t-defence

Open source vulnerability hunting: Joe editor command injection

Code & OS Command Injection nel parsing del file tags

#TDefenceBusiness

Malware Lab

Summary

Our Malware Lab

T-Defence Malware Lab daily performs dissection of malware with the aim of timely understanding the technological evolutions of attacks, consolidating the knowledge of necessary to make more effective and faster the process of incidents responding, contributing to spreading information about emerging threats into the expert's community and among its clients.

Malware Lab analysts are continuously engaged in searching and experimenting new analysis tools, for increasing accuracy and scope of action with regard to the proliferation of new evasion and anti-analysis techniques adopted by malware.

The Malware Lab is also committed to the development of proprietary tools for malware analysis and supporting the management and response of incidents.

Besides malware analysis, Malware Lab ideated and implemented an automatic process of extraction of **Indicators of Compromise (IOC)** that is daily run on dozens of new malwares, intercepted in the wide for populating our Knowledge Base.

Executive Summary

La funzionalità di ricerca per tag di JOE (Joe's Own Editor[1]) consente di saltare alla definizione di un simbolo leggendo un file `tags`. Il campo `FILE` della voce corrispondente viene passato direttamente alle routine di apertura file dell'editor. Per una convenzione storica di JOE, un nome file che inizia con il carattere ! non viene trattato come percorso, bensì eseguito tramite `/bin/sh -c`. Poiché nessuna validazione viene effettuata sul campo `FILE` letto dal file `tags`, una voce malevola permette l'esecuzione di comandi arbitrari nel momento in cui la vittima esegue una ricerca per tag corrispondente.

La vulnerabilità è stata corretta nella repository ufficiale[2] tramite la Pull Request #105[3] il 20 aprile 2026. Il presente documento descrive in forma sintetica la natura del difetto, il vettore di attacco e la correzione applicata.

Classificazione:

- **Tipologia:** Code Injection (CWE-94)[4] e OS Command Injection (CWE-78)[5]
- **Severità:** CVSS 3.1 base score **7.8 (High)** - Vettore AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H.
- **Stato:** corretto a monte in PR #105.

Autori:

- Gaetano Zappulla: CISO
- Alberico Ciriello: Responsabile Security Operations

[1] Joe's Own Editor – sito ufficiale del progetto: https://joe-editor.sourceforge.io/

[2] Repository ufficiale GitHub: https://github.com/joe-editor/joe

[3] Pull Request #105 – «Fix tags file security vulnerability», merge del 20/04/2026: https://github.com/joe-editor/joe/pull/105

[4] Definizione CWE-94 (Improper Control of Generation of Code – 'Code Injection'), MITRE: https://cwe.mitre.org/data/definitions/94.html

[5] Definizione CWE-78 (Improper Neutralization of Special Elements used in an OS Command – 'OS Command Injection'), MITRE: https://cwe.mitre.org/data/definitions/78.html

Lo strumento interessato

JOE (Joe's Own Editor) è un editor di testo da terminale, distribuito sotto licenza GNU GPL, attivo dal 1988 e incluso di serie in numerose distribuzioni Linux. Adotta un'interfaccia «mode-less» ispirata a WordStar e al pacchetto «Turbo» di Borland, risultando familiare a chi proviene da quegli ambienti, pur offrendo le funzionalità complete di un editor UNIX evoluto per la modifica di codice e testo.

Il programma viene distribuito in più varianti che condividono il medesimo motore: `joe`, `jmacs`, `jpico` e `jstar`. Tra le sue funzionalità rientra la ricerca per tag (associata di default alla combinazione ^K ;), che consente di navigare verso la definizione di un simbolo a partire da un indice precalcolato e memorizzato in un file `tags`, lo stesso meccanismo al centro della vulnerabilità analizzata in questo documento.

Componenti affetti

File e funzione: il difetto risiede nel parser del file tags in joe/utag.c (funzione dotag()), che inoltra il campo FILE di una voce alle routine di caricamento file senza alcuna validazione. La convenzione di shell-escape sfruttata è invece implementata in joe/b.c: qui bload() instrada i nomi file che iniziano con ! verso joe_popen() e quindi verso /bin/sh -c.

Versione affetta: JOE 4.7 (ramo main, corrente al momento della segnalazione). Ci si attende l'interessamento di tutte le release precedenti che condividono lo stesso parser di utag.c e la medesima convenzione di shell-escape di bload(). L'impatto su JOE 4.6 è stato confermato in fase di revisione della Pull Request.

Binari interessati: le personalità predefinite joe, jmacs, jpico e jstar sono affette da questo problema. Fa eccezione rjoe, che disabilita il binding della ricerca per tag in rjoerc.in e non risulta quindi vulnerabile nella configurazione di default.

Causa principale

La funzione di ricerca per tag analizza il file tags e passa il campo `FILE` della voce corrispondente alle routine di caricamento file `bfind()` / `bload()`. Quest'ultima rispetta la convenzione di JOE secondo cui i nomi file che iniziano con ! vengono eseguiti tramite `/bin/sh -c` (comportamento concepito per i prompt interattivi). Tuttavia, non viene effettuata alcuna validazione sul campo `FILE` letto dal file `tags`.

È quindi sufficiente scrivere il seguente codice malevolo per innescarne l'esecuzione:

```
intmain<TAB>!/path/to/payload<TAB>1
```

Questo comporta l'esecuzione di `/bin/sh -c /path/to/payload` nell'istante in cui la vittima esegue una ricerca per tag corrispondente. In presenza dell'opzione `-notagsmenu` (o in caso di singolo match) non viene mostrato alcun prompt di conferma, eliminando l'unica interazione che potrebbe insospettire l'utente.

Trigger

Ciò che fa emergere la vulnerabilità è l'esecuzione di una ricerca per tag (default ^K ;) da parte della vittima all'interno di un progetto che contiene un file tags di provenienza non fidata. La condizione di innesco non è la semplice apertura di un file, bensì la corrispondenza (match) tra il simbolo cercato e una voce malevola del file `tags`, il cui campo `FILE` inizia con !.

Scenari tipici: clonazione di un repository di terze parti che include un file `tags` predisposto dall'attaccante, oppure un file `tags` generato in una directory condivisa o scrivibile da altri. L'assenza del menu di conferma con `-notagsmenu` o su singolo risultato, rende l'esecuzione immediata e silenziosa.

Fix

Corretto a monte da Joseph H. Allen (maintainer del progetto, `jhallen`) tramite la PR #105, confluita nel ramo `main` il 20 aprile 2026 con il commit `85cbce7`.

Il difetto è stato risolto rifiutando in fase di parsing i campi `FILE` che verrebbero interpretati da `bload()` come qualcosa di diverso da un nome file, prima che possano raggiungere l'esecuzione della shell. Viene così introdotta, a tal fine, una funzione di controllo `hack_check()` e ne aggiunge la chiamata di guardia nel parser di `utag.c`.

Nuova funzione di controllo (`joe/b.c`)

```
+ /* Return true if filename will be interpreted as something other than a filename by bload()
+    It would be better to have a flag on bload, but more plumbing required */
+
+ int hack_check(const char *name)
+ {
+     if (name[0] == '!')
+         return 1;
+     else if (name[0] == '>' && name[1] == '>')
+         return 1;
+     else
+         return 0;
+ }
```

Guardia nel parser (joe/utag.c, funzione dotag())

```
-             if (x != y) {
+             if (x != y && !hack_check(buf + x)) { /* Do not allow shell commands in tags file! */
```

In questo modo, qualsiasi campo `FILE` che inizi con `!` o con `>>` viene scartato prima di poter essere passato a `bload()`, neutralizzando sia l'esecuzione via `/bin/sh -c` sia la redirezione su file. La correzione mantiene intatto il comportamento legittimo per i nomi di file ordinari.

Credits

Vulnerabilità individuata, analizzata e segnalata da Gaetano Zappulla, CISO di Defence Tech S.p.A.; correzione sviluppata e rilasciata da Joseph H. Allen.

t-defence

Next | Donexit | Foramil | Innodesi

Via Giacomo Peroni, 452 – 00131 Roma
tel. 06.45752720 – info@defencetech.it
www.t-defence.it

#TDefenceBusiness