1.

- xpath przyda sie do wyszukiwania elementow na stronie

- fajnie to zrobić jak studenci, czyli skrypt który podswietla elementy możliwe, do zaznacznia

2. web scraping ebook

- wget, curl nie wystarcza, bo obecnie strony czerpią dane z roznych serwerow, a wiec najpierw trzeba poprać stronke i poczekać o skrypte w niej pobiara dane

3.

<https://fsgeek.pl/post/sop-cors/>

tutaj na koncu mowia jak poradzić sobie z sop, wlaczajac odpowiednia opcje w przegladarce lub wtyczke

4. filmik o scrapowaniu tekstuz internetu i wysylania maili. Co prawda ja chce scrapowac cala strone nie tylko text, ale wysylac maile tez chce. 😉

<https://www.youtube.com/watch?v=1UMHhJEaVTQ&list=WL&index=4&t=228s>

5. hashowanie haseł u mnie

To paraphrase [my answer to an earlier question](https://crypto.stackexchange.com/questions/8159/what-is-the-difference-between-scrypt-and-pbkdf2/8170#8170), [PBKDF2](https://en.wikipedia.org/wiki/PBKDF2) is a generic high-level algorithm that internally calls a [pseudorandom function](http://en.wikipedia.org/wiki/Pseudorandom_function) (PRF) to process its input. The PBKDF2 spec does not mandate any particular PRF, so implementors are free to choose any PRF they want (as long as it meets the definition of a secure PRF, and can accept the input PBKDF2 gives it).

As it happens, by far the most common choice of PRF for PBKDF2 is [HMAC](http://en.wikipedia.org/wiki/HMAC), which is another high-level construction that internally uses a [cryptographic hash function](http://en.wikipedia.org/wiki/Cryptographic_hash_function). Again, the HMAC spec does not mandate any particular hash function,\* so implementors are free to choose any hash they want. Probably the most common choice today is one of the [SHA-2](http://en.wikipedia.org/wiki/SHA-2) family of hashes, which include SHA-256 and SHA-512.

So "SHA512 within PBKDF2" almost certainly means that they're using PBKDF2 with HMAC as the PRF, and with SHA-512 as the hash inside HMAC.\*\*

What may be confusing is that, at a glance, this PBKDF2-with-HMAC-with-SHA512 may look like it's doing something very similar to just plain SHA-512: both take an arbitrary password as input and turn it into a pseudorandom bit string from which the original password cannot be easily reconstructed. However, there are actually several differences, the most important ones being that:

* SHA-512 is fast. Very fast. PBKDF2 is deliberately slow to compute, and its slowness can be controlled by adjusting the iteration count parameter.
* As a direct consequence of its speed, SHA-512 alone is vulnerable to brute force password guessing attacks using software like [hashcat](https://en.wikipedia.org/wiki/Hashcat), which simply generate lots of passwords and hash them until they find one that produces a matching hash. A single modern CPU can easily hash millions of passwords per second, and GPUs are even faster.

In addition, there are a few other minor differences with noting:

* PBKDF2 takes a "salt" parameter, which can be used to [ensure that the outputs are unique even if the input passwords are not](https://crypto.stackexchange.com/questions/1776/can-you-help-me-understand-what-a-cryptographic-salt-is). (Using a salt also thwarts precomputation-based attacks, since the attacker cannot usefully begin hashing guessed password before they know the salt.) SHA-512 by itself does not have the concept of a salt, although an equivalent effect can be achieved e.g. by prepending the salt to the password.
* SHA-512 always produces a 512-bit output (hence its name), although you can always truncate it to a shorter length if you don't need the full 512 bits. PBKDF2 can, in principle, produce output of any length, although in practice [using it to generate more than one internal hash output's worth of bits is not recommended](https://crypto.stackexchange.com/questions/54922/pbkdf2-with-longer-than-native-output-length-will-hashing-the-result-help).

\*) The original HMAC definition and security proofs effectively assume that the hash function used is of a particular type known as a[Merkle–Damgård hash function](https://en.wikipedia.org/wiki/Merkle%E2%80%93Damg%C3%A5rd_construction). As it happens, all the most popular cryptographic hash functions for the past several decades, including the SHA-2 family, have been of this type, so this limitation has not been much of an issue in practice. This may be gradually changing with the standardization of[SHA-3](https://en.wikipedia.org/wiki/SHA-3)(a.k.a. Keccak), which isnota Merkle–Damgård hash, but conveniently, comes with its own[security claim for HMAC-SHA3](https://crypto.stackexchange.com/questions/15782/how-secure-would-hmac-sha3-be/15825#15825).

\*\*) This is a fine and traditional choice, as far as it goes. It's not as resistant to GPU-based and other parallel attacks as more modern KDFs like[scrypt](https://en.wikipedia.org/wiki/Scrypt)or[Argon2](https://en.wikipedia.org/wiki/Argon2)would be, but it's still a lot better than plain old un-iterated hashing. That said, to properly evaluate its security, we would also need to know the iteration count used for PBKDF2. Unfortunately, many PBKDF2 implementations tend to use the old "recommended minimum" of 1000 iterations, which is little more than a speed bump nowadays. Personally, on a modern CPU,[I'd prefer something closer to 1,000,000 or 1,000,000,000 iterations](https://crypto.stackexchange.com/questions/46955/what-is-the-correct-way-to-implement-pbkdf2-aes-cbc-hmac/46978#46978).

zrodlo

https://crypto.stackexchange.com/questions/35275/whats-the-difference-between-pbkdf-and-sha-and-why-use-them-together