

Achtergrondinformatie

- Zoals je hebt geleerd in de activiteiten vier en vijf, is een **cryptografische hashfunctie** een eenrichtingsberekening; het oorspronkelijke wachtwoord kan niet omgekeerd worden berekend vanuit de hash. Wanneer een hacker in een computer inbreekt en de wachtwoord-hashes steelt, kunnen ze deze niet gebruiken om in te loggen, maar ze kunnen wel gebruikt worden in een **rainbow table** of een brute-force aanval.
- Een brute-force aanval creëert alle **permutaties** van mogelijke **platte tekst** wachtwoorden, berekent de hash en vergelijkt deze met de gestolen hash. Een brute-force aanval is een langdurig proces dat **veel rekentijd** vereist.
- Een reden waarom deze aanval zoveel rekentijd vereist, is het aantal permutaties voor een **wachtwoordregel**.
- De permutaties van een wachtwoordregel vertegenwoordigen alle manieren waarop het wachtwoord kan worden gevormd op basis van het aantal en de soorten vereiste tekens in de regel.
- Bijvoorbeeld, een wachtwoordregel met precies vier tekens, die alleen hoofdletters gebruikt, heeft 1 van de 26 verschillende tekens op elke positie. Het aantal manieren waarop een wachtwoord op basis van deze regel kan worden gevormd, is $26 \times 26 \times 26 \times 26 = 456.976$, of 26^4 verschillende wachtwoordpermutaties!
- Het bovenstaande voorbeeld kan veralgemeend worden als:

aantal mogelijke tekens ^{lengte van het wachtwoord}

- Regels die lange wachtwoorden vereisen die zijn gevormd uit veel mogelijke tekens, zijn moeilijker te breken via een brute-force aanval, simpelweg omdat er veel meer mogelijke wachtwoorden zijn om te testen. Bijvoorbeeld, een regel die acht tekens vereist, bestaande uit een kleine letter, een hoofdletter, een cijfer en een speciaal teken (er zijn 32 speciale tekens) heeft $(26+26+10+32)^8 = 6.095.689.385.410.816$ mogelijke wachtwoorden!
- Lange wachtwoorden die uit veel verschillende soorten tekens bestaan en die regelmatig worden veranderd, verkleinen de kans dat je slachtoffer wordt van een brute-force aanval.

Wat is jouw opdracht?

1. Oefenen met wachtwoordsterkte (permutaties):
 - a. Ga naar pagina 1.3, "**mogelijke_wachtwoorden.py**", en bekijk de Python-code. Zie je hoe drie geneste lussen elk mogelijk wachtwoord genereren waarbij drie hoofdletters mogen worden gebruikt? Voer het programma uit en observeer hoe het werkt. Zie je hoe de drie geneste lussen de permutaties creëren? Hoeveel mogelijke wachtwoorden kunnen er worden gemaakt van drie hoofdletters? Opmerking: **Dit programma duurt ongeveer tien minuten om te voltooien!** Als je ongeduldig wordt, houd dan de [on]-toets ingedrukt om het programma te onderbreken.
 - b. Ga naar de rekentoolpagina en bekijk de voorbeeldberekening. Zie je hoe de wachtwoordregel die drie hoofdletters vereist, 17.576 mogelijke wachtwoorden oplevert? Heeft deze berekening het aantal wachtwoorden opgeleverd dat het vorige Python-programma genereerde?
 - c. Ga naar de volgende rekentoolpagina met een probleemopdracht en voltooi de oefenberekening. Plaats de cursor onder de probleemopdracht en gebruik de rekenmachineknoppen om de berekening in te voeren. Kreeg je 3.226.266.762.397.899.821.056? Kun je bedenken hoe je de geneste lussen in het Python-programma zou aanpassen om elk

TI-Nspire CX II

mogelijk wachtwoord te genereren op basis van de nieuwe regel? Kun je je voorstellen hoe lang het zou duren om het programma uit te voeren?

- d. Ga naar de volgende grafiekpagina. Gebruik de traceerfunctie om de grafiek van het aantal wachtwoordpermutaties (y) als functie van de wachtwoordlengte (x) te verkennen. Wat valt je op over de vorm van deze curve? Aangezien veel permutaties meer rekentijd vereisen, wat impliceert de grafiek dan over de wachtwoordlengte en de kwetsbaarheid voor brute-force aanvallen?
2. Stel een wachtwoord in op je micro:bit.
 - a. Verbind je micro:bit met je rekenmachine.
 - b. Ga naar pagina 2.1 en voer het programma uit. Maak een wachtwoord met als voorwaarde: vier tekens bestaande uit alleen letters. Opmerking: **Kies voor een wachtwoord met een eerste letter uit het begin van het alfabet** om de rekentijd later in de activiteit "brute_force.py" te verkorten. Bijvoorbeeld, "Abba" of "Abby" zal VEEL sneller gekraakt worden dan "Zeno" of "Zola." Ook is "ABBA" sneller dan "abba." Kun je naar de Python-code kijken en uitleggen waarom? Opmerking: als je rekenmachine er te lang over doet, houd dan de [on]-toets ingedrukt om het programma te onderbreken.
 3. De brute-force aanval:
 - a. Wissel jouw micro:bit uit met die van een groepslid. *Vertel hem/haar jouw wachtwoord niet!*
 - b. Ga naar pagina 3.1 en voer het programma uit. De hash van het wachtwoord wordt opgehaald uit het bestand "wachtwoord.txt" en weergegeven. Dit is het bestand en de hash die een hacker probeert te stelen om de beveiliging van je micro:bit te omzeilen! Opmerking: de hash kan niet worden omgekeerd; de brute-force aanval is echter een omweg die het platte tekst-wachtwoord van het gehashte micro:bit wachtwoord vindt.
 - c. Na het uitvoeren van het programma in de Python-shell op pagina 3.2, druk je op de [var]-toets. Merk je de variabele genaamd "hacked_hash" op? Selecteer de variabele uit het menu en zie dat deze de 256-bits hash van je wachtwoord retourneert naar de Python REPL-prompt >>> in de shell. Opmerking: je zult deze variabele gebruiken in stap 5, de remote login-activiteit.
 - d. Ga naar de volgende pagina met "brute_force.py" en bekijk het programma. Merk op dat er geen printopdrachten binnen de vier geneste lussen staan. Uitvoer van welke aard dan ook, zoals het afdrukken naar het scherm, is relatief langzaam. Het elimineren van print opdrachten maakt het programma veel sneller. Het **optimaliseren** van vaak herhaalde code is essentieel bij het schrijven van programma's. Voer het programma uit en merk het aantal iteraties en de verstreken tijd op. Vergelijk je verstreken tijd met anderen in je groep. Het programma zal het platte tekst-wachtwoord retourneren met een hash die overeenkomt met de hash. Onthoud het platte tekst-wachtwoord om het in de volgende activiteit te testen.
 4. Authenticatie:
 - a. Ga naar pagina 4.1, "**authenticatie.py**" en voer het programma uit. Voer drie keer een verkeerd wachtwoord in. Na het bekijken van de Python-code, kun je uitleggen hoe het programma herhaalde authenticatiepogingen voorkomt? Hoe draagt het beperken van het aantal pogingen bij aan de veiligheid?
 - b. Voer het programma opnieuw uit met het gekraakte platte tekst-wachtwoord om het resultaat van de brute-force aanval te testen. Is het hacken gelukt?

5. Remote login:

- De **ontvanger**
 - Wijzig het wachtwoord zoals beschreven in activiteit 2. Oefen goede **wachtwoordhygiëne** en hergebruik hetzelfde wachtwoord niet. Kies een wachtwoord dat aan het begin van het alfabet ligt. Fluister je wachtwoord naar de verzender zodat de verzender in kan loggen op je micro:bit. Zorg ervoor dat je het wachtwoord geheim houdt voor hackers. Ga naar **'student_ontvanger.py'**, wijzig de groep naar het toegewezen nummer en voer het programma uit voordat de zender het programma heeft uitgevoerd.
- De **zender**
 - Ga naar **'student_zender.py'**, wijzig de groep naar het toegewezen nummer en voer je programma uit nadat de ontvanger en hacker hun programma's hebben gestart. Gebruik het wachtwoord van de ontvanger om vanop afstand op hun micro:bit in te loggen.
- De **hacker**
 - Ga naar **'student_hacker.py'**, wijzig de groep naar het toegewezen nummer en voer het programma uit voordat de zender het programma heeft uitgevoerd. Opmerking: dit programma ontvangt de wachtwoord-hash die door de zender wordt verzonden om in te loggen op de micro:bit van de ontvanger.
 - Gebruik de gestolen hash en de `brute_force_kraken` module om het wachtwoord van de ontvanger te kraken vanuit de onderschepte hash. Gebruik de `"brute(hacked_hash)"` functie vanuit deze module om de hash te hacken. Typ hiervoor `>>>brute(hacked_hash)` in de shell in. De variabele `"hacked_hash"` kan geselecteerd worden met de `[var]-toets`.
- Nadat de hacker het wachtwoord van de ontvanger heeft gekraakt, moet de ontvanger het **'student_ontvanger.py'**-programma opnieuw uitvoeren om te testen of de hacker in staat is om toegang te krijgen tot je micro:bit. De hacker moet naar **'student_zender.py'** gaan en het gekraakte platte tekst-wachtwoord versturen. Opmerking: als de hacker succesvol is, zou hij in staat moeten zijn om in te loggen op de micro:bit van de ontvanger zonder ooit het wachtwoord te hebben gekregen!

De code

Zender

```
student_zender.py 10/10
from microbit_radio import *
from hashing import *
# Gebruik het wachtwoord van de ontvanger
kanaal = 1
groep = 1
clear_history()
wachtwoord = input("Voer wachtwoord in: ")
wachtwoord_hash = sha_hash(wachtwoord)
tx(wachtwoord_hash,kanaal,groep)
```

Ontvanger

```
student_ontvanger.py 11/14
from microbit_radio import *
from hashing import *
# Deel je wachtwoord met de zender
kanaal = 1
groep = 1
clear_history()
test_hash = rx(kanaal,groep)
authentic_hash = read_file("wachtwoord.txt")
if test_hash == authentic_hash:
    display.show(Image.YES)
    music.play(music.BA_DING)
```

Hacker

```
student_hacker.py 1/14
from microbit_radio import *
from brute_force_kraken import *
# De hacker zal brute force gebruiken om het
# wachtwoord te vinden wat verzonden is naar
# de ontvanger.
kanaal = 1
groep = 1
clear_history()
print("Man-in-the-middle aanval!")
hacked_hash = rx(kanaal,groep)
print("hash string = {}".format(hacked_hash))
```

Extra uitdagingen

- Elk teamlid zou iedere rol een keer geprobeerd moeten hebben, om zo elkaars wachtwoord te hacken.
- Time hoe lang het duurt om een wachtwoord beginnende met de letter "A" te kraken, herhaal dit voor een wachtwoord met de beginletter "B". Ga zo verder tot aan de letter "E". Maak nu een grafiek met de letters op de x-as: A = 1, B = 2, etc., en de tijd om het wachtwoord te kraken op de y-as. Kan je de vorm van deze grafiek voorspellen?

Samengevat

- Een brute-force aanval is een rekenintensieve hack die de hash berekent van alle permutaties van wachtwoorden voor een gegeven wachtwoordregel en deze vergelijkt met een gestolen wachtwoordhash. Als de hashes hetzelfde zijn, wordt het platte tekst-wachtwoord voor de gestolen hash ontdekt.
- Het regelmatig veranderen van je wachtwoord, het gebruiken van niet frequent gebruikte tekens, het niet hergebruiken van hetzelfde wachtwoord voor verschillende accounts, en het niet delen of opschrijven van wachtwoorden helpt je accounts te beschermen tegen hackers.

Tips voor als het misgaat

- Controleer of iedereen in het team hun toegewezen groepsnummer gebruikt.
- Zorg ervoor dat de ontvanger en hacker hun programma's uitvoeren en wachten voordat de verzender het bericht verzendt.
- Onthoud, de zender probeert in te loggen op de micro:bit van de ontvanger en moet het wachtwoord voor de micro:bit van de ontvanger sturen.