

Memory safety: la classe di vulnerabilità che i governi vogliono estinguere



La memory safety è uno dei problemi di sicurezza più antichi del software, e nel 2026 è tornata in cima all'agenda con la spinta delle agenzie governative. Si tratta dell'assenza di una intera famiglia di difetti, quelli che nascono quando un programma gestisce a mano la memoria e sbaglia: un *buffer overflow* che scrive oltre i confini di un'area, un *use-after-free* che accede a memoria già liberata, una lettura fuori dai limiti. Non sono bug esotici, sono i mattoni con cui si costruiscono da decenni gli *exploit* più gravi, e continuano a esserlo. Secondo la guida di CISA e agenzie alleate del dicembre 2023, due terzi delle vulnerabilità segnalate nei linguaggi non *memory-safe* restano legate proprio alla gestione della memoria.

Il punto di svolta non è tecnico ma di prospettiva. Per anni questi difetti sono stati trattati come errori da correggere uno per uno; oggi l'idea che si sta affermando è che siano una classe da eliminare alla radice, scegliendo strumenti che non li rendano possibili. È lo spostamento dal "programmare con più attenzione" al "programmare in un linguaggio che non lascia commettere quell'errore", e ha smesso di essere una preferenza accademica per diventare una richiesta esplicita di chi regola il mercato.

Che cos'è la memory safety, e perché C e C++ non ce l'hanno

Un linguaggio è *memory-safe* quando impedisce, per costruzione, gli accessi scorretti alla memoria: non si può leggere oltre

la fine di un array, usare un puntatore a memoria già liberata o dimenticare di controllare un limite, perché il linguaggio non lo consente o lo verifica al posto dello sviluppatore. Rientrano in questa categoria linguaggi molto diversi tra loro, da Java e C# a Go, Swift, Python e Rust, che ci arrivano per strade differenti: alcuni con un *garbage collector* che gestisce la memoria automaticamente, Rust con un sistema di proprietà verificato dal compilatore che ottiene lo stesso risultato senza rinunciare al controllo di basso livello.

C e C++, i linguaggi su cui poggia gran parte del software di sistema, non offrono questa garanzia: lasciano allo sviluppatore la gestione manuale della memoria, e con essa la possibilità di sbagliare. La conseguenza è quantificata da chi ha più codice al mondo. Microsoft ha stimato nel 2019 che circa il 70% delle proprie vulnerabilità con CVE derivasse ogni anno da problemi di memory safety, e in Chrome, su un'analisi di 912 bug di gravità alta o critica segnalati dal 2015 in poi, Google ne ha ricondotto alla stessa classe circa il 70%. Non è una questione di programmatori distratti: è che una classe di errori, su basi di codice enormi, si ripresenta comunque, a prescindere dalla bravura di chi scrive.

Da problema tecnico a richiesta delle autorità

La novità è che la memory safety è entrata nei documenti delle autorità. La stessa guida di CISA, NSA, FBI e delle agenzie di Australia, Canada, Regno Unito e Nuova Zelanda chiede ai produttori di software di pubblicare una *roadmap* memory-safe, cioè un piano concreto per ridurre nel tempo la dipendenza dai linguaggi non sicuri. Con il documento Product Security Bad Practices, arrivato alla versione 2.0 nel gennaio 2025, il tono si è fatto più netto: sviluppare nuove linee di prodotto in linguaggi non *memory-safe*, per il software che sostiene funzioni critiche, viene indicato come una cattiva pratica, e

ai produttori si chiede di pubblicare la propria roadmap entro la fine del 2025, scadenza ormai alle spalle, con l'eccezione dei prodotti la cui fine del supporto è prevista prima del 2030. Lo stesso documento precisa di non imporre alcun obbligo: non è una legge, ma una guida; sposta però le attese, e con esse la responsabilità di chi continua a ignorare il problema.

L'orientamento è confermato e dettagliato dal documento congiunto NSA e CISA sui linguaggi memory-safe del giugno 2025. In Europa la spinta è meno esplicita ma va nella stessa direzione, per la via del security by design: il Cyber Resilience Act, i cui requisiti essenziali di cybersicurezza si applicheranno dall'11 dicembre 2027, imporrà sicurezza fin dalla progettazione e gestione delle vulnerabilità per i prodotti con elementi digitali, mentre l'obbligo di segnalare le vulnerabilità attivamente sfruttate scatta già dall'11 settembre 2026. Il regolamento non nomina i linguaggi memory-safe, e il collegamento resta quindi inferenziale, ma un difetto di memoria evitabile alla radice è proprio il tipo di rischio che quell'obbligo mira a comprimere. La memory safety, insomma, sta migrando dal terreno delle buone pratiche a quello delle aspettative di mercato e, in prospettiva, degli obblighi.

La lezione di Android: non riscrivere tutto, cambiare il nuovo codice

La domanda ovvia è come si applichi tutto questo a basi di codice che contano decine di milioni di righe in C e C++. La risposta più convincente arriva da Google, che sul codice di Android ha ottenuto un risultato netto senza una riscrittura di massa. La serie storica è eloquente: la quota di vulnerabilità dovute a problemi di memoria è passata dal 76% del 2019 al 24% del 2024, e nel 2025, secondo la rilevazione

di novembre, è scesa sotto il 20% per la prima volta. Google non ha riscritto il vecchio codice, ha spostato sui linguaggi memory-safe il codice nuovo, e la differenza si legge nella densità dei difetti: una stima prudente colloca il codice Rust di Android intorno a 0,2 vulnerabilità di memoria per milione di righe, contro le circa mille del C e del C++, oltre mille volte in meno.

Funziona perché le vulnerabilità non si distribuiscono in modo uniforme nel tempo: il codice appena scritto ne contiene molte di più di quello che è invecchiato ed è stato ripulito a forza di correzioni. Concentrare i linguaggi sicuri dove nasce il rischio, cioè nelle nuove funzionalità, fa crollare la quota complessiva senza il costo proibitivo di rifare ciò che già funziona. E smonta anche l'idea che la sicurezza costi in produttività: sempre su Android, sulle modifiche medie e grandi il tasso di *rollback* delle modifiche in Rust è circa quattro volte più basso che in C++, e le modifiche in Rust passano in generale circa il 25% di tempo in meno in revisione. Non bisogna migrare tutto, insomma: bisogna smettere di aggiungere il problema.

Rust non risolve tutto: i limiti e la strada realistica

Sarebbe però un errore trasformare la memory safety in un tifo per un linguaggio. Anche il codice memory-safe ha i suoi confini: Rust prevede blocchi *unsafe* per le operazioni di basso livello e per il dialogo con il codice C, che riaprono in parte i rischi che il linguaggio altrove elimina. Il peso di questi blocchi, però, va misurato e non esagerato: nel codice Rust circa il 4% sta dentro blocchi *unsafe*, e *unsafe* non disattiva la maggior parte dei controlli del linguaggio. E soprattutto la sicurezza della memoria non elimina le altre categorie di bug, dalla logica errata alle vulnerabilità di iniezione: un programma memory-safe può essere insicuro in

mille altri modi. Ciò che si guadagna è la scomparsa della classe più numerosa e più sfruttata, non la sicurezza assoluta.

Resta poi l'enorme patrimonio di software che non si può realisticamente riscrivere, dai kernel al mondo *embedded* e industriale, dove C e C++ domineranno ancora a lungo. Per quel codice la risposta non è il linguaggio ma l'irrobustimento a più livelli: mitigazioni del compilatore, analisi statica lungo la pipeline di DevSecOps con strumenti come i SAST, fuzzing sistematico e architetture che confinano i danni. Che le difese a più livelli funzionino lo mostra un caso reale, arrivato peraltro dal versante opposto: una vulnerabilità del 2025 in un decoder Rust di Android (CVE-2025-48530), annidata in un blocco *unsafe*, non è mai arrivata in un rilascio pubblico, e il codice CVE le è stato assegnato solo per tracciarne la correzione con la giusta priorità. Il postmortem ha stabilito che l'allocatore *hardened* Scudo la rendeva comunque deterministicamente non sfruttabile grazie alle sue *guard pages*, che anzi hanno contribuito a scoprirla trasformando un overflow silenzioso in un crash rumoroso. Scudo è l'allocatore predefinito di Android, benché Google stia ancora lavorando con i partner per renderlo obbligatorio ovunque. È la difesa in profondità che fa il suo lavoro, e la conferma che la strada realistica è duplice: linguaggi memory-safe per il nuovo, difese a più livelli per il vecchio.

La pressione, del resto, si sente anche fuori da Android. Il mondo C++ ha reagito ai richiami delle agenzie scegliendo la via dei Profiles, un insieme di regole di sicurezza opzionali, dopo aver di fatto accantonato la proposta più radicale di un *borrow checker* in stile Rust, una scelta che diversi critici giudicano insufficiente a garantire una sicurezza stretta. Dentro Microsoft, un *distinguished engineer* ha reso pubblico a fine 2025 il proprio obiettivo, eliminare C e C++ dal codice dell'azienda entro il 2030, usando AI e algoritmi per tradurre i *codebase* più grandi, come da strategia del gruppo di ricerca

di cui fa parte. L'autore ha poi precisato che si tratta di un progetto di ricerca e non di una strategia aziendale, ma la sola circolazione dell'idea dice quanto si sia spostato il baricentro del dibattito. E un argomento nuovo rende tutto più urgente: quando la generazione di codice assistita dall'AI produce software più in fretta di quanto gli umani riescano a rivederlo, conviene che quel codice nasca in un linguaggio che certi errori non li permette nemmeno.

Cosa aspettarsi, senza illusioni

La memory safety non è una bacchetta magica e non renderà sicuro il software di colpo, ma è probabilmente l'intervento con il miglior rapporto tra costo e beneficio disponibile oggi: rimuove alla radice la classe di vulnerabilità più numerosa e più sfruttata, e lo fa nel punto più economico, cioè prima che il difetto esista. Per chi sviluppa significa adottare una disciplina di secure coding che parte dalla scelta del linguaggio per il codice nuovo; per chi compra software significa iniziare a chiedere ai fornitori la loro roadmap memory-safe, esattamente come oggi si chiede un SBOM. Il messaggio che arriva dalle agenzie è che continuare a produrre nuovi difetti di memoria, avendo gli strumenti per non farlo, non sarà più considerato normale. La memory safety, da nicchia per specialisti, è diventata il modo più concreto per ridurre la superficie d'attacco alla fonte.

Condividi sui Social Network:

<https://www.ictsecuritymagazine.com/articoli/memory-safety-linguaggi-sicuri/>