Bezpieczeństwo systemów komputerowych

Zbiór zadań, część pierwsza

Katarzyna Mazur

18 września 2024

Art. 267 KK

§ *Kto bez uprawnienia uzyskuje dostęp do całości lub części systemu informatycznego podlega karze pozbawienia wolności do lat 2.*

Art. 269a KK

§ *Kto, nie będąc do tego uprawnionym w istotnym stopniu zakłóca pracę systemu systemu teleinformatycznego lub sieci teleinformatycznej, podlega karze pozbawienia wolności od 3 miesięcy do lat 5.*

Art.269c KK

§ *Nie podlega karze za przestępstwo określone w art. 267 § 2 lub art. 269a, kto działa wyłącznie w celu zabezpieczenia systemu teleinformatycznego albo opracowania metody takiego zabezpieczenia i niezwłocznie powiadomił dysponenta tego systemu lub sieci o ujawnionych zagrożeniach, a jego działanie nie naruszyło interesu publicznego lub prywatnego i nie wyrządziło szkody.*

1 Informacje

GitHub

Na potrzeby zajęć swoje własne repozytorium na Githubie. Rezpozytorium będziesz wykorzystywać, aby wrzucać do niego rozwiązania zadań z przedmiotu. Możesz nazwać repozytorium dowolnie, ale najlepiej wykorzystaj nazwę: `imie-nazwisko-bsk-umcs`.

Wirtualne środowisko pracy

Wirtualne środowisko pracy w Python, znane również jako `virtualenv` lub `venv`, jest sposobem za zapewnienie działania naszego programu niezależnie od maszyny na której jest uruchamiany ORAZ sposobem na uruchomienie innych programów na NASZEJ maszynie, tak aby instalowane z nią zależności nie zakłóciły pracy innych programów. Wirtualne środowisko pracy, jest swojego rodzaju odizolowanym katalogiem, zawierającym instalację języka programowania Python oraz zainstalowane biblioteki na potrzeby programu, który będziemy w nim uruchamiać.

Instalacja

```
sudo apt update
sudo apt install python3-pip
python3 -m pip install --upgrade pip

sudo apt install python3-venv
python3 -m pip install --user virtualenv
```

Tworzenie środowiska

```
python3 -m venv venv
source ./venv/bin/activate
```

Instalowanie pakietów

```
pip list
pip install black
black test.py
```

Dezaktywacja (koniec pracy) wirtualnego środowiska

```
deactivate
```

Dobre praktyki

W zadaniach najczęściej będziemy wykorzystywać język programowania Python. Aby zachować dobre praktyki programistyczne, będziemy używać black'a oraz pylamy. Jeśli to możliwe, możesz wykorzystać inny język programowania (jest to zależne od zadania). Pamiętaj jednak również o zachowaniu dobrych praktyk.

Materiały

- `https://securecodingdojo.owasp.org/public/index.html`
- `https://overthewire.org/wargames/`
- `https://pentesterlab.com/exercises`

2 Funkcje haszujące

2.1 Teoria

Funkcje haszujące są kluczowym elementem w informatyce, wykorzystywanym do przekształcania danych o różnych rozmiarach i typach w znormalizowane wartości o stałej długości. Poniżej najważniejsze cechy funkcji haszujących:

- **Jednostronność**: Funkcje haszujące są jednostronne, co oznacza, że trudno jest odwrócić proces haszowania i odzyskać oryginalną wartość z jej skrótu.
- **Stała długość skrótu**: Bez względu na rozmiar wejściowego danych, funkcje haszujące generują skróty o stałej długości, co zapewnia spójność w przechowywaniu danych.
- **Unikalność**: Idealnie funkcja haszująca powinna generować unikalne skróty dla różnych wejść, chociaż w praktyce mogą występować kolizje.
- **Deterministyczność**: Funkcje haszujące są deterministyczne, co oznacza, że dla tej samej wartości wejściowej zawsze generują ten sam skrót.
- **Szybkość**: Efektywne funkcje haszujące muszą być szybkie, aby umożliwić przetwarzanie dużych ilości danych w krótkim czasie.
- **Odporność na kolizje**: Odporność na kolizje jest kluczowa dla zapewnienia bezpieczeństwa danych, choć niemożliwe jest całkowite uniknięcie ich wystąpienia.

Funkcje haszujące mają szerokie zastosowanie w różnych obszarach informatyki i technologii. Oto kilka przykładów ich zastosowań:

- **Zabezpieczanie haseł użytkowników**: W systemach uwierzytelniania hasła użytkowników są zazwyczaj zapisywane jako skróty za pomocą funkcji haszujących. Dzięki temu, nawet w przypadku wycieku danych, oryginalne hasła są trudne do odzyskania.
- **Kontrola integralności danych**: Funkcje haszujące są wykorzystywane do generowania sum kontrolnych plików. Porównuje się te sumy kontrolne, aby sprawdzić, czy plik nie został uszkodzony lub zmieniony.
- **Autoryzacja użytkowników**: W systemach autoryzacji, funkcje haszujące są stosowane do porównywania haszy haseł wprowadzonych przez użytkowników z haszami przechowywanymi w bazie danych.
- **Kryptografia**: W kryptografii funkcje haszujące są wykorzystywane do wielu celów, w tym do generowania kluczy kryptograficznych, podpisów cyfrowych, czy też w protokołach bezpiecznej komunikacji.
- **Eksport kontrolowanych danych**: W niektórych przypadkach funkcje haszujące są używane do eksportu kontrolowanego danych, gdzie oryginalne dane są zastępowane ich skrótami w celu ochrony prywatności.

Te przykłady pokazują wszechstronność funkcji haszujących i ich kluczową rolę w zapewnianiu integralności danych oraz bezpieczeństwa w różnych dziedzinach informatyki i technologii.

2.1.1 Standardowe funkcje haszujące

W kryptografii i informatyce istnieje wiele standardowych funkcji haszujących, z których najbardziej powszechne to:

- **MD5 (Message Digest Algorithm 5)**:
  - Generuje 128-bitowy (16-bajtowy) skrót.
  - Został stworzony przez Ronalda Rivesta w 1991 roku.
  - Uważany jest za obecnie złamaną i nie nadaje się do zastosowań, w których wymagane jest wysokie bezpieczeństwo.
- **SHA-1 (Secure Hash Algorithm 1)**:
  - Generuje 160-bitowy (20-bajtowy) skrót.
  - Jest często stosowany, ale również uważany za obecnie złamaną funkcję haszującą.
- **SHA-256 (Secure Hash Algorithm 256-bit)**:
  - Generuje 256-bitowy (32-bajtowy) skrót.
  - Jest obecnie uważany za bezpieczny i jest szeroko stosowany w różnych zastosowaniach, takich jak generowanie skrótów haseł czy podpisywanie cyfrowe.

2.1.2 Nowoczesne funkcje haszujące

Argon2 to funkcja haszująca i haszowanie haseł, która jest zaprojektowana w celu zapewnienia wysokiego poziomu bezpieczeństwa i odporności na różne ataki kryptograficzne, takie jak ataki brute force czy ataki z użyciem tęczowych tablic. Algorytm Argon2 został wybrany jako zwycięzca konkursu Password Hashing Competition (PHC) w 2015 roku, gdzie oceniano różne algorytmy haszujące pod kątem ich bezpieczeństwa i wydajności w kontekście haszowania haseł. Oto główne parametry algorytmu Argon2:

- **Pamięć** (`Memory`): Określa ilość pamięci w kibibajtach, która jest używana podczas obliczeń. Większa ilość pamięci zwykle oznacza większe bezpieczeństwo, ale może prowadzić do większego obciążenia systemu. Parametr ten oznaczony jest jako `m` i jest wyrażany w kibibajtach.
- **Czas** (`Time`): Określa liczbę iteracji, które algorytm wykonuje podczas obliczeń. Większa liczba iteracji oznacza większe bezpieczeństwo, ale również dłuższy czas obliczeń. Parametr ten oznaczony jest jako `t` i wyrażany jest w liczbach całkowitych.
- **Wątki** (`Threads`): Określa równoczesną liczbę wątków użytych do obliczeń. Większa liczba wątków może przyspieszyć obliczenia na wielordzeniowych procesorach, ale może również zwiększyć obciążenie systemu. Parametr ten oznaczony jest jako `p` i jest liczbą całkowitą.
- **Sól** (`Salt`): Jest to losowo wygenerowany ciąg bitów używany do uniknięcia tęczowych tablic i innych ataków kryptograficznych. Każde hasło powinno być haszowane z unikalną solą.
- **Wynikowy rozmiar hasza** (`Output Size`): Określa długość wynikowego hasza. Długość ta jest zwykle wyrażona w bajtach.

2.2 Narzędzia

Hashcat

To potężne narzędzie do łamania haseł, które jest często używane przez specjalistów ds. bezpieczeństwa informatycznego do testowania i oceny siły haseł oraz algorytmów haszujących. Poniżej przedstawiam kilka kluczowych cech Hashcata:

- **Wsparcie dla wielu algorytmów haszujących**: Hashcat obsługuje szeroki zakres algorytmów haszujących, w tym popularne funkcje takie jak MD5, SHA-1, SHA-256, bcrypt, scrypt i wiele innych.
- **Zaawansowane techniki ataków**: Narzędzie Hashcat oferuje różnorodne techniki ataków, takie jak ataki brute-force, ataki słownikowe, ataki kombinatoryczne, ataki hybrydowe i wiele innych, co pozwala na skuteczne łamanie haseł przy różnych scenariuszach i konfiguracjach.
- **Wsparcie dla platform sprzętowych**: Hashcat korzysta z potęgi obliczeniowej kart graficznych (GPU), co znacznie przyspiesza proces łamania haseł. Ponadto, jest również kompatybilny z CPU, co daje większą elastyczność w wyborze platformy sprzętowej.
- **Możliwość pracy na wielu platformach**: Hashcat jest dostępny na różnych systemach operacyjnych, w tym na Windowsie, Linuxie i macOS, co czyni go wszechstronnym narzędziem dostępnym dla szerokiego grona użytkowników.
- **Otwarte oprogramowanie**: Hashcat jest projektem open source, co oznacza, że jego kod jest dostępny publicznie i może być modyfikowany i rozwijany przez społeczność, co sprzyja ciągłemu ulepszaniu i dostosowywaniu narzędzia do różnych potrzeb i zastosowań.

OpenSSL

To wieloplatformowa, otwarta implementacja protokołów SSL (wersji 2 i 3) i TLS (wersji 1) oraz algorytmów kryptograficznych ogólnego przeznaczenia. Dostępna jest dla systemów uniksopodobnych (m.in. Linux, BSD, Solaris), OpenVMS i Microsoft Windows. OpenSSL zawiera biblioteki implementujące wspomniane standardy oraz mechanizmy kryptograficzne, a także zestaw narzędzi konsolowych (przede wszystkim do tworzenia kluczy oraz certyfikatów, zarządzania urzędem certyfikacji, szyfrowania, dekryptażu i obliczania podpisów cyfrowych). OpenSSL pozwala na używanie wszystkich zastosowań kryptografii. Poniżej kilka kluczowych cech OpenSSL:

- **Wszechstronność**: OpenSSL oferuje bogaty zestaw narzędzi i bibliotek do obsługi różnych protokołów kryptograficznych, w tym SSL/TLS, kryptografię klucza publicznego, szyfrowanie danych i wiele innych.
- **Bezpieczeństwo sieciowe**: OpenSSL dostarcza narzędzia do zarządzania certyfikatami, generowania kluczy, podpisywania cyfrowego i szyfrowania danych, co umożliwia tworzenie bezpiecznych aplikacji internetowych i usług.
- **Wsparcie dla różnych platform**: OpenSSL jest dostępny na wielu platformach, w tym na systemach Unix, Linux, macOS, Windows oraz innych, co czyni go popularnym narzędziem wśród programistów i administratorów systemów.

- **Otwarty kod źródłowy**: OpenSSL jest projektem open-source, co oznacza, że jego kod jest dostępny publicznie i może być modyfikowany oraz rozwijany przez społeczność, co sprzyja ciągłemu ulepszaniu i dostosowywaniu narzędzia do różnych potrzeb i zastosowań.
- **Wsparcie dla wielu protokołów**: OpenSSL obsługuje wiele standardowych protokołów kryptograficznych, takich jak SSL/TLS, SSH, S/MIME, PKCS, co czyni go wszechstronnym narzędziem do implementacji bezpiecznych komunikacji w różnych aplikacjach i systemach.

Linki

- https://en.wikipedia.org/wiki/Hash_function
- https://cryptobook.nakov.com/cryptographic-hash-functions
- https://www.baeldung.com/cs/hashing
- https://www.cyber.mil.pl/funkcje-skrotu/
- https://informatykzakladowy.pl/jak-serwer-sprawdza-haslo
- https://www.openssl.org/
- https://hashcat.net/hashcat/
- https://hashcat.net/wiki/doku.php?id=example_hashes
- https://kapitanhack.pl/2018/12/22/hack-tools/narzedzie-do-hackowania-offline-czyli-hashcat/
- https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
- https://pl.wikipedia.org/wiki/Potok_(Unix)
- https://manpages.ubuntu.com/manpages/impish/pl/man1/head.1.html
- https://manpages.ubuntu.com/manpages/impish/pl/man1/cut.1.html
- https://manpages.ubuntu.com/manpages/impish/pl/man1/base64.1.html
- https://manpages.ubuntu.com/manpages/impish/pl/man1/tr.1.html
- http://linuxwiki.pl/wiki/Tr
- https://linux.die.net/man/4/urandom
- https://unix.stackexchange.com/questions/324209/when-to-use-dev-random-vs-dev-urandom
- https://tools.kali.org/password-attacks/crunch
- https://www.openwall.com/john/
- https://www.openwall.com/john/doc/EXAMPLES.shtml
- https://www.openwall.com/john/doc/OPTIONS.shtml
- https://www.varonis.com/blog/john-the-ripper/
- https://www.soisk-me.pl/klasa-iii-linux/plik-passwd-i-shadow
- https://chameleonstales.blogspot.com/2019/04/co-to-jest-za-plik-etcpasswd-i-etcshadow.html
- https://docs.python.org/3/library/hashlib.html
- http://manpages.ubuntu.com/manpages/impish/man1/fcrackzip.1.html
- https://www.cyberpratibha.com/blog/add-kali-linux-repository/
- https://hashcat.net/wiki/doku.php?id=example_hashes
- https://miloserdov.org/?p=5477
- https://countuponsecurity.files.wordpress.com/2016/09/jtr-cheat-sheet.pdf
- https://argon2.online/
- https://www.youtube.com/watch?v=xwfnqTjfApA&ab_channel=SekurakTV
- https://sekurak.pl/jak-szybko-przestepcy-moga-zlamac-hashowanie-hasel-wykorzystane-w-morele-net/

Zadania

Hashowanie

1.1 Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/bsk-book-p1-ch1-ex001:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 `127.0.0.1` na porcie TCP o numerze `10001` działa serwer obsługujący protokół HTTP w wersji 1.1. Serwer udostępnia 2 endpointy:

- `/random` - generuje losowe słowo
- `/check_md5/[hash]` - sprawdza, czy hash podany jako argument jest prawidłowym hashem MD5 wylosowanego słowa

Aby rozwiązać zadanie:

- Wylosuj słowo za pomocą endpointa `/random`
- Następnie, pomocą narzędzia `OpenSSL` wygeneruj hash MD5 wylosowanego słowa. Wykorzystując endpoint `/check_md5/[hash]` sprawdź, czy wygenerowałeś prawidłowy hash MD5.
- Do rozwiązania zadania możesz użyć przeglądarki lub narzędzia `cURL`.

1.2 Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/bsk-book-p1-ch1-ex002:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 `127.0.0.1` na porcie TCP o numerze `10002` działa serwer obsługujący protokół HTTP w wersji 1.1. Serwer udostępnia 2 endpointy:

- `/random` - generuje losowe słowo
- `/check_sha256/[hash]` - sprawdza, czy hash podany jako argument jest prawidłowym hashem SHA-256 wylosowanego słowa

Aby rozwiązać zadanie:

- Wylosuj słowo za pomocą endpointa `/random`
- Następnie, pomocą narzędzia `OpenSSL` wygeneruj hash SHA-256 wylosowanego słowa. Wykorzystując endpoint `/check_sha256/[hash]` sprawdź, czy wygenerowałeś prawidłowy hash SHA-256.
- Do rozwiązania zadania możesz użyć przeglądarki lub narzędzia `cURL`.

1.3 Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/bsk-book-p1-ch1-ex003:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 `127.0.0.1` na porcie TCP o numerze `10003` działa serwer obsługujący protokół HTTP w wersji 1.1. Serwer udostępnia 2 endpointy:

- `/random` - generuje losowe słowo
- `/check_sha512/[hash]` - sprawdza, czy hash podany jako argument jest prawidłowym hashem SHA-512 wylosowanego słowa

Aby rozwiązać zadanie:

- Wylosuj słowo za pomocą endpointa `/random`
- Następnie, pomocą narzędzia `OpenSSL` wygeneruj hash SHA-512 wylosowanego słowa. Wykorzystując endpoint `/check_sha512/[hash]` sprawdź, czy wygenerowałeś prawidłowy hash SHA-512.
- Do rozwiązania zadania możesz użyć przeglądarki lub narzędzia `cURL`.

1.4 Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/bsk-book-p1-ch1-ex004:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 `127.0.0.1` na porcie TCP o numerze `10004` działa serwer obsługujący protokół HTTP w wersji 1.1. Serwer udostępnia 4 endpointy:

- `/random` - generuje losowe słowo
- `/check_argon2d/[hash]` - sprawdza, czy hash podany jako argument jest prawidłowym hashem Argon2D wylosowanego słowa
- `/check_argon2i/[hash]` - sprawdza, czy hash podany jako argument jest prawidłowym hashem Argon2I wylosowanego słowa
- `/check_argon2id/[hash]` - sprawdza, czy hash podany jako argument jest prawidłowym hashem Argon2ID wylosowanego słowa

Parametry dla wszystkich wersji algorytmu Argon:

- Pamięć: 8192
- Liczba wątków: 1
- Sól: `NaCl2024`
- Liczba iteracji: 1
- Rozmiar hasha: 24

Aby rozwiązać zadanie:

- Wylosuj słowo za pomocą endpointa `/random`
- Następnie, pomocą narzędzia `OpenSSL` wygeneruj hash za pomocą algorytmów Argon2D, Argon2I, Argon2ID wylosowanego słowa.
- Wykorzystując endpointy `/check_argon2d/[hash]`, `/check_argon2i/[hash]` oraz `/check_argon2id/[hash]` sprawdź, czy wygenerowałeś prawidłowe hashe algorytmów Argon2D, Argon2I, Argon2ID.
- Do rozwiązania zadania możesz użyć przeglądarki lub narzędzia `cURL`.

UWAGA Wsparcie dla algorytmu Argon w bibliotece OpenSSL zostało wprowadzone w wersji `3.2`. Jeśli nie posiadasz OpenSSLa w tej wersji, możesz wykorzystać kontener Dockerowy z zainstalowanym OpenSSL-em w wersji `3.3.2`. Uruchom kontener: `docker run -it mazurkatarzyna/openssl-332-ubuntu`.

1.5 Zapisz do pliku `ex13.txt` swoje imię i nazwisko, a następnie wygeneruj hash MD5 tego pliku.

1.6 Zapisz do pliku `ex14.txt` swoje imię i nazwisko. Następnie:

- Za pomocą narzędzia `OpenSSL` wygeneruj hash MD5 pliku `ex14.txt`. Zapisz hash do pliku `res14a.txt`.
- Zmień zawartość pliku `ex14.txt`, dopisz do niego swój numer indeksu.
- Ponownie, za pomocą narzędzia `OpenSSL` wygeneruj hash MD5 pliku `ex14.txt`. Zapisz hash do pliku `res14b.txt`.
- Czy hashe w plikach `res14a.txt` oraz `res14b.txt` są takie same? Dlaczego?

1.7 Za pomocą narzędzia `rand` dostarczonego wraz z pakietem `OpenSSL`, wygeneruj 4 bitowe hasło i zakoduj go za pomocą Base64. Następnie, za pomocą narzędzia `OpenSSL`, wygeneruj hash MD5 tego hasła. Nie korzystaj z pomocniczych plików.

1.8 Za pomocą narzędzia `rand` dostarczonego wraz z pakietem `OpenSSL`, wygeneruj 16 bitowe hasło i zakoduj go za pomocą Base64. Następnie, za pomocą narzędzia `OpenSSL`, wygeneruj hash SHA-512 tego hasła. Nie korzystaj z pomocniczych plików.

1.9 Korzystając z poleceń systemowych (np. `tr`, `head`, `cut`, `base64`) oraz pliku `/dev/urandom` wygeneruj bezpieczne hasło dla użytkownika (hasło ma mieć długość 16 znaków, nie może posiadać nie-alfanumerycznych znaków) a następnie wygeneruj, za pomocą narzędzia `OpenSSL`, funkcję skrótu MD5 dla wygenerowanego wcześniej hasła. Nie korzystaj z pomocniczych plików.

Łamanie hashy, słowniki

1.10 Korzystając z narzędzia `crunch`, wygeneruj do pliku listę haseł składających się z samych cyfr o długości 3 znaków. Zapisz je do pliku. Za pomocą narzędzia `OpenSSL` wygeneruj funkcje skrótu SHA-1 dla wszystkich wygenerowanych haseł.

1.11 Korzystając z narzędzia `crunch`, wygeneruj do pliku listę haseł o długości 5 znaków według wzoru, gdzie:

- pierwszy znak hasła to cyfra
- drugi znak hasła to mała litera *a*
- trzeci znak hasła to znak specjalny
- czwarty znak hasła to mała litera *b*
- piąty znak hasła to cyfra

Zapisz je do pliku. Za pomocą narzędzia `OpenSSL` wygeneruj funkcje skrótu SHA-3 dla wszystkich wygenerowanych haseł.

1.12 Korzystając z narzędzia `crunch`, wygeneruj do pliku listę haseł o długości 3 znaków według wzoru:

- pierwszy znak hasła to litera ze zbioru `{a, b, c}`
- drugi znak hasła to cyfra za zbioru `{4, 6, 8}`
- trzeci znak hasła to znak specjalny ze zbioru: `{?, %, :}`

Przykładowe hasła: `a4?, b6%, c8:, ...`. Zapisz je do pliku. Za pomocą narzędzia `OpenSSL` wygeneruj funkcje skrótu SHA-3 dla wszystkich wygenerowanych haseł.

1.13 Za pomocą słownika `rockyou.txt` oraz programu JohnTheRipper, spróbuj złamać hash MD5: `8afa847f50a716e64932d995c8e7435a`.

1.14 Za pomocą słownika `rockyou.txt` oraz programu JohnTheRipper, spróbuj złamać hash SHA-256: `437d9b521abe3c4102db90f7873cb4699cf9e38476c32b586cb786eb39eb6992`.

1.15 Wykonaj zadanie **1.13** za pomocą narzędzia `hashcat`.

1.16 Użytkownik ma hasło (plik `z5.shadow`) składające się z 3 dowolnych znaków. Jakie to hasło? Do rozwiązania zadania użyj oprogramowania `JohnTheRipper`.

1.17 Za pomocą słownika `rockyou.txt` oraz oprogramowania `JohnTheRipper` sprawdź jakie hasła mają użytkownicy `u1` i `u2` (plik `z2.shadow`). Hasła te mają charakter słów słownikowych.

1.18 Użytkownik ma hasło (plik `z6.shadow`) składające się z 5 dowolnych znaków. Jakie to hasło? Do rozwiązania zadania użyj oprogramowania `JohnTheRipper`.

1.19 W pliku `z7.shadow` jest wiersz z hasłem użytkownika `user1`. Hasło ma charakter słownikowy, przy czym jest zapisane w taki sposób, że niektóre litery zamienione są na cyfry i tak: każde wystąpienie małego ***a*** zamienione jest na ***@***, małego ***i*** na ***1***, małego ***e*** na ***3***. Ponadto hasło ma jeszcze na początku i na końcu znak # (hash). Do rozwiązania zadania użyj oprogramowania `JohnTheRipper`.

1.20 Wykonaj zadanie **1.17** za pomocą narzędzia `hashcat`.

1.21 Wykonaj zadanie **1.16** za pomocą narzędzia `hashcat`.

1.22 Wykonaj zadanie **1.18** za pomocą narzędzia `hashcat`.

1.23 Wykonaj zadanie **1.19** za pomocą narzędzia `hashcat`.

Hashowanie w Pythonie

1.24 Napisz skrypt w języku Python, w którym wygenerujesz hash MD5 dowolnego ciągu znaków podawanego jako argument wywołania skryptu. Sprawdź poprawność wygenerowanego hasha porównując go z wynikiem otrzymanym przy pomocy `md5sum` lub `openssl`.

1.25 Napisz skrypt w języku Python, w którym wygenerujesz hash SHA-1 dowolnego pliku podawanego jako argument wywołania skryptu. Sprawdź poprawność wygenerowanego hasha porównując go z wynikiem otrzymanym przy pomocy `sha1sum` lub `openssl`.

1.26 Mając dany początkowy ciąg znaków `R3iSrSNmgU9SFHxVekUD`, który następnie został zahashowany, określ, jaka funkcja skrótu została wykorzystana do utworzenia hasha: `48cab4b54bef42fddaa6353c68a20b369f40026e`.

Hashowanie i integralność

1.27 Ze strony `https://gparted.org/download.php` pobierz plik `gparted-live-1.3.1-1-amd64.iso`. Za pomocą `OpenSSL` sprawdź integralność pobranego pliku. (*Integralność polega na zapewnieniu, że przetwarzana informacja nie została w żaden sposób zmieniona. Zmiana taka może być przypadkowa (błąd podczas transmisji) jak i celowa (zmiana przez atakującego)*).

1.28 Sprawdź, czy pliki `a.txt` oraz `b.txt` mają taką samą zawartość.