Buffer overflow

Un **buffer overflow** si verifica quando un programma scrive più dati di quanti ne possa contenere un buffer, ovvero una porzione di memoria allocata per memorizzare i dati. Questo comportamento può sovrascrivere altre aree della memoria, portando a comportamenti imprevisti, crash del programma o, nel peggiore dei casi, all'esecuzione di codice arbitrario.

Ci sono diversi tipi di buffer overflow:

1. **Stack Overflow**: Si verifica nello stack di esecuzione, dove le variabili locali e gli indirizzi di ritorno delle funzioni vengono memorizzati.
2. **Heap Overflow**: Si verifica nell'heap, dove vengono allocati dinamicamente gli oggetti. Può sovrascrivere strutture di dati, compromettendo il flusso di esecuzione.
3. **Global Overflow**: Avviene in variabili globali, con conseguenze simili agli altri tipi.

Come Funziona?

1. **Allocazione della Memoria**: Quando un programma viene eseguito, richiede spazio in memoria per memorizzare variabili, dati e strutture.
2. **Scrittura e Lettura**: Se il programma non gestisce correttamente i limiti della memoria e scrive più dati del previsto in un buffer, i dati in eccesso possono sovrascrivere altre aree di memoria.
3. **Esecuzione di Codice Malevolo**: Un attaccante può sfruttare questa vulnerabilità per iniettare codice malevolo. Ad esempio, se riesce a sovrascrivere l'indirizzo di ritorno di una funzione, può dirottare il flusso di esecuzione del programma verso il codice dannoso.

Attacchi di buffer:

1. **Shellcode Injection:** Inserimento di codice eseguibile (shellcode) nel buffer. Una volta sovrascritto l'indirizzo di ritorno, il programma esegue il shellcode.
2. **Return-to-libc Attack:** Invece di iniettare codice, l'attaccante modifica l'indirizzo di ritorno per saltare a una funzione di libreria esistente (es. system()).
3. **Format String Exploit:** Utilizza vulnerabilità nelle funzioni di formattazione (es. printf) per sovrascrivere la memoria.

Contesto di Applicazione

I buffer overflow sono particolarmente rilevanti nei linguaggi di programmazione come C e C++, che non gestiscono automaticamente la memoria. Molti attacchi informatici sfruttano questa vulnerabilità:

- **Malware**: Gli attaccanti possono creare malware che utilizza buffer overflow per ottenere accesso non autorizzato ai sistemi.

- **Exploits**: Le tecniche di exploit possono variare, ma spesso coinvolgono la scrittura di payload dannosi che vengono eseguiti quando il programma viene compromesso.

Mitigazione

Quasi tutte le applicazioni ed i server web sono vulnerabili agli attacchi buffer overflow.
Gli ambienti scritti in linguaggi come Java e Python sono gli unici potenzialmente immuni agli attacchi, ad eccezione degli overflow nel loro interprete.

Gli sviluppatori di applicazioni possono prevenire la minaccia inserendo misure di sicurezza nel loro codice di sviluppo, nonché utilizzare linguaggi di programmazione che includono protezione integrata e test per rilevare e correggere errori. Tra le principali raccomandazioni per prevenire il sovraccarico del buffer si annoverano:

- evitare funzioni di libreria standard che non sono state controllate
- assicurarsi di rispettare i limiti che vengono applicati in fase di esecuzione.

Ciò verifica automaticamente che i dati scritti su un buffer siano entro i confini appropriati. In sintesi, tra le tecniche più efficaci per contrastare un attacco di tipo buffer oveflow troviamo:

Address Space Layout Randomization (casualizzazione dello spazio degli indirizzi)

L'ASLR consiste nel rendere casuale l'indirizzo delle funzioni di libreria e le più importanti aree di memoria. Così un attacco informatico che cerca di eseguire codice malevolo sul computer è costretto a cercare gli indirizzi del codice e dei dati che gli servono prima di poterli usare, provocando una serie di crash del programma dannoso.

Prevenzione dell'esecuzione dei dati

Questo metodo impedisce ad un attacco di poter eseguire codice in regioni non eseguibili distinguendo le aree di memoria come eseguibili o non eseguibili.

Structured Exception Handler Overwrite Protection (Gestione eccezioni strutturate)

Tutti i programmi implementano delle procedure per la gestione delle eccezioni, cioè quelle situazioni in cui si verificano particolari condizioni od eventi che alterano la normale esecuzione delle istruzioni. La funzionalità SEHOP, si occupa di esaminare in tempo reale quando viene generata un'eccezione in un qualunque programma in esecuzione. A questo punto, SEHOP provvede a verificare che la catena SEH non sia stata in alcun modo modificata.

Data Execution Prevention o DEP (prevenzione dell'esecuzione dei dati)

Grazie a questa tecnica, l'attaccante non può eseguire il codice se si trova nello spazio di memoria assegnato allo stack o all'heap e, in alcuni casi, anche in altre aree. L'implementazione di misure di sicurezza sul codice di sviluppo e sui sistemi operativi, però, non è garanzia di successo. Quando viene scoperta una nuova vulnerabilità di buffer overflow, è fondamentale patchare rapidamente il software e assicurarsi che l'aggiornamento sia reso disponibile a tutti gli utenti.

Analizziamo brevemente quanto fin qui illustrato in materia di buffer overflow:

- Il buffer overflow si verifica quando ci sono in un buffer più dati di quanti se ne possano gestire, causando lo spostamento degli stessi nell'archivio adiacente
- questa vulnerabilità può causare un arresto anomalo del sistema o, peggio, creare un punto di ingresso per un attacco informatico
- i linguaggi di programmazione C e C ++ sono più vulnerabili a questo tipo di errore

- le pratiche di sviluppo di app sicure dovrebbero includere test regolari per rilevare e correggere eventuali falle. Queste best practice dovrebbero includere: la protezione automatica a livello di codice e il controllo dei limiti in fase di esecuzione