Valutazione di Flawfinder su 3 progetti popolari su GitHub: Indagine sulle vulnerabilità in codice C

Approfondimento per il corso di Sicurezza Informatica

6 luglio 2023

Rocco Gianni Rapisarda, 845197

1 Introduzione

In questa relazione viene mostrata un'analisi dettagliata sull'utilizzo di Flawfinder[1], uno strumento di analisi statica ampiamente adottato nel campo della sicurezza informatica, per valutare la presenza potenziale di vulnerabilità in tre progetti scritti in linguaggio C, disponibili su GitHub e caratterizzati da un significativo numero di stelle (stars).
Flawfinder è un tool che si concentra sull'individuazione di difetti di programmazione nel codice sorgente C e C++, al fine di rilevare vulnerabilità di sicurezza note. Questo strumento si basa su un database di pattern di codice che corrispondono a vulnerabilità comuni, come ad esempio buffer overflow, race condition e mancanza di controlli sugli input.
L'accuratezza di Flawfinder non è infallibile e possono verificarsi situazioni in cui vengono segnalati potenziali problemi di sicurezza che in realtà non sono presenti nel codice. Questi falsi positivi possono essere il risultato di complessità nell'analisi statica, dell'incapacità di riconoscere determinati modelli di codice o di altre limitazioni dell'algoritmo utilizzato.
Nel corso di questa relazione, saranno presentati i risultati ottenuti dall'applicazione di Flawfinder sui tre progetti prescelti.
La lista delle CWE che Flawfinder rileva è disponibile a pagina 10 della documentazione del tool e di seguito ne viene riportata una sotto lista.

- CWE-20: validazione impropria dell'input.
- CWE-120: copia del buffer senza verifica della dimensione dell'input.
- CWE-126: Buffer Over-read.
- CWE-250: esecuzione senza privilegi.

Nonostante la presenza di falsi positivi, l'utilizzo di Flawfinder rimane uno strumento valido per avviare un processo di analisi delle vulnerabilità di sicurezza nel codice sorgente scritto in C e per una valutazione completa e accurata può essere necessario eseguire dell'analisi statica con altri strumenti oppure dell'analisi dinamica.

[1] https://dwheeler.com/flawfinder/

2 Metodologia d'analisi

Per condurre l'analisi statica dei progetti presi in considerazione, è stato sviluppato uno script in *Bash* che automatizza il processo di recupero delle repository GitHub dei progetti, l'esecuzione di Flawfinder e la generazione dei grafici. L'intero flusso di lavoro è stato strutturato come segue:

1. **Creazione delle cartelle di output:** lo script verifica se le cartelle "projects" e "output" esistono. In caso contrario, le crea per consentire l'organizzazione dei file generati durante l'analisi.

2. **Recupero delle repository GitHub:** una volta verificata la presenza delle cartelle, lo script procede al recupero delle repository dei progetti specificati. Le repository vengono clonate all'interno della cartella "projects" per consentire un'analisi locale.

3. **Esecuzione di Flawfinder:** dopo aver clonato le repository, lo script esegue Flawfinder sui file sorgentedi ciascun progetto considerando le vulnerabilità con un livello pari o superiore a 4 (su una scala da 0 a 5), al fine di focalizzare l'analisi sulle potenziali vulnerabilità più critiche.

4. **Salvataggio dei risultati in formato CSV:** i risultati ottenuti dall'analisi di Flawfinder vengono salvati in un file CSV all'interno della cartella "output". Questo file contiene informazioni dettagliate sulle vulnerabilità individuate, inclusi i nomi dei file, i livelli di rischio associati e i tipi di vulnerabilità corrispondenti.

Successivamente, per la generazione dei grafici, sono stati sviluppati due script in Python che elaborano i dati presenti nei file CSV e crea due tipi di grafici:

- **Grafico a barre:** vengono utilizzati i valori della colonna "CWEs" (Common Weakness Enumerations) del file CSV, che rappresentano i tipi di vulnerabilità identificate da Flawfinder. Utilizzando tali valori, viene generato un grafico a barre che mostra la distribuzione delle vulnerabilità per ogni progetto analizzato.

- **Grafico a torta:** i dati presenti nel file CSV vengono analizzati per generare un grafico a torta che visualizza la percentuale di ogni categoria di vulnerabilità rispetto al totale delle vulnerabilità rilevate. Le vulnerabilità con una percentuale inferiore al 3% vengono raggruppate nella categoria *Others*.

Il codice per quanto descritto è disponibile nel repository GitHub[2].

[2]https://github.com/roccograpisarda/flawfinder-github-best-projects.

3 Risultati

I risultati dell'analisi condotta utilizzando Flawfinder sui progetti presi in esame presentano un'importante fonte di informazioni riguardo alle vulnerabilità potenziali rilevate nel codice sorgente scritto in linguaggio C.
La valutazione è stata effettuata considerando i livelli di rischio pari o superiori a 4, come definito all'interno dello strumento Flawfinder.

Progetto	**Numero di ⋆**	**Commit hash**
OBS-Studio	47,800	adb702929
Scrcpy	84,000	958f2249
VLC	10,900	f7bb59d9f5

Tabella 1: Informazioni relative alla repository GitHub. Statistics

3.1 OBS-Studio

OBS[3] è un software open-source per la registrazione ed è ampiamente utilizzato per lo streaming su piattaforme come Twitch e YouTube, offrendo funzionalità avanzate come la cattura di schermo, l'aggiunta di overlay e la gestione di sorgenti video e audio.
Nel progetto OBS-Studio, come è possibile notare dalla figura 1, vi è una predominanza della vulnerabilità *CWE-134*[4](48.9%), seguita dalla *CWE-120*[5] (33.3%).
La prima vulnerabilità si verifica se è utilizzata una funzione come `printf` oppure `snprintf` che accetta una stringa di formato come argomento, ma questa proviene da un'origine esterna. Un utente malintenzionato può modificare una stringa di formato controllata dall'esterno e può causare Buffer overflow oppure DoS oppure problemi relativi alla rappresentazione dei dati.
In alcune circostanze, come l'internazionalizzazione, l'insieme di stringhe di formato è controllato esternamente dalla progettazione.

```
1                int main(int argc, char **argv){
2                      char buf[128];
3                      snprintf(buf,128,argv[1]);
4                }
```

Codice 1: Esempio di codice in cui è presente un'istanza di CWE-134.

La *CWE-120*, invece è presente nei programmi in cui viene copiato un buffer di input in un buffer di output senza verificare che la dimensione del buffer di input sia inferiore alla dimensione del buffer di output e questo porta a un possibile Buffer overflow.

[3]https://github.com/obsproject/obs-studio
[4]Use of Externally-Controlled Format String.
[5]Buffer Copy without Checking Size of Input ('Classic Buffer Overflow').

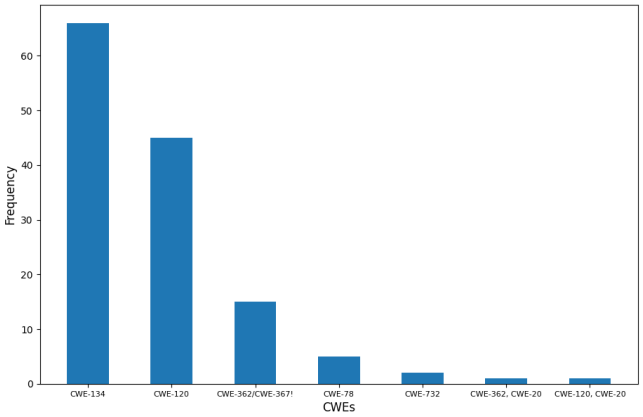

Figura 1: Istogramma delle CWE individuate in OBS-Studio tramite Flawfinder.

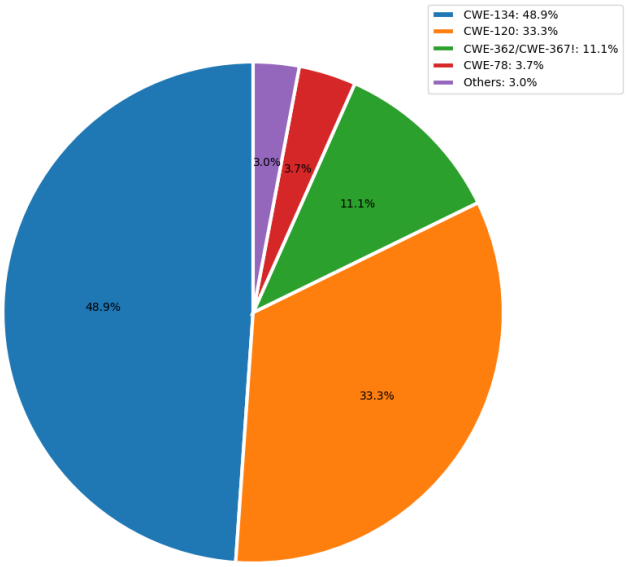

Figura 2: Grafico a torta contenente la percentuale di presenza per ciascuna CWE individuata in OBS-Studio tramite Flawfinder.

3.2 Scrcpy

Scrcpy[6] è uno strumento open-source che consente di visualizzare e controllare utilizzando il mouse e la tastiera del computer un dispositivo Android tramite una connessione USB o una connessione Wi-Fi.

[6]https://github.com/Genymobile/scrcpy

Rispetto agli altri progetti che sono stati analizzati, Scrcpy contiene meno vulnerabilità e la più presente è la *CWE-78*[7], che permette ad un attaccante di eseguire comandi imprevisti e pericolosi direttamente sul sistema operativo. Inoltre se il programma viene eseguito con i privilegi d'amministratore allora l'attaccante può eseguire dei comandi con privilegi che non possiede.

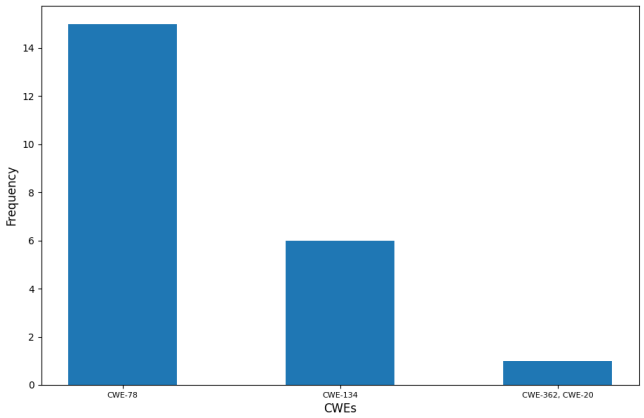

Figura 3: Istogramma delle CWE individuate in Scrcpy tramite Flawfinder.

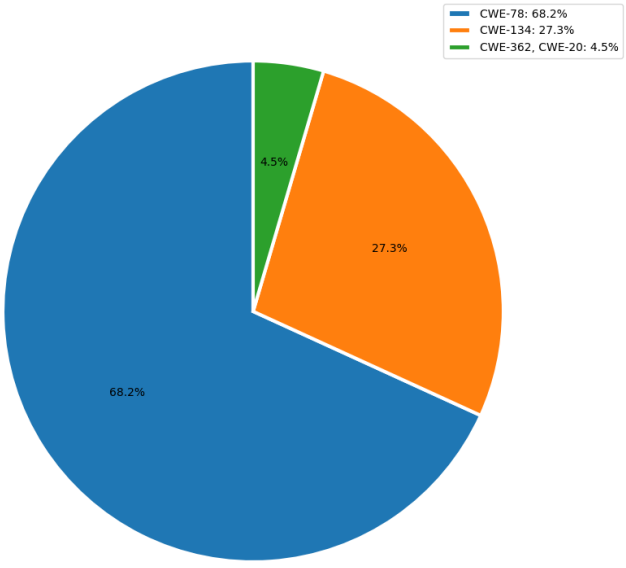

Figura 4: Grafico a torta contenente la percentuale di presenza per ciascuna CWE individuata in Scrcpy tramite Flawfinder.

[7]Improper Neutralization of Special Elements used in an OS Command.

3.3 VLC

Nel progetto VLC[8], come è possibile notare dalla figura 5, vi è una predominanza della vulnerabilità *CWE-362/CWE-367*[9](53.9%).
La vulnerabilità CWE-362/CWE-367 è conosciuta come "Race Condition" e si verifica quando il comportamento di un programma dipende dall'ordine in cui vengono eseguite diverse operazioni concorrenti o interagenti. Nel contesto del progetto VLC questa vulnerabilità può emergere se un aggressore riesce a modificare qualcosa lungo il percorso tra la chiamata ad access() e l'utilizzo effettivo del file, ad esempio spostando i file. Durante questo intervallo di tempo può modificare il percorso del file o i suoi permessi sfruttando così la race condition per ottenere un accesso non autorizzato al file.
Una possibile soluzione può essere quella di aprire direttamente il file dopo la chiamata ad access(), senza lasciare un intervallo di tempo in cui la race condition possa essere sfruttata. Altre due vulnerabilità rilevate da Flawfinder sono la CWE-120, la CWE-134 e la CWE-327.
Le istanze di CWE-327[10] sono segnalate nel codice del file *file.c*, in cui sono presenti direttive di compilazione condizionale (#ifdef CRYPTFILE) che suggeriscono la presenza di funzionalità di crittografia. Tuttavia, gli algoritmi di crittografia effettivi e le loro implementazioni non sono visibili nel frammento di codice.

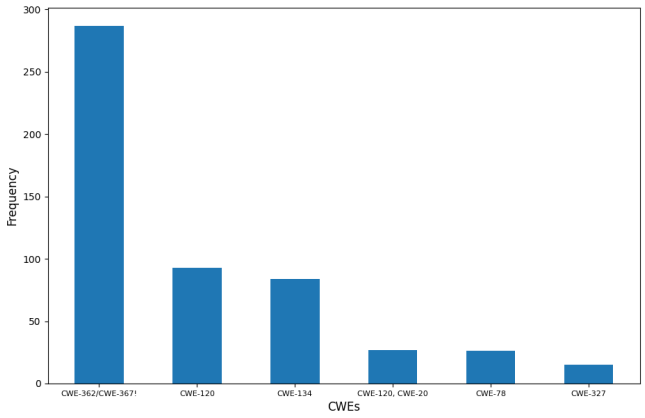

Figura 5: Istogramma delle CWE individuate in VLC tramite Flawfinder.

[8] https://github.com/videolan/vlc
[9] Concurrent Execution using Shared Resource with Improper Synchronization.
[10] Use of a Broken or Risky Cryptographic Algorithm

4 Conclusioni

Dall'analisi effettuata è emerso che i tre progetti sono affetti da varie tipologie di vulnerabilità ed in alcuni casi si riscontrano le stesse vulnerabilità in tutti e tre i progetti, come ad esempio la *CWE-134* e la *CWE-78*. Questa informazione è rappresentata in modo sintetico nel grafico riassuntivo mostrato nella figura 6.

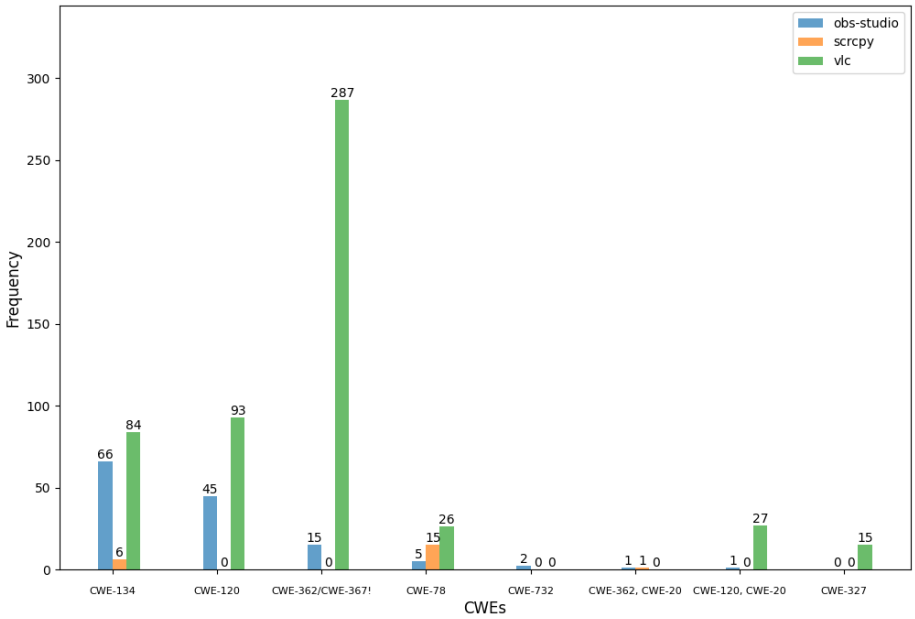

Figura 6: Istogramma delle CWE individuate in tutti i progetti tramite Flawfinder.

Come detto nell'introduzione Flawfinder esegue un'analisi statica del codice, il che significa che potrebbe produrre falsi positivi. Per gestire correttamente i falsi positivi, è consigliabile eseguire una verifica manuale delle linee di codice segnalate da Flawfinder e nel caso in cui non si trattasse di un'istruzione pericolosa indicare al tool, utilizzando uno dei due commenti presenti nel listato 2 nel codice, di ignorare le linee che non rappresentano una reale minaccia per la sicurezza. In questo modo, si ottiene un'analisi più accurata e si riduce il rischio di segnalazioni errate.

```
1 // Flawfinder: ignore
2 /* Flawfinder: ignore */
```

Codice 2: Commento per ignorare falso positivo.