

# EXPLOIT FARM

Road to “1.0.0” Release



DOMINGO DIRUTIGLIANO

# CONTENTS

## 01

### Introduzione

- Il progetto
- Stato attuale
- Cosa manca

## 02

### Requisiti e analisi

- Documento di specifica dei requisiti
- Analisi dei rischi
- COCOMO Analysis

## 03

### Design

- Frontend
- Xfarm (CLI)
- Backend

# CONTENTS

## 04

### Shared Attacks

- Design architetturale
- Algoritmo di assegnazione dei task
- Algoritmo per il calcolo del timeout

## 05

### Management

- Kanban & Backlog
- Ri-Progettazione DB
- Scheduling GANTT

# INTRODUZIONE

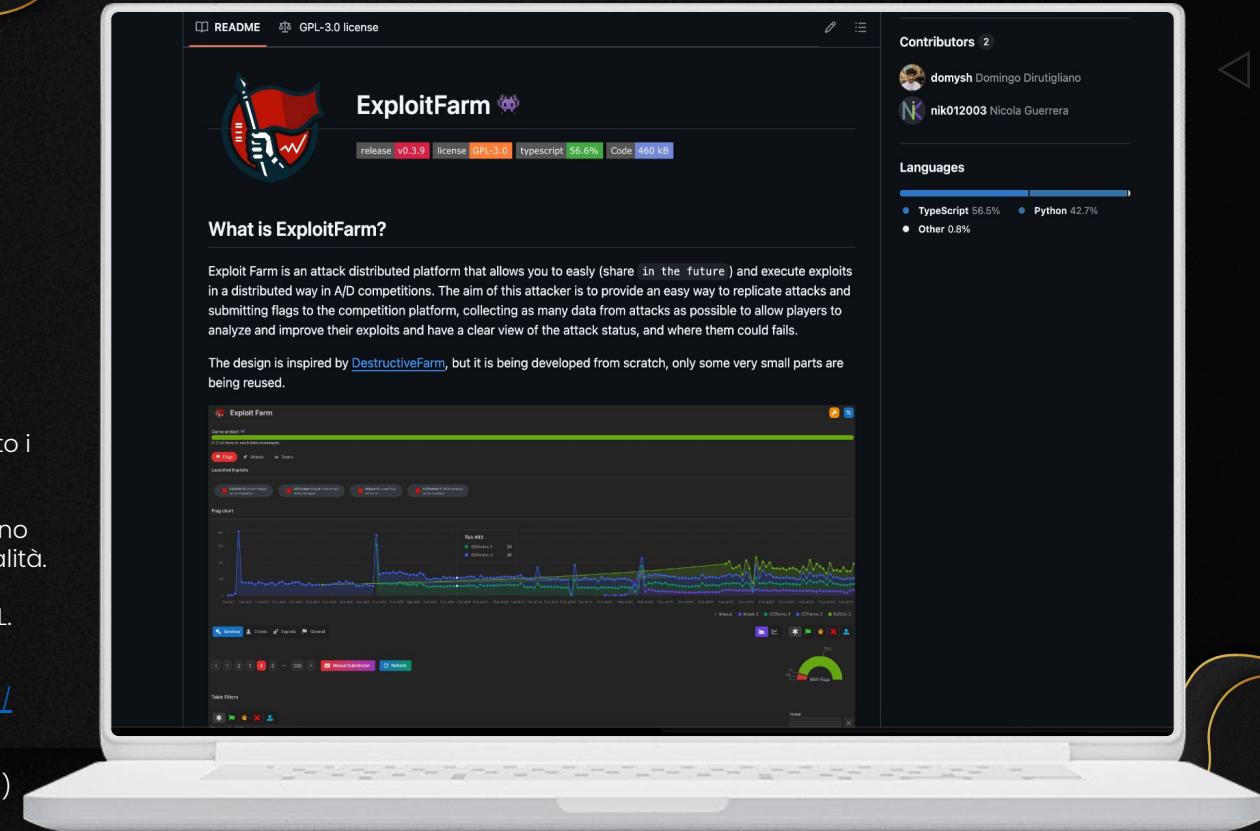
# IL PROGETTO

Exploit farm è una piattaforma per la gestione di attacchi distribuiti e di submission di flag per competizioni CTF Attack Defence. Il suo target sono appunto i team CTF in tutto il mondo, e si propone come drop-replacement di moltissime soluzioni già presenti, che tuttavia peccano di alcune o quasi la totalità delle funzionalità.

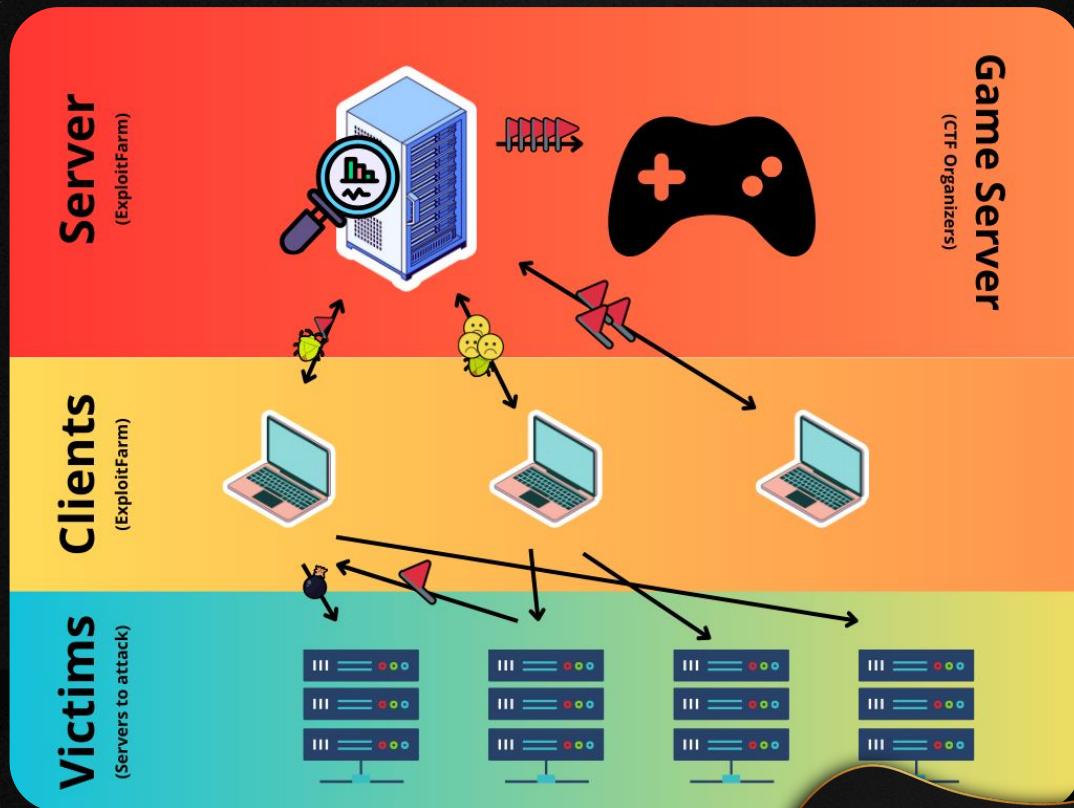
Il progetto è open-source con licenza GPL.

<https://github.com/Pwnzer0ttl/exploitfarm/>

DEMO: <https://exploit.domy.sh>  
(protetta da password)



# STATO ATTUALE



# STATO ATTUALE



## ATTACCHI E SUBMISSION

La gestione dell'avvio degli attacchi parallelo e il sistema che consegna le flag è funzionante, utilizzato anche a CC2024.



## STATISTICHE

Il sistema colleziona informazioni su tempo di attacco, stdout, stderr, client, orario di esecuzione e ovviamente le flag collezionate



## EASY CONFIG

Dispone di interfaccia web, [libreria python](#), con funzionalità semplici per configurare, e modificare anche a runtime le configurazione.



## XFARM (TUI)

Interfaccia Terminal UI per l'avvio, test degli attacchi con configurazione semplice e intuitiva, con visualizzazione dello stato degli attacchi

# COSA MANCA (OBIETTIVI DEL PROGETTO)



## GESTIONE DEI SOURCE CODE DEGLI EXPLOIT + VERSIONING

- Non viene presa traccia dello stato del source code degli exploit
- Non è possibile condividere degli exploit tramite la piattaforma
- Non viene tenuto traccia sull'exploit utilizzato per ogni attacco submittarlo



## GESTIONE DEGLI ATTACCHI CONDIVISI

- Non è possibile distribuire il carico di un singolo attacco su più client
- Manca un sistema di orchestrazione di attacchi condivisi

L'OBIETTIVO DEL PROGETTO È COMPLETARE I REQUISITI E RILASCIARE LA 1.0.0

# REQUISITI E ANALISI

# DOCUMENTO DI SPECIFICA DEI REQUISITI

Il documento raccoglierà le specifiche sull'intero progetto che mancava di una componente simile, con tutte le funzionalità richieste, implementate e da implementare nel sistema con alcune caratteristiche tecniche.

Contenuti principali del documento:

- Descrizione generale
- Descrizione dell'architettura generale della piattaforma
- □ Specifica di alcune fasi di utilizzo del software
- Descrizione delle specifiche generali
- Dettagli tecnici su alcune parti del sistema
- Impostazione dell'organizzazione di progetto
- Scheduling e analisi COCOMO
- Risk management
- △ Analisi SWOT

## Contents

1	Introduzione	3
1.1	Descrizione generale	3
1.2	Obiettivi	3
1.3	Perche lo sviluppo di un nuovo attacker/submitter?	3
2	Specifiche	3
2.1	Composizione del progetto	3
2.1.1	Backend	4
2.1.2	Frontend	4
2.1.3	CLI (xfarm)	4
2.2	Specifiche dei requisiti	5
2.2.1	Setup	5
2.2.2	Dinamicità delle configurazioni	5
2.2.3	Gestione Exploit	5
2.2.4	Esecuzione Attacchi	5
2.2.5	Analisi statistiche e visualizzazione	6
2.2.6	Gestione di attacchi distribuiti	6
2.3	Specifiche Tecniche	6
2.3.1	Databases	6
2.3.2	Gestione funzionalità backend	7
2.3.2.1	API HTTP	7
2.3.2.2	Stats processor	8
2.3.2.3	Submitter Process	8
2.3.3	Funzionalità frontend	8
2.3.4	Strutturazione del client (xfarm)	8
2.3.5	Autobilanciamento del carico sul client per l'esecuzione degli attacchi	9
2.3.6	Gestione e versioning degli exploit	9
2.3.7	Controllo distribuito degli attacchi condivisi	9
3	Project Management	9
3.1	Gestione generale	9
3.2	Kanban (github)	9
3.3	Scheduling	10
3.4	COCOMO Analysis	10
3.4.1	Effort Multipliers	11
3.4.2	Scale Factors	11
3.4.3	Risultato Finale	11
3.5	Risk Management and Analysis	12
3.5.1	Stima dell'Effort e requisiti	12
3.5.2	Rilascio di versioni non totalmente funzionanti	12
3.5.3	Rischio di rendere la piattaforma complessa gestire	12
3.6	Release Management	12
3.7	Modello di Business	12
4	Analisi SWOT	13
4.1	S: Punti di forza	13

# ANALISI DEI RISCHI

## SOTTOSTIMA

di effort e requisiti

Rischio di aver sottostimato eccessivamente il refactoring e lo sviluppo necessario alle nuove funzionalità: rischio medio. Mitigazione: accelerare lo sviluppo per far emergere il prima possibile eventuali problematiche aggiuntive

## COMPLICAZIONE

dell'architettura

Essendoci anche 1 solo sviluppatore e senza avere feedback esterni, c'è la possibilità molto alta che si crei un bias per cui si comincino a sviluppare sistemi difficili da utilizzare.

Mitigazione: chiedere il continuo confronto durante lo sviluppo con i probabili utenti che useranno la piattaforma.

## RELEASE

non funzionanti

Attualmente i test sono manuali e finalizzati al verificare il funzionamento di quella componente che stiamo sviluppando. Questo lascia libero spazio a bug un po' più "lateralì" di crearsi a seguito di release. Soluzione: scrivere test automatici da eseguire prima di ogni release nella release build chain.

# COCOMO ANALYSIS

Il modello COCOMO utilizzato per la stima è quello del Post-architecture model dato che la fase in sviluppo è una fase di implementazione di feature con un progetto di base già pre-ingegnerizzato.

Size stimata = 2k righe di codice

$$E = A \times \text{Size}^B \times \prod_{i=1}^n EM_i$$

$$B = 0.91 + 0.01 \times \sum \text{Scale Factors}$$

$$\prod_{i=1}^n EM_i = 1.431$$

$$B = 0.91 + 0.01 \times \sum \text{Scale Factors} = 0.99$$

$$E = A \times \text{Size}^B \times \prod_{i=1}^n EM_i = 8.36$$

## Moltiplicatori di effort scelti:

Si utilizzerà un subset dei 17 moltiplicatori previsti per COCOMO post-architecture.

- RELY = 1.26 (Very High)
- DATA = 1.00 (Nominal)
- CPLX = 1.17 (High)
- RUSE = 0.95 (Low)
- DOCU = 1.00 (Nominal)
- TIME = 1.29 (Very High)
- PVOL = 1.00 (Nominal)
- PCAP = 0.88 (High)
- TOOL = 0.90 (High)
- SITE = 1.00 (Nominal)
- SCED = 1.00 (Nominal)

## Moltiplicatori di scala scelti:

- PREC = 1.24 (Very High)
- FLEX = 1.01 (Very High)
- RESL = 4.24 (Nominal)
- TEAM = 1.0 (No team, not used)
- PMAT = 1.56 (Very High)
- SCED = 1.00 (Nominal)

## COCOMO ANALYSIS - DURATION

Utilizzando il risultato precedente usiamo la formula di COCOMO per stimare la durata del progetto:

$$C = 3.67$$

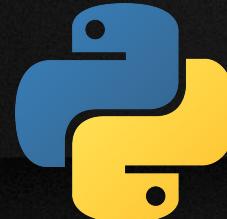
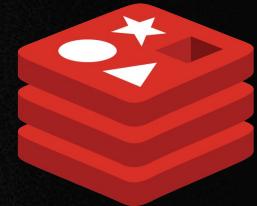
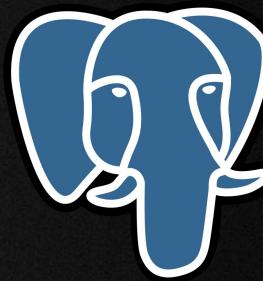
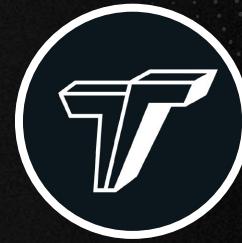
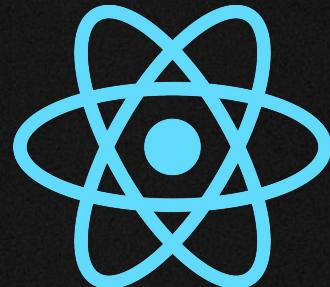
$$D = 0.28 + 0.2 \times (B - 0.91) = 0.296$$

$$T = C \times E^D = 6.8(\text{mesi})$$

Tuttavia assumendo gli obiettivi del progetto, è intuitivo comprendere come la stima eseguita è eccessiva rispetto al reale tempo necessario al raggiungimento dell'obiettivo.

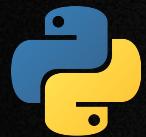
**DESIGN**

# TECHNOLOGIES



FastAPI

SQLA



## TECHNOLOGIES - BACKEND



FastAPI



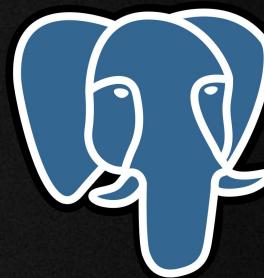
Socket.IO (WebSocket)



SqlAlchemy + SqlModel



Pydantic



Postgres



Redis



# TECHNOLOGIES - BACKEND

The screenshot shows the ExploitFarm API documentation. It includes sections for Status, Clients, and Exploits. Each section lists various HTTP methods (POST, GET, DELETE, PUT) and their corresponding URLs and descriptions. The 'Status' section includes endpoints like /api/setup and /api/status. The 'Clients' section includes endpoints like /api/clients and /api/clients/{client\_id}. The 'Exploits' section includes endpoints like /api/exploits, /api/exploits/{exploit\_id}, and /api/exploits/{exploit\_id}/submit. Each endpoint entry has a lock icon indicating it's secured.

<https://exploit.domy.sh/api/docs>

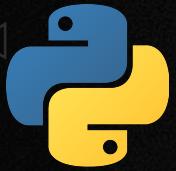
```
@router.post("/{exploit_id}/submit", response_model=MessageResponse[Dict[str, int]])
async def exploit_submit(exploit_id: ExploitID, data: List[ExploitSubmitForm], db: DBSession):
    config = await Configuration.get_from_db()
    stmt = sqla.select(Exploit).where(Exploit.id == exploit_id)
    exploit = (await db.scalars(stmt)).one_or_none()
    if not exploit:
        raise HTTPException(404, "Exploit not found")

    # Check if the exploit was disabled, trigger an update
    trigger_exploit_update = False
    statm = (sqla.select(AttackExecution)
             .where(AttackExecution.exploit_id == exploit_id)
             .order_by(AttackExecution.received_at.desc())
             .limit(1))
    last_attack = (await db.scalars(statm)).one_or_none()
    current_exploit_status = await get_exploit_status(config, last_attack)
    if current_exploit_status == ExploitStatus.disabled:
        trigger_exploit_update = True

async def attack_exec_commit(parsed_data: Dict[str, str]) -> int:
    flags = extract_values_by_regex(config.FLAG_REGEX, parsed_data["flags"])
    del parsed_data["flags"]
    parsed_data["exploit_id"] = exploit.id
    attack_execution = (await db.scalars(
        sqla.insert(AttackExecution)
        .values(parsed_data)
        .returning(AttackExecution)
    )).one()
    db.add(attack_execution)
    if len(flags) > 0:
        flags = (await db.scalars(
            sqla.insert(Flag)
            .values([{"flag": flag, "attack_id": attack_execution.id} for flag in flags])
            .on_conflict_do_nothing(index_elements=[Flag.flag])
            .returning(Flag)
        )).all()

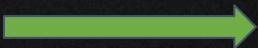
    if len(flags) == 0 and attack_execution.status == AttackExecutionStatus.done:
        attack_execution.status = AttackExecutionStatus.noflags
    return len(flags)

results = await asyncio.gather(*[attack_exec_commit(data) for data in [e]])
await db.commit()
if trigger_exploit_update:
    await redis_conn.publish(redis_channels.exploit, "update")
    await redis_conn.publish(redis_channels.attack_execution, "update")
return { "message": "Attacks results submitted successfully", "response": results }
```



## TECHNOLOGIES - BACKEND BAD STORY :(

<https://github.com/collerek/ormar>

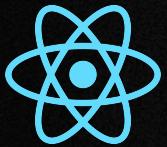


Redis



Socket.IO (WebSocket)

**SQLA** SQLAlchemy + SqlModel



## TECHNOLOGIES - FRONTEND



Mantine UI



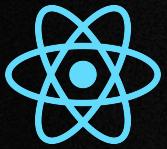
React Query



Socket.IO (Websocket)



Zustand



# TECHNOLOGIES - FRONTEND

**Exploit Farm**

Game ended: 100%  
Click here to see hidden messages

Flags Attacks Teams

Launched Exploits

- ExCCel-2 (token-forge) ran by Alessandro
- CCForms-2 (path/traversal) ran by Giuseppe
- Inlook-2 (overflow) ran by Giuseppe
- CCForms-1 (AES-attack) ran by Giuseppe

Flag chart

Legend: Manual, ExCCel-2, Inlook-2, CCForms-1, CCForms-2, CCForms-3

Services Clients Exploits General

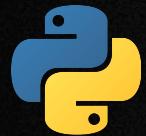
1 2 3 4 5 ... 290 Manual Submission Refresh

7933 681 Flags

Table Filters

Total results: 8691

ID	FLAG	SERVICE	TEAM	EXECUTION	RESPONSE	STATUS
8932	GZC2G60AKLSONOB894NXXEGZVUKL3937=	ExCCel-2 using token-forge exploit	Università degli Studi di Salerno	time: 3 s by Alessandro	Accepted: 7.606737901101299 flag points Submitted At: 04/07/2024 17:58:19	GREEN
8931	Z3HMVA3RQEfi0U2D0189ZJ30AV9C379=	ExCCel-2 using token-forge exploit	Università degli Studi di Verona	time: 3 s by Alessandro	Accepted: 5.808579410609684 flag points Submitted At: 04/07/2024 17:58:19	GREEN
8930	U479URP9V9392M4A1I4V4T4X2A3	ExCCel-2	Università degli Studi di Verona	time: 3 s by Alessandro	Accepted: 5.852136174835577 flag points Submitted At: 04/07/2024 17:58:19	GREEN



## TECHNOLOGIES - XFARM (CLI)



**Textual**



**Socket.IO (Websocket)**

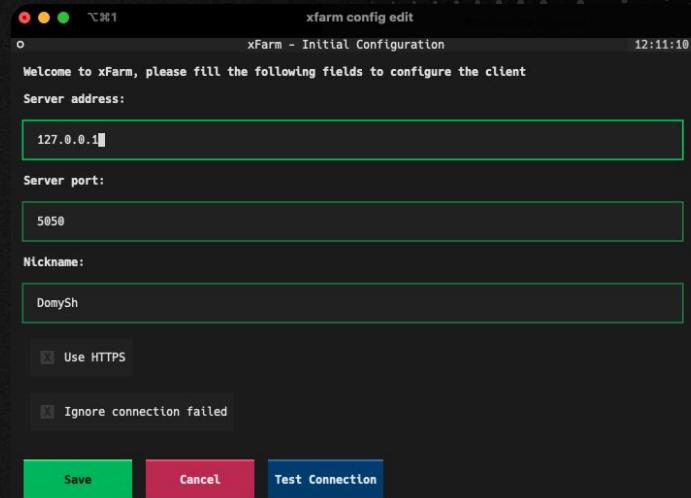


# TECHNOLOGIES - XFARM (CLI)

```
xploit xploit_test is running.. Submitter status: (queued: 0) Exploit timeout: 118 s Tick Duration: 120 s Next Attacks: 0:01:44 s System CPU: 26.0% Flag Format: [a-zA-Z0-9]{32} Running worker: 0 (max: 80) System Memory: 23.9% Teams Log
```

Team	Host	Flags	Last status	Last attack	Time to exploit	Executing
1: Fake team 1	127.0.0.1	60	✓ ➔	0:00:11 ago	0:00:03	➡
2: Fake team 2	127.0.0.2	56	✓ ➔	0:00:11 ago	0:00:03	➡
3: Fake team 3	127.0.0.3	61	✓ ➔	0:00:11 ago	0:00:03	➡
4: Fake team 4	127.0.0.4	64	✓ ➔	0:00:11 ago	0:00:04	➡
5: Fake team 5	127.0.0.5	68	✓ ➔	0:00:11 ago	0:00:03	➡
6: Fake team 6	127.0.0.6	59	✓ ➔	0:00:12 ago	0:00:02	➡
7: Fake team 7	127.0.0.7	59	✓ ➔	0:00:11 ago	0:00:03	➡
8: Fake team 8	127.0.0.8	54	✓ ➔	0:00:11 ago	0:00:03	➡
9: Fake team 9	127.0.0.9	61	✓ ➔	0:00:11 ago	0:00:03	➡
10: Fake team 10	127.0.0.10	70	✓ ➔	0:00:11 ago	0:00:04	➡
11: Fake team 11	127.0.0.11	56	✓ ➔	0:00:11 ago	0:00:04	➡
12: Fake team 12	127.0.0.12	52	✓ ➔	0:00:10 ago	0:00:04	➡
13: Fake team 13	127.0.0.13	63	✓ ➔	0:00:11 ago	0:00:04	➡
14: Fake team 14	127.0.0.14	54	✓ ➔	0:00:11 ago	0:00:04	➡
15: Fake team 15	127.0.0.15	56	✓ ➔	0:00:11 ago	0:00:04	➡
16: Fake team 16	127.0.0.16	64	✓ ➔	0:00:10 ago	0:00:04	➡
17: Fake team 17	127.0.0.17	58	✓ ➔	0:00:10 ago	0:00:04	➡
18: Fake team 18	127.0.0.18	62	✓ ➔	0:00:10 ago	0:00:04	➡

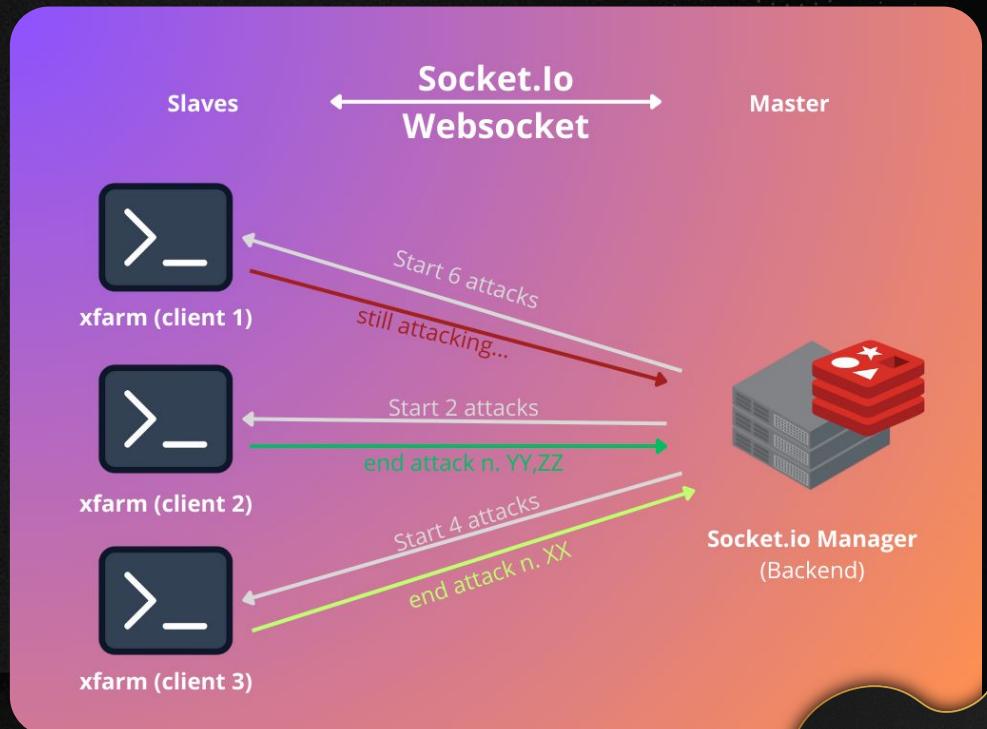
\*c Close attack 1 Show teams 2 Show logs



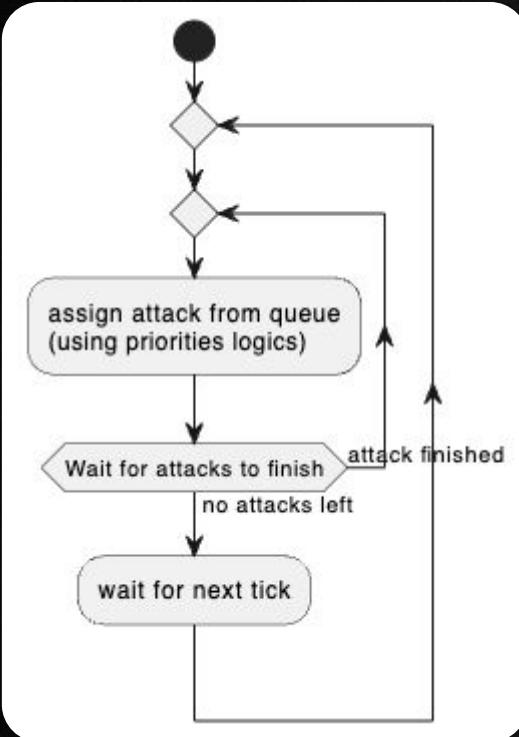
```
~c Cancel configuration ~t Test connection to server ~s Save configuration  
Development mode detected!  
~/repos/exploitfarm/tests/xploit_test (main*) » xfarm exploit push  
Development mode detected!  
Waiting for a server response... ━━  
Exploit source uploaded!  
~/repos/exploitfarm/tests/xploit_test (main*) » xfarm exploit info  
Development mode detected!  
Exploit Source Log of xploit_test (f396776e-dd32-4712-b61b-5c6cb90d5a01)  
Current Commit  
» Commit 02b2350c-6f23-48e0-bf99-6bf9b5f47a by DomySh  
at 2024-11-07 12:12:08 hash:  
2dbb5ba27e084d8fcc90811a8896e7024fdf3ac07adb6665be9def67374af79b  
Message: No message  
~/repos/exploitfarm/tests/xploit_test (main*) »
```

# SHARED ATTACKS

# DESIGN DELL'ARCHITETTURA



# ALGORITMO DI ASSEGNAZIONE DEI TASK



## Definizioni

- $N_C$ : Numero di client nel gruppo
- $N_T$ : Numero di team totali

Individueremo in seguito  $i = [0, N_C - 1]$  l'indice relativo al client  
Inoltre definiamo  $j = [0, N_T - 1]$  l'indice relativo ai team, e quindi all'attacco eseguito nel tick

- $Q_i$ : grandezza della coda di esecuzione per l' i-esimo client. (definita esplicitamente in seguito)
- $R_i$  attacchi in esecuzione sul client (sempre  $\leq Q_i$ )
- $T_R$  Tick/Round time (Tempo definito nel setup relativo alla durata di un round)
- $T_K$  Timeout associato agli attacchi (Kill time)
- $T_{C,j}$  Tempo effettivamente consumato per il j-esimo attacco (per il j-esimo team)

## Calcolo della Priorità

$$P_{CL,i} = \frac{Q_i - R_i}{\min\{Q_i\}}$$

Regole di assegnazione:

Se  $P_{CL,i} \geq 1$  ed associa ad ognuno un numero di attacchi pari a  $\lfloor P_{CL,i} \rfloor$

Se per ogni i risulta  $P_{CL,i} > 0$  assegnamo ciclicamente un processo al client per cui risulta:  $P_{CL,i} = \max\{P_{CL_i}\}$ .

Altrimenti il processo di assegnazione termina

# ALGORITMO DI CALCOLO DEL TIMEOUT

## Definizioni

- $T_{J,i}$ : Tempo di join dell' i-esimo client: Questo tempo vale 0 se il client ha eseguito il join prima dell'inizio del tick, mentre assume un valore che corrisponde al tempo mancante al termine del tick se riguarda un client che ha eseguito il join durante questo tick.
- $T_{L,i}$ : Tempo di left dell' i-esimo client: Questo tempo vale il tempo rimanente lasciato non in esecuzione dalla fine del tick del j-client che esce dal gruppo. Se il j-esimo client non è uscito dal gruppo questo tick questo valore vale 0
- $T_{LJ,i} = T_{J,i} - T_{L,i}$ : Tempo di left/join per client (definisce il guadagno totale dato dai left e join di un singolo client).
- $T_{LJ} = \sum_i T_{LJ,i} Q_i$ : Tempo totale di left/join (definisce in totale il tempo guadagnato/perso a causa delle operazioni di join e left dei client)
- $T_{G,j} = T_R - T_{C,j}$ : Tempo di guadagno per attacco
- $T_G = \sum_j T_{G,j}$ : Tempo di guadagno totale
- $T_{VB} = \sum_i Q_i T_R$ : (tempo virtualmente disponibile di base) interpretabile come il tempo totale disponibile di esecuzione su un core singolo per tick, calcolato all'inizio del tick.
- $T_V = T_{VB} + T_G + T_{LJ}$ : Tempo virtualmente disponibile

**Calcolo del timeout:**  $T_K = \frac{T_V}{N_T}$

# MANAGEMENT

# KANBAN & BACKLOG

Il progetto è interamente condiviso e gestito su github, grazie alla funzionalità dei progetti di cui si sfrutta il backlog che permette di gestire le user-stories con tag, ordini di priorità e direttamente associabili ai branch in cui si esegue l'attività di sviluppo, per questo molto integrato con lo sviluppo stesso.

Software Engineering - Road to ExploitFarm 1.0.0

Backlog | Team capacity | Current iteration | Roadmap | My items | New view

Filter by keyword or by field

**Todo** 4 / 100 Estimate: 0

- This item hasn't been started
- exploitfarm #13: Remote downloaded exploit start (enhancement, Exploit Execution Sharing)
- exploitfarm #6: Remote Uploaded exploit start (enhancement, Exploit Execution Sharing)
- exploitfarm #10: Shared attack test (testing)
- exploitfarm #12: README edits (documentation)

**In Progress** 3 / 100 Estimate: 0

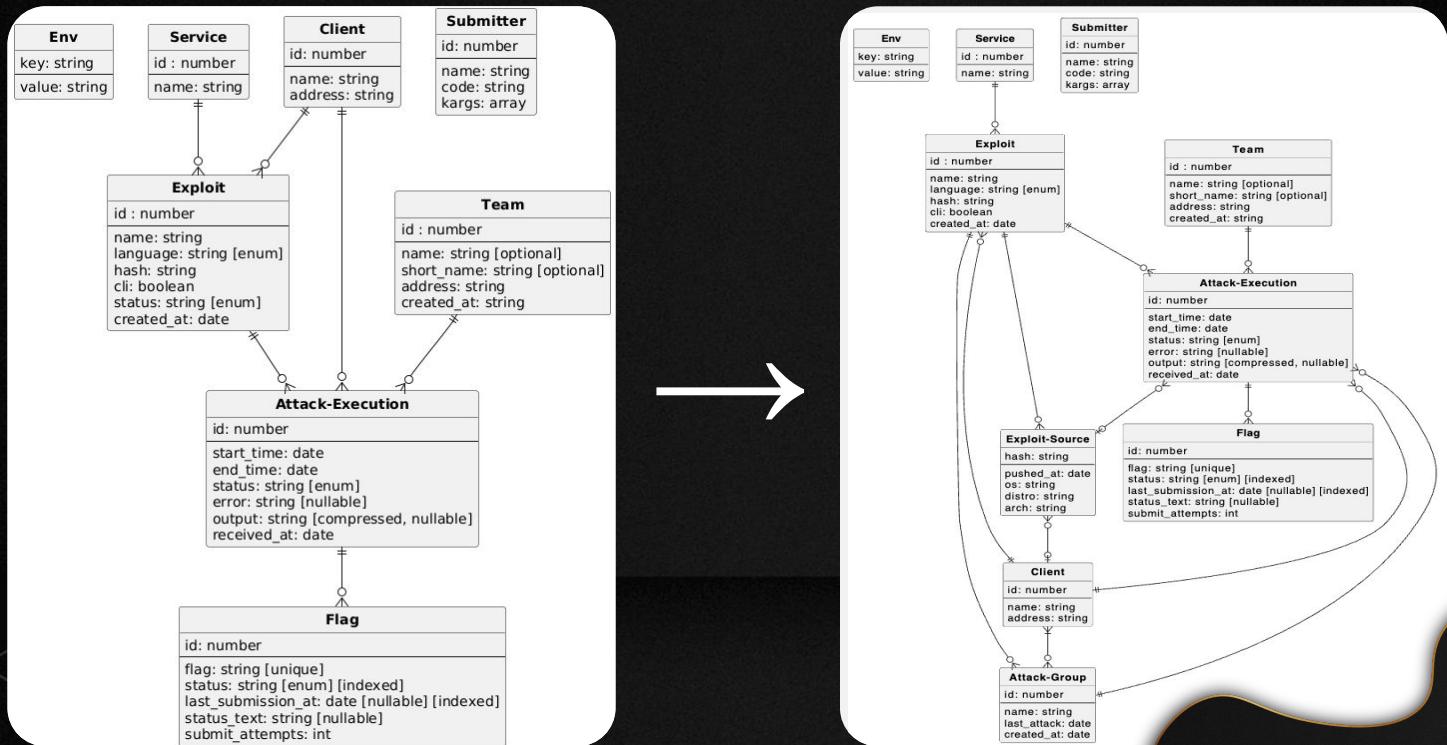
- This is actively being worked on
- exploitfarm #22: Exploit source reference on Attack-Execution (enhancement)
- exploitfarm #8: Group creation (enhancement, Exploit Execution Sharing)
- exploitfarm #9: Group join and start (enhancement, Exploit Execution Sharing)

**Done** 10 / 100 Estimate: 0

- This has been completed
- exploitfarm #11: Business Canvas (documentation)
- exploitfarm #5: Specifics Document with technical details (documentation)
- exploitfarm #23: Implementet export/import db data (enhancement)
- exploitfarm #21: Redis implementation: event driven stats and submitter processes + socket.io init (enhancement)

# RI-PROGETTAZIONE DB

Durante la fase di progettazione delle nuove funzionalità, si è prima fatto brainstorming su idee tecniche di implementazione, e successivamente si è subito provveduto a ristrutturare il database già esistente



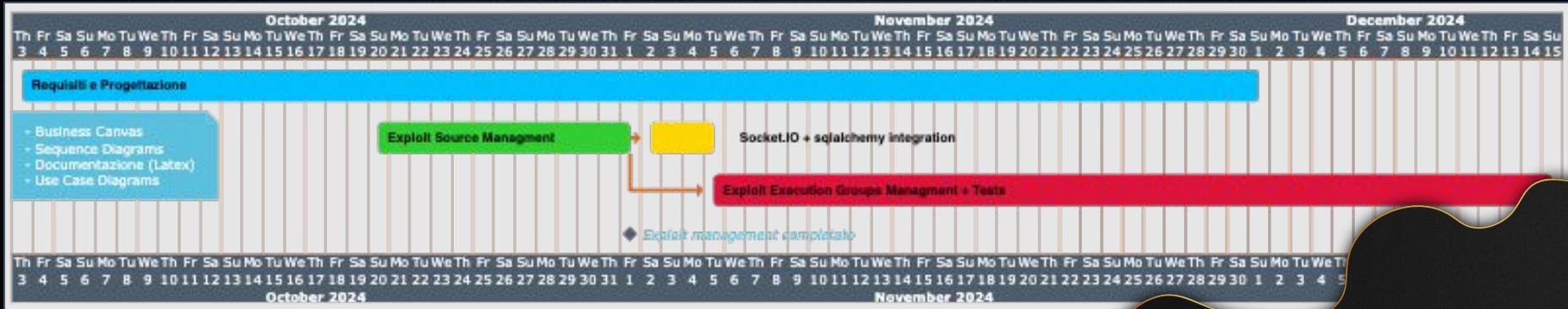
# SCHEDULING

Sono definite 3 fasi principali nel progetto:

- Brainstorming, definizione dei requisiti tecnici e documentazione
- 1° Milestone/Sprint: gestione degli exploit
- 2° Milestone/Sprint: gestione dei gruppi di client per gli attacchi condivisi

Le fasi sono definite proprio per la necessità di progettare lo sviluppo stesso dei nuovi requisiti, e dalla mancanza della feature di gestione degli exploit necessaria allo sviluppo della seconda milestone.

- L'inizio del progetto con la prima fase è iniziata il 03-10-2024, la consegna è fissata per il 20-12-2024.





**FINE !**