oniux: l'isolamento Tor a livello kernel per qualsiasi applicazione Linux (e addio torsocks)

oniux: l'isolamento Tor a livello kernel per qualsiasi applicazione Linux (e addio torsocks)

Il sogno di ogni sysadmin paranoico (o di chiunque tenga alla propria OPSEC) è semplice: lanciare un programma e avere la certezza matematica che nessun pacchetto IP possa leakare al di fuori di Tor. La realtà invece è un inferno di proxy configurati male, librerie dynamic-link che fanno le bizze e applicativi che decidono arbitrariamente di fare chiamate di rete in chiaro aggirando i vostri sforzi.

Per anni lo standard di fatto per forzare applicazioni terze a passare attraverso Tor è stato `torsocks`. Tuttavia, si tratta di un residuo basato su tecniche datate che oggi mostrano evidenti limiti.

Oggi parliamo di **oniux**, la nuova utility a riga di comando presentata dal Tor Project, progettata per archiviare definitivamente il passato e blindare le applicazioni a livello kernel.

Il problema con torsocks (La morte di LD_PRELOAD)

Come funziona `torsocks`? Sfrutta la variabile d'ambiente `LD_PRELOAD` per intercettare (dirottare) le funzioni di rete della libreria standard C (`libc`). Quando un'applicazione dinamica cerca di aprire una socket, `torsocks` si intromette e costringe la chiamata a instradarsi attraverso il proxy SOCKS locale di Tor.

Questo approccio, seppur portabile, ha dei buchi grandi quanto una casa:

1. **Binari statici**: Se un binario è compilato staticamente (come accade regolarmente in Go, Rust o nel mondo Zig), non carica le

librerie dinamiche. `LD_PRELOAD` viene ignorato completamente e il programma comunica felicemente in chiaro.
2. **System call dirette**: Se l'applicativo effettua system call di rete dirette bypassando la `libc`, `torsocks` non può intercettarle.
3. **Malware ed evasioni**: Qualsiasi codice minimamente ostile può eludere l'hijack della `libc` con estrema facilità.

In parole povere: se usate `torsocks`, state affidando la vostra libertà alla speranza che il compilatore altrui sia stato "gentile". Spoiler: non lo è.

La via del Kernel: Tor + Namespaces

`oniux` risolve il problema alla radice spostando l'isolamento dall'applicativo al kernel Linux, sfruttando i **Namespaces**. I namespaces sono la stessa tecnologia primitiva alla base di Docker e dei container.

Quando avviate un programma con `oniux`, l'utility esegue le seguenti operazioni:

1. **Isolamento di rete**: Crea un nuovo network namespace dedicato. Da questo ambiente viene rimossa ogni interfaccia fisica della macchina host (come `eth0` o `wlan0`). Il programma si ritrova in una cella di isolamento senza alcuna rotta verso la WAN.
2. **L'interfaccia onion0**: All'interno del namespace viene montata un'interfaccia TUN virtuale chiamata `onion0`, gestita da **onionmasq** (il framework di tunneling del Tor Project) e basata su **Arti** (la riscrittura di Tor in Rust). Questa è l'unica interfaccia di rete visibile.
3. **DNS Blindato**: Attraverso un mount namespace privato, `oniux` inietta una versione ad-hoc di `/etc/resolv.conf` che punta direttamente al resolver DNS sicuro fornito da `onionmasq`. Le query DNS non possono scappare all'esterno.

Il risultato? Il programma isolato non deve essere configurato. Può fare chiamate di rete normali, convinto di trovarsi su internet, ma ogni singolo bit TCP/UDP o DNS inviato a qualsiasi indirizzo viene forzato dal kernel a transitare sul TUN `onion0` e quindi cifrato e instradato nella rete Tor. Se il tunnel muore, la rete muore. Zero leak.

Vantaggi e Svantaggi di oniux

Come ogni strumento di sicurezza, anche `oniux` ha i suoi pro e contro.

Vantaggi:

- **Nessun leak per errore**: L'applicazione non ha letteralmente rotte per la rete in chiaro. Anche se si sforza di bypassare i proxy, non può farlo.
- **Supporto universale**: Funziona con binari compilati staticamente (Go, Rust, Zig, C statico).
- **Trasparenza**: Non occorre configurare porte SOCKS o HTTP proxy nell'applicazione target.

Svantaggi:

- **Sperimentale**: Lo strumento è giovane, basato sulla codebase di Arti e onionmasq ancora in fase di maturazione.
- **Solo Linux**: Richiede le feature dei namespace del kernel Linux e il modulo `tun` caricato.
- **Falle di IPC (Inter-Process Communication)**: Se lanciate con `oniux` un'applicazione che si collega a un demone già in esecuzione sul sistema host (ad esempio, un browser che riutilizza una sessione Firefox aperta fuori dal namespace), il traffico potrebbe transitare fuori da Tor attraverso i socket IPC.

Note e Accorgimenti Pratici

Per utilizzare `oniux` senza intoppi, tenete a mente queste regole:

1. **Caricare il modulo TUN**: Il kernel deve supportare i dispositivi TUN. Se riscontrate errori all'avvio, assicuratevi che il modulo sia caricato:

   ```
   sudo modprobe tun
   ```

2. **Esempi di utilizzo**: Per verificare che l'isolamento sia attivo, potete interrogare un servizio di IP lookup:

   ```
   oniux curl https://icanhazip.com
   ```

Oppure lanciare una shell blindata in cui ogni comando successivo passerà automaticamente sotto Tor:

```
oniux bash
```

3. **Debug logging**: Se qualcosa non funziona, potete abilitare il log dettagliato di Arti/onionmasq impostando la variabile d'ambiente di Rust:

```
RUST_LOG=debug oniux curl http://example.onion
```

Il futuro dell'isolamento Tor su Linux è qui, ed è scritto in Rust a livello kernel. Archiviate `torsocks` e iniziate a fare OPSEC seria.

[Che il talpone sia con voi]

Fluxer Icon

Canale di discussione e supporto

Canale di discussione e supporto. Per confrontarti su configurazioni o altro entra su fluxer.dreadful.work (https://fluxer.dreadful.work/invite/dreadful).