht |
| Redundancy | A system design in which a component is duplicated so if it fails there will be a backup | overtolligheid |
| CVE | A list of Common Vulnerabilities and Exposures entries, each containing a number, a description and at least one public reference | Vaak voorkomende kwetsbaarheden en blootstellingen |
| CMDB | A database that contains all relevant information about the hardware and software components used in an organization’s IT services (and related services) | Configuratie management database |
| Perimeter | The boundary between private and locally managed-and-owned side of a network, the public and usually provider-managed side of a network | Omtrek/grens |
| Fail-over | A procedure by which a system automatically transfers control to a duplicate system when it detects a fault or failure. | Faaltransfer |
| ZFS | A combined file system and logical volume manager. (designed by Sun Microsystems) | Een gecombineerd bestandssysteem & logisch volume beheerder |
| 4eyes principle | A principle that means that a certain activity, (decision, transaction, …), must be approved by at least 2 people | 4-ogen principe |
| NAC | An approach to computer security that attempts to unify endpoint security technology, user or system authentication and network security enforcement. | Netwerk toegangscontrole |
| Snort | An open-source, free and lightweight network intrusion detection system (NIDS) software for Linux and Windows to detect emerging threats | Snort |

**5.2: Crack The Hash**

MD5Hashing.net  
password12345 - PXL{Cr4ck\_Th3\_P4ssw0rd}



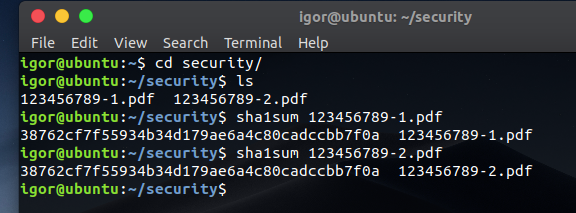
**5.3: SHA1 Collisions**

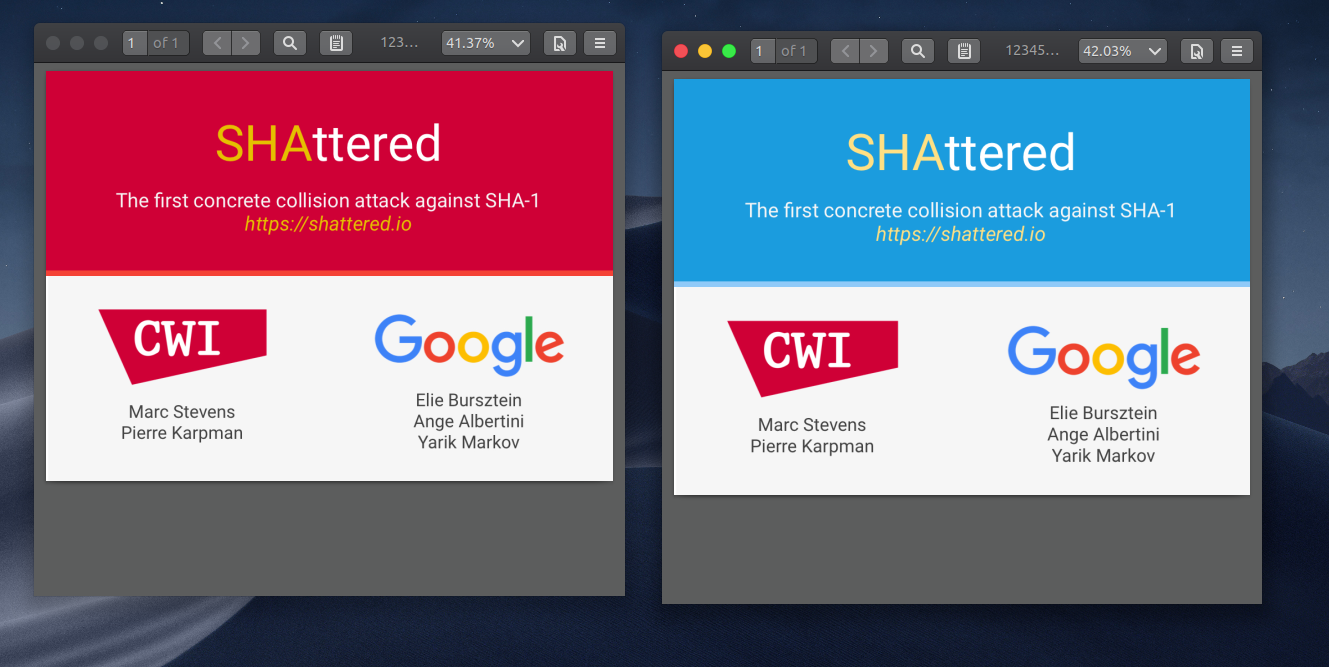
**Introduction**

Werd eerst ontdekt op:

<https://shattered.io/>

Niemand had het gedacht maar de twee hashes zijn hetzelfde terwijl het wel om twee verschillende bestanden gaat.





**Big deal, so what?**

Stel je voor dat je een file zou moeten ondertekenen waarbij je huur staat op $500 per maand. Opeens krijg je het bestand, je checkt de SHA-1 sum na, deze komt overeen, en van wetend onderteken je het bestand. Op deze staat de huur wel $3000 per maand. Zie je het probleem?

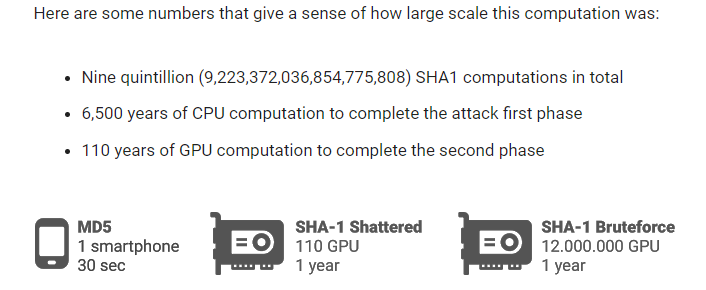
Dit zou dus rampzalige gevolgen kunnen hebben.

Hoe kon dit toch gebeuren dan? Is het toeval? Of is dit kwaadaardig opzet?

Wel, eigenlijk heeft het security team van Google dit bewust gedaan.

**Creating SHA-1 Collisions**

ier zijn wat cijfers wat hoe moeilijk het eigenlijk was:



Hoe dat ze het precies gedaan hebben is hier te vinden:

<https://marc-stevens.nl/research/papers/SBKAM17-SHAttered.pdf>

Omdat het vrij geadvanceerd is en een team van 5 security experts kostte gaan wij er niet te fel op in gaan.

In tussentijd is er ook al een tool om zelf SHA1 colliding pdf’s te maken.

<https://alf.nu/SHA1>

**Conclusion**

Technologie blijft evolueren en SHA-1 had zeker zijn flaws. Nu naar de toekomst toe beginnen we stilaan alles SHA-1 vrij te maken en steeds meer en meer te gaan naar SHA-256 en SHA-512.

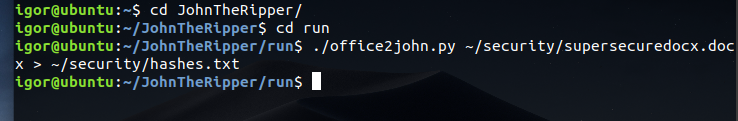
Dat betekent niet dat SHA-1 zwak is. Een bedrijf dat een SHA-1 collide zou kunnen maken is nog steeds zeer onrealistisch. De collide van Google kostte hen alvast $130 000 aan cloud computing power.

Terwijl SHA-1 niet het meest secure is blijft het zeker en vast betrouwbaar.

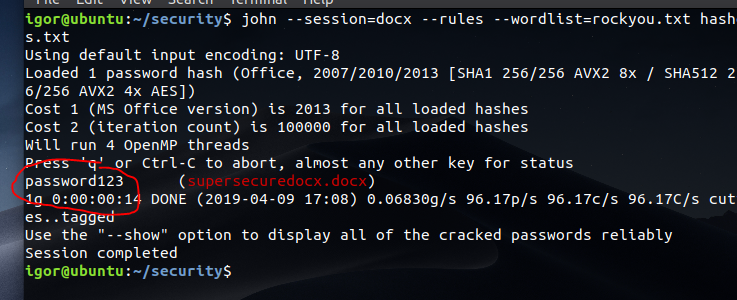
**5.4: Brute Force**

We hebben John The Ripper en een basic wachtwoorden bestand (Rockyou.txt) gedownload om een bruteforce te doen op de versleuteling van het .docx document. Hieronder de werkwijze:

Eerst moesten we de hash krijgen van het bestand. Dit kon makkelijk gedaan worden met office2john

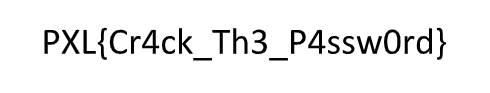


Dan konden we bruteforcen met onze wordlist rockyou.txt en hash



Uiteindelijk hebben we dan ***password123*** gevonden.

Door het in te geven konden we dan de flag vinden:



**5.5: SSL Certificates**

**UHasselt.be**

www.UHasselt.be heeft een Overall Rating A, dit is het beste wat kan behaald worden. Als er iets is waar nog verbetering mogelijk in is, dan is het Protocol Suppoert en Cipher Strength. Certificate en Key Exchange halen hier een perfecte score.

**pxl.be**

www.pxl.beheeft een Overall Rating B, normaal gezien had SSL Labs het een A gegeven, maar het wordt gebottleneckt door de Key Exchange, die maar matig scoort in de ratings. www.pxl.be haalt hier wel een perfecte score op certificaat en zeer goede scores op Protocol Support en Cipher Strength.

**howest.be**

www.howest.behaalt een Overall Rating A, Alle onderdelen halen ongeveer hetzelfde, buiten dat Certificate een perfecte score haalt.

**vub.ac.be**

www.vub.ac.be haalt een Overall Rating B, ook deze website haalt een perfecte score op Certificate, gevolgd door uitstekende scores op Protocol Support en Key Exchange. Als het niet voor Cipher Strength was geweest, waarop de site een matige score kreeg, zou het een Overall Rating A gekregen hebben.

**xjufe.edu.cn**

**www.xjufe.edu.cn** Heeft een Overall Rating T, dit wilt zeggen dat de server zijn SSL Certificaat niet trusted is door SSL Labs. Hierdoor heeft het een minimumscore voor Certificate. Daarbuiten haalt de site een uitstekende score op Protocol Support, Key Exchange en Cipher Strength

**Conclusie:** We kunnen afleiden dat, indien de server zijn certificaat erkent is door SSL Labs, je automatisch een maximum-score haalt op Certificate. Statistisch gezien zijn Cipher Strength en Key Exchange de moeilijkste om een "A" score op te behalen

**5.6: Eigenschappen Van Een Hash**  
  
A hash is comparable to a barcode. It uses a series of numbers that sums up everything that is in that file. A hash is a crushed down file, so it contains everything in that file. You cannot use this process backwards, so you can’t use the hash to recreate the file. Kind of like a signature, it’s unique per file.

Hash algorithm requirements:

-It has to be fast, being able to make itself for a large file in a second or two at most. It can’t be to quick however, as this will make it easy to break. (further explanation below)

-If one byte/bit has been changed in the file, the whole hash should be different.

-Being able to avoid hash collisions (two files that have the same hash)

Pigeonhole principle: There are always going to be multiple files with the same hash. This is naturally unavoidable, however the chances of this happening are incredibly low. If, however, you are able to create a hash collision artificially, like by for instance changing the name, it comes to have very low security. (you could intercept an important document, change it and it would still have the same hash. This could be used with malicious intent)

If the hash process is too fast, it is too easy to replicate hashes. They could just use code to change a bit/byte and eventually have the same hash.

MD5 hash, used to be the most used hash, however now it has become easy to crack. For example: By simply typing in the hash on google, you could find the English word it was supposed to protect.

This could have been prevented by not using the MD5 hash for everything, as it was for several years. Multiple MD5 Hash crackers can be found by simply searching on google.

Currently, people are still using SSH1, however there are rumors that this one will break soon, if it has not already. So people will move on to using SSH2 and government agencies are already using SSH3. SSH3 will, in a few years, become the new standard. (Should only be used for files, not password)

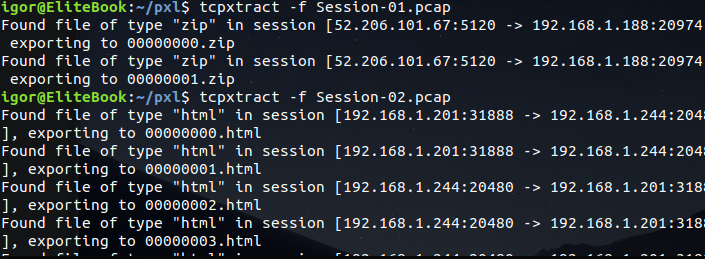
Some sites will show the hash on download, but this is a bad idea, if a person with malicious intent went onto their site, changed the file and cracked the hash (or changed it with access to the site) it is fairly easy to spread malware or other malicious programs.

So essentially, a hash takes a big chunk of data and turns it into a small amount for verification purposes.

**5.7: Using Digital Signatures**

Na een uur te zoeken op WireShark was het tijd voor andere tools. Tcpxtract leek wel alsof het hulp kon bieden. Het maakt gebruik/misbruik van de TCP en hier worden files uit geextractet.

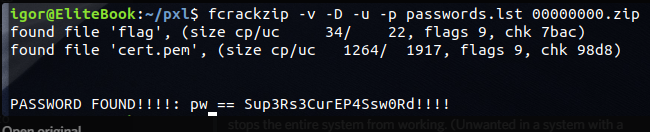
Zoals onder op de screenshot te zien had het in Session-01.pcap twee zip files gevonden.



Toen we de zip wouden extracten vroeg het helaas om een password.

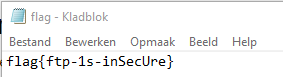
Gelukkig hadden we een paar passwords om uit te kiezen door het meegegeven passwords.lsd

Door een bruteforce te doen met passwords.lsd als passwordbestand kwamen we uit op Sup3Rs3CurEP4Ssw0Rd!!!! als het juiste paswoord, wat erg voor de hand lag.



Zo konden we dus de file upzippen en de flag bekijken.

Flag:



Tijdsraming

|  |  |  |  |
| --- | --- | --- | --- |
| Naam | Opdracht | Tijd | Totaal |
| Vicky | 5.2, 5.4, 5.7, Opmaak | 1 uur, 1.30 uur, 1 uur 30 min, 30 min | 4.30 uur |
| Bram | 5.1, 5.6 | 45 min, 1 uur | 1.45 uur |
| Igor | 5.3, 5.4, 5.7 Nakijken + Aanpassingen | 1 uur, 1 uur, 1 uur 30 min, 30 min | 4 uur |
| Vincent | 5.5 |  | 45 min |

Doorsturen document: Igor

Taakverdeling: Vicky