LISV Laboratoire d'ingénierie des systèmes de Versailles | UVSQ université PARIS-SACLAY

Programmation sécurisée (Secure coding)
TP 1
Utilisation d'une chaîne de formatage contrôlée par une source externe

Halim Djerroud

révision 1.0

— Exercices avec solutions —

1 – CWE-134 –

Exercice 1 : Chaînes et Formatage

Écrivez un programme qui demande à l'utilisateur son nom et son âge, puis affiche un message formaté de manière sécurisée.

Solution :

```
name = input("Quel est votre nom ? ")
age = input("Quel est votre âge ? ")

if not age.isdigit():
    print("Erreur : L'âge doit être un nombre entier.")
else:
    age = int(age)
    print(f"Bonjour, {name} ! Vous avez {age} ans.")
```

Conseils :

- Toujours valider les entrées utilisateur, en particulier lorsque vous attendez un type spécifique (comme un entier).
- Utilisez des chaînes formatées (f-strings) ou des méthodes sécurisées comme `str.format`.

Ce qu'il ne faut pas faire :

```
# Mauvaise pratique
name = input("Quel est votre nom ? ")
age = input("Quel est votre âge ? ")
print("Bonjour, " + name + "! Vous avez " + age + " ans.")
```

Problème : Aucun contrôle n'est effectué pour s'assurer que `age` est un entier.

Exercice 2 : Validation des Entrées

Écrire un programme qui demande à l'utilisateur de saisir un nombre entier entre 1 et 100 inclus. Si l'utilisateur entre autre chose, affichez un message d'erreur et redemandez l'entrée.

Solution :

```
while True:
    user_input = input("Entrez un nombre entre 1 et 100 : ")
    if user_input.isdigit():
        number = int(user_input)
        if 1 <= number <= 100:
            print(f"Merci ! Vous avez entré : {number}")
            break
        else:
            print("Erreur : Le nombre doit être entre 1 et 100.")
    else:
        print("Erreur : Veuillez entrer un nombre valide.")
```

Conseils :

— Vérifiez si l'entrée est un entier avant de la convertir.
— Gérez les cas hors limites après la conversion.

Ce qu'il ne faut pas faire :

```
# Mauvaise pratique
user_input = input("Entrez un nombre entre 1 et 100 : ")
number = int(user_input)  # Pas de vérification préalable
if number < 1 or number > 100:
    print("Erreur.")
```

Problème :

Si l'entrée est une chaîne non numérique (par ex., "abc"), le programme plante.

Exercice 3 : Calculatrice Sécurisée

Écrire programme calculatrice qui permet à l'utilisateur d'effectuer une addition, une soustraction, une multiplication ou une division. Ne pas utiliser `eval`.

Solution :

```
def calculer(a, b, operation):
    if operation == '+':
        return a + b
    elif operation == '-':
        return a - b
    elif operation == '*':
        return a * b
    elif operation == '/':
        if b != 0:
            return a / b
        else:
            return "Erreur : Division par zéro."
    else:
        return "Erreur : Opération invalide."

try:
    a = float(input("Entrez le premier nombre : "))
    b = float(input("Entrez le second nombre : "))
    operation = input("Choisissez une opération (+, -, *, /) : ")
    resultat = calculer(a, b, operation)
```

```
    print(f"Résultat : {resultat}")
except ValueError:
    print("Erreur : Veuillez entrer des nombres valides.")
```

Ce qu'il ne faut pas faire :

```
# Mauvaise pratique
result = eval(f"{a} {operation} {b}")  # Dangereux
```

Problème :

L'utilisation de `eval` permet d'exécuter du code malveillant. Cela permet à un attaquant d'exécuter n'importe quel code Python en fournissant une entrée malveillante, comme :

```
Entrez le premier nombre : __import__('os').system('rm -rf /')
```

Exercice 4 : Journalisation d'Activités Utilisateur

Écrire un programme qui journalise les actions des utilisateurs dans un fichier `log.txt` avec un horodatage.

Solution :

```
import datetime

def journaliser(action):
    date_heure = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    with open("log.txt", "a") as log_file:
        log_file.write(f"[{date_heure}] Action de l'utilisateur : {action}\n")

action = input("Entrez une action (Connexion, Déconnexion, etc.) : ")
journaliser(action)
print("Action journalisée avec succès.")
```

Exercice 5 : Analyseur de Fichiers Sécurisé

Écrire un programme qui lit un fichier CSV contenant des produits et calcule leur valeur totale (prix × quantité).

Produit	Prix	Quantité
Stylo	1.5	100
Cahier	3.2	50
Trousse	8.5	20
Sac	25.0	10
Livre	15.0	5

TABLE 1 – Exemple de contenu du fichier CSV `produits.csv`

Solution :

```
import csv

def calculer_valeur_totale(fichier_csv):
    try:
        total = 0.0
        with open(fichier_csv, mode="r") as file:
            reader = csv.DictReader(file)
            for row in reader:
```

```
                try:
                    prix = float(row["Prix"])
                    quantite = int(row["Quantité"])
                    total += prix * quantite
                except (ValueError, KeyError):
                    print(f"Erreur dans la ligne : {row}")
        return total
    except FileNotFoundError:
        print("Erreur : Fichier non trouvé.")
        return None

fichier = "produits.csv"
valeur_totale = calculer_valeur_totale(fichier)
if valeur_totale is not None:
    print(f"Valeur totale des produits en stock : {valeur_totale}")
```

Ce qu'il ne faut pas faire :

```
# Mauvaise pratique
produit, prix, quantite = ligne.strip().split(",")
total += float(prix) * int(quantite)  # Pas de gestion des erreurs
```

Problème :

Aucune gestion des erreurs pour des données mal formatées.

Exercice 6

Etudier le code suivant :

```
# SPDX-FileCopyrightText: OpenSSF project contributors
# SPDX-License-Identifier: MIT
""" Non-compliant Code Example """
import sys

# Simulating a global include of sensitive information:
ENCRYPTION_KEY = "FL4G1"

# Simulating a include per language:
MESSAGE = "Contract '{0.instance_name}' created for "


class MicroService:
    """Fancy MicroService"""
    def __init__(self, instance_name):
        self.instance_name = instance_name


def front_end(customer):
    """Display service instance"""
    message_format = MESSAGE + customer
    mc = MicroService("big time microservice")
    print(message_format.format(mc))


####################
# exploiting above code example
####################
```

```python
if __name__ == "__main__":
    if len(sys.argv) > 1:  # running from command line
        # you can print the global encryption key by using
        #'{0.__init__.__globals__[ENCRYPTION_KEY]}' as
        # argument.
        front_end(sys.argv[1])
    else:
        # running in your IDE, simulating command line:
        # Printing the ENCRYPTION_KEY via the global accessible object
        front_end("{0.__init__.__globals__[ENCRYPTION_KEY]}")
```

Solution :

(Exemple est solution extrait : https://github.com/ossf/wg-best-practices-os-developers/blob/main/docs/Secure-Coding-Guide-for-Python/CWE-664/CWE-134/README.md)

```python
# SPDX-FileCopyrightText: OpenSSF project contributors
# SPDX-License-Identifier: MIT
""" Compliant Code Example """
import sys
from string import Template

# Simulating a global include of sensitive information:
ENCRYPTION_KEY = "FL4G1"

# Simulating a include per language for international support:
MESSAGE = Template("Contract '$instance_name' created for '$customer'")


class MicroService:
    """Fancy MicroService"""
    def __init__(self, instance_name):
        self.instance_name = instance_name

    def get_instance_name(self) -> str:
        """return instance_name as string"""
        return self.instance_name


def front_end(customer):
    """Display service instance"""
    mc = MicroService("big time microservice")
    print(MESSAGE.substitute(instance_name=mc.get_instance_name(),
                             customer=customer))


######################
# exploiting above code example
######################
if __name__ == "__main__":
    if len(sys.argv) > 1:  # running from command line
        # you can print the global encryption key by using '{0.__init__.__globals__[ENCRYPTION_KEY]}' a
        # argument.
        front_end(sys.argv[1])
    else:
        # running in your IDE, simulating command line:
        # Printing the ENCRYPTION_KEY via the global accessible object
        front_end("{0.__init__.__globals__[ENCRYPTION_KEY]}")
```

2 – CWE-197 –

Exercice 1 :

Corriger le code suivant pour qu'il soit compatible avec : CWE-197

```
value = float(0.0)
while value <= 1:
    print(value)
    value = value + float(1.0/9.0)
```

Solution :

```
value = 0
while value <= 9:
    print(value / 9)
    value = value + 1
```

Exercice 2 :

Corriger le code suivant pour qu'il soit compatible avec : CWE-197

```
value = float(1.0) + float("1e-18")
target = float(1.0) + float("1e-17")
while value <= target:
    print(value)
    value = value + float("1e-18")
```

Solution :

```
value = 1
while value <= 10:
    print(f"{value / 10 ** 14:.14f}")
    value = value + 1
```

3 – CWE-754 –

Exercice 1 :

Corriger le code suivant pour qu'il soit compatible avec : CWE-754

```
import sys


class Package:
    def __init__(self):
        self.package_weight = float(1.0)

    def put_in_the_package(self, object_weight):
        value = float(object_weight)
        print(f"Adding an object that weighs {value} units")
        self.package_weight += value

    def get_package_weight(self):
        return self.package_weight


####################
```

```
# exploiting above code example
#####################
package = Package()
print(f"\nOriginal package's weight is {package.get_package_weight():.2f} units\n")
for item in [sys.float_info.max, "infinity", "-infinity", "nan"]:
    try:
        package.put_in_the_package(item)
        print(f"Current package weight = {package.get_package_weight():.2f}\n")
    except Exception as e:
        print(e)
```

Solution :

```
import sys
from math import isinf, isnan


class Package:
    def __init__(self):
        self.package_weight = float(1.0)

    def put_in_the_package(self, user_input):
        try:
            value = float(user_input)
        except ValueError:
            raise ValueError(f"{user_input} - Input is not a number!")

        print(f"value is {value}")

        if isnan(value) or isinf(value):
            raise ValueError(f"{user_input} - Input is not a real number!")

        if value < 0:
            raise ValueError(
                f"{user_input} - Packed object weight  cannot be negative!"
            )

        if value >= sys.float_info.max - self.package_weight:
            raise ValueError(f"{user_input} - Input exceeds acceptable range!")
        self.package_weight += value

    def get_package_weight(self):
        return self.package_weight


#####################
# exploiting above code example
#####################
package = Package()
print(f"\nOriginal package's weight is {package.get_package_weight():.2f} units\n")
for item in [sys.float_info.max, "infinity", "-infinity", "nan"]:
    try:
        package.put_in_the_package(item)
        print(f"Current package weight = {package.get_package_weight():.2f}\n")
    except Exception as e:
        print(e)
```

4 – CWE-1109 –

Exercice 1 :

Corriger le code suivant pour qu'il soit compatible avec : CWE-1109

```
numbers = ["one", "two", "three"]

print(f"len({numbers}) == {len(numbers)}")

def len(x):
    """ implementing a dodgy version of a build in method """
    return sum(1 for _ in x) + 1

print(f"len({numbers}) == {len(numbers)}")
```

Solution :

```
numbers = ["one", "two", "three"]

print(f"len({numbers}) == {len(numbers)}")


def custom_len(x):
    """ implementing a dodgy version of a build in method """
    return sum(1 for _ in x) + 1


print(f"len({numbers}) == {len(numbers)}")
```

5 – CWE-1339 –

Exercice 1 :

Corriger le code suivant pour qu'il soit compatible avec : CWE-1339

```
balance = 3.00
item_cost = 0.33
item_count = 5

######################
# exploiting above code example
######################
print(
    f"{str(item_count)} items bought, ${item_cost} each. "
    f"Current account balance: ${str(balance - item_count * item_cost)}"
)
```

Solution :

```
balance = 300
item_cost = 33
item_count = 5

######################
# exploiting above code example
######################
print(
```

```
    f"{str(item_count)} items bought, ${item_cost / 100} each. "
    f"Current account balance: ${str((balance - item_count * item_cost) / 100)}"
)
```

6 – CWE-392 –

Exercice 1 :

Corriger le code suivant pour qu'il soit compatible avec : CWE-392

```
import math
from concurrent.futures import ThreadPoolExecutor


def get_sqrt(a):
    return math.sqrt(a)


def run_thread(var):
    with ThreadPoolExecutor() as executor:
        return executor.submit(get_sqrt, var)

######################
# exploiting above code example
######################
# get_sqrt will raise ValueError that will be suppressed by the ThreadPoolExecutor
arg = -1
result = run_thread(arg)  # The outcome of the task is unknown
```

Solution :

```
import math
from concurrent.futures import ThreadPoolExecutor


def get_sqrt(a):
    return math.sqrt(a)


def run_thread(var):
    with ThreadPoolExecutor() as executor:
        future = executor.submit(get_sqrt, var)
        if future.exception() is not None:
            # handle exception...
            raise ValueError(f"Invalid argument: {var}")
        return future


######################
# exploiting above code example
######################
arg = -1
result = run_thread(arg)  # Now code execution will be interrupted by ValueError
```

7 – CWE-230 –

Exercice 1 :

Corriger le code suivant pour qu'il soit compatible avec : CWE-230

```
def balance_is_positive(value: str) -> bool:
    """Returns True if there is still enough value for a transaction"""
    _value = float(value)
    if _value == float("NaN") or _value is float("NaN") or _value is None:
        raise ValueError("Expected a float")
    if _value <= 0:
        return False
    else:
        return True


#####################
# attempting to exploit above code example
#####################
print(balance_is_positive("0.01"))
print(balance_is_positive("0.001"))
print(balance_is_positive("NaN"))
```

Solution :

```
from decimal import ROUND_DOWN, Decimal


def balance_is_positive(value: str) -> bool:
    """Returns True if there is still enough value for a transaction"""
    # TODO: additional input sanitation for expected type
    _value = Decimal(value)
    # TODO: exception handling
    return _value.quantize(Decimal(".01"), rounding=ROUND_DOWN) > Decimal("0.00")


#####################
# attempting to exploit above code example
#####################
print(balance_is_positive("0.01"))
print(balance_is_positive("0.001"))
print(balance_is_positive("NaN"))
```

8 – CWE-838 –

Exercice 1 :

Corriger le code suivant pour qu'il soit compatible avec : CWE-838

```
def report_record_attack(stream: bytearray):
    print("important text:", stream.decode("utf-8"))

#####################
# attempting to exploit above code example
#####################
payload = bytearray("user:  attempted a directory traversal".encode("utf-8"))
```

```
# Introducing an error in the encoded text, a byte
payload[3] = 128
report_record_attack(payload)
```

Solution :

```
import base64


def report_record_attack(stream: bytearray):
    try:
        decoded_text = stream.decode("utf-8")
    except UnicodeDecodeError as e:
        # Encode the stream using Base64 if there is an exception
        encoded_payload = base64.b64encode(stream).decode("utf-8")
        # Logging encoded payload for forensic analysis
        print("Base64 Encoded Payload for Forensic Analysis:", encoded_payload)
        print("Error decoding payload:", e)
    else:
        print("Important text:", decoded_text)


####################
# attempting to exploit above code example
####################
payload = bytearray("user:  attempted a directory traversal".encode("utf-8"))
# Introducing an error in the encoded text, a byte
payload[3] = 128
report_record_attack(payload)
```