RFC-023: Agent Spawning and Sandboxing

Status: Draft **Date:** January 2026 **Author:** Derrell Piper ddp@eludom.net
Implementation: Proposed

Abstract

This RFC specifies agent spawning and sandboxing for the Library of Cyberspace: how autonomous agents are created, constrained, monitored, and terminated. Agents operate with capability-based authority in isolated sandboxes, enabling safe delegation of tasks while maintaining security boundaries.

Motivation

The General Magic Telescript vision from 1994:

> "Programs that travel from machine to machine, carrying your authority, doing things while you sleep."

That vision failed because:

- **No security model** - Agents ran with ambient authority
- **No isolation** - One agent could corrupt another
- **No accountability** - No audit of agent actions
- **No revocation** - Once launched, agents were uncontrollable

The Library realizes this vision with:

- **Capability-based authority** - Agents have only granted permissions
- **Sandbox isolation** - Process, filesystem, network boundaries
- **Full audit trail** - Every agent action is logged
- **Remote termination** - Agents can be killed at any time

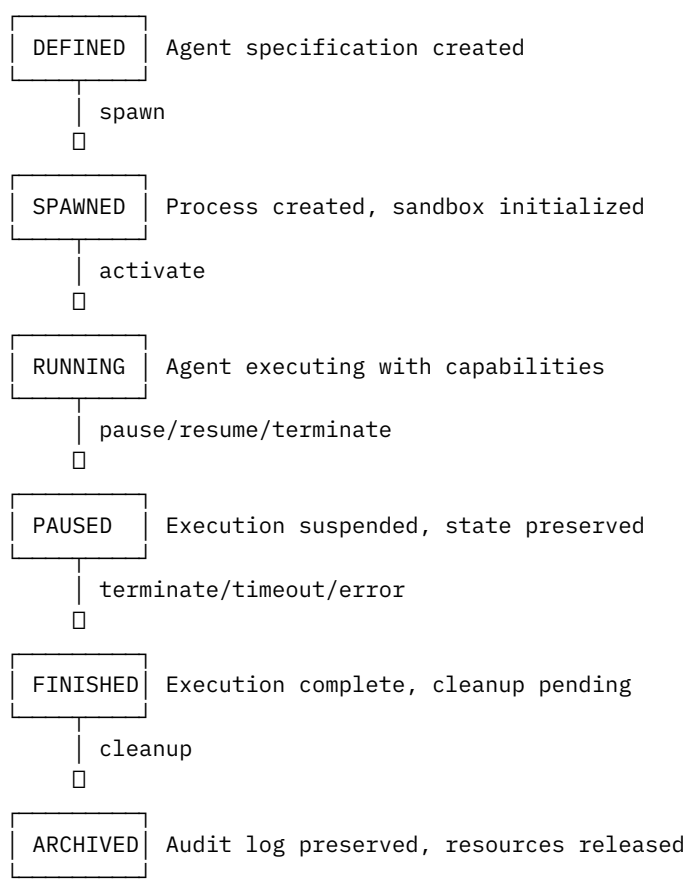
Agent Model

What is an Agent?

```
(agent
  (id "agent-2026-001")
  (spawned-by user-principal)
  (purpose "Monitor RSS feeds and archive new papers")
  (capabilities
    (read (url "https://arxiv.org/rss/*"))
    (write (path "/vault/papers/"))
    (network (hosts ("arxiv.org" "doi.org"))))
```

```
  (constraints
    (max-runtime (* 24 3600))  ; 24 hours
    (max-memory (* 512 1024 1024))  ; 512MB
    (max-storage (* 1 1024 1024 1024)))  ; 1GB
  (sandbox posix-sandbox)
  (status running))
```

Agent Lifecycle



Spawning Agents

Spawn Request

```
(define (spawn-agent spec)
  "Spawn new agent from specification"
  (let* ((agent-id (generate-agent-id))
```

```
        (spawner (current-principal))

        ;; Validate spawner has required capabilities
        (_ (verify-spawn-authority spawner (spec-capabilities spec)))

        ;; Create sandbox
        (sandbox (create-sandbox (spec-sandbox-type spec)
                                 (spec-constraints spec)))

        ;; Create agent record
        (agent (make-agent
                id: agent-id
                spawned-by: spawner
                spec: spec
                sandbox: sandbox
                status: 'spawned)))

    ;; Audit spawn
    (audit-append
      action: `(agent-spawn ,agent-id)
      motivation: (spec-purpose spec))

    ;; Initialize agent process
    (sandbox-exec sandbox (spec-code spec))

    ;; Return agent handle
    agent))
```

Capability Verification

```
(define (verify-spawn-authority spawner requested-caps)
  "Verify spawner can grant requested capabilities"
  (for-each
    (lambda (cap)
      (unless (authorized? spawner 'delegate cap)
        (error "Cannot delegate capability" cap)))
    requested-caps))
```

Agent Code Loading

```
(define (load-agent-code spec)
  "Load and verify agent code"
  (let* ((code-hash (spec-code-hash spec))
         (code (cas-get code-hash)))

    ;; Verify code hash
```

```
  (unless (equal? code-hash (sha256 code))
    (error "Code integrity failure"))

  ;; Verify code signature (if required)
  (when (spec-require-signed? spec)
    (unless (verify-code-signature code (spec-signer spec))
      (error "Code signature invalid")))

  code))
```

Sandbox Types

POSIX Sandbox

Process isolation using OS primitives:

```
(define (create-posix-sandbox constraints)
  "Create POSIX-based sandbox"
  (sandbox
    (type posix)

    ;; Process isolation
    (process
      (uid (allocate-sandbox-uid))
      (gid (allocate-sandbox-gid))
      (chroot (create-sandbox-root))
      (rlimits
        (cpu ,(constraints-max-cpu constraints))
        (memory ,(constraints-max-memory constraints))
        (files ,(constraints-max-files constraints))
        (processes 1)))  ; No forking

    ;; Filesystem isolation
    (filesystem
      (root ,(sandbox-root))
      (mounts
        (("/lib" read-only)
         ("/usr/lib" read-only)
         ,(sandbox-work-dir) read-write)))

    ;; Network isolation
    (network
      (allowed-hosts ,(constraints-network-hosts constraints))
      (allowed-ports ,(constraints-network-ports constraints)))))
```

Scheme Sandbox

Language-level isolation for Scheme agents:

```
(define (create-scheme-sandbox constraints)
  "Create Scheme-level sandbox"
  (sandbox
    (type scheme)

    ;; Safe environment - no dangerous primitives
    (environment
      (import (scheme base)
              (scheme write)
              (library sandbox-io)
              (library sandbox-net))
      (exclude system exit eval load
               file-delete directory-delete
               process-fork process-exec))

    ;; Resource limits via fuel
    (fuel
      (computation ,(constraints-max-steps constraints))
      (allocation ,(constraints-max-memory constraints)))

    ;; Capability-restricted I/O
    (io-capabilities ,(constraints-io constraints))))
```

Container Sandbox

OCI container isolation:

```
(define (create-container-sandbox constraints image)
  "Create container-based sandbox"
  (sandbox
    (type container)

    ;; Container configuration
    (container
      (image ,image)
      (readonly-rootfs #t)
      (no-new-privileges #t)
      (cap-drop ALL)
      (cap-add ,(minimal-caps constraints)))

    ;; Resource limits
    (resources
      (memory ,(constraints-max-memory constraints))
```

```
      (cpu-shares ,(constraints-cpu-shares constraints))
      (pids-limit 10))

    ;; Network policy
    (network
      (mode bridge)
      (egress-policy ,(network-policy constraints)))))
```

WASM Sandbox

WebAssembly isolation:

```
(define (create-wasm-sandbox constraints)
  "Create WebAssembly sandbox"
  (sandbox
    (type wasm)

    ;; WASM runtime configuration
    (runtime
      (memory-limit ,(constraints-max-memory constraints))
      (table-limit 10000)
      (fuel ,(constraints-max-steps constraints)))

    ;; WASI capabilities
    (wasi
      (preopens ,(wasi-preopens constraints))
      (env ,(wasi-env constraints))
      (args ,(wasi-args constraints)))

    ;; Host function imports (minimal)
    (imports
      (log "library:log")
      (cas-get "library:cas_get")
      (cas-put "library:cas_put"))))
```

Capability Enforcement

Capability Proxy

All agent I/O goes through capability-checking proxies:

```
(define (make-capability-proxy agent capabilities)
  "Create proxy that enforces capability checks"

  (lambda (operation . args)
    (let ((required-cap (operation->capability operation args)))
```

```
      ;; Check capability
      (unless (capability-granted? capabilities required-cap)
        (audit-violation agent operation required-cap)
        (error "Capability denied" operation))

      ;; Audit allowed operation
      (audit-agent-action agent operation args)

      ;; Execute operation
      (apply (operation->handler operation) args))))

(define (operation->capability op args)
  "Map operation to required capability"
  (case op
    ((read-file)
     `(read (path ,(car args))))
    ((write-file)
     `(write (path ,(car args))))
    ((http-get)
     `(network (url ,(car args))))
    ((cas-get)
     `(read (hash ,(car args))))
    (else
     (error "Unknown operation" op))))
```

Attenuation on Delegation

```
(define (agent-spawn-child parent-agent child-spec)
  "Agent spawning child agent with attenuated capabilities"
  (let* ((parent-caps (agent-capabilities parent-agent))
         (requested-caps (spec-capabilities child-spec))

         ;; Child can only have subset of parent's capabilities
         (child-caps (capability-intersect parent-caps requested-caps)))

    (unless (equal? child-caps requested-caps)
      (warn "Child capabilities attenuated"))

    (spawn-agent (spec-with-capabilities child-spec child-caps))))
```

Agent Communication

Message Passing

Agents communicate via typed, capability-checked messages:

```
(define (agent-send from-agent to-agent message)
  "Send message between agents"

  ;; Verify sender has send capability to receiver
  (unless (capability-granted? (agent-capabilities from-agent)
                               `(send (agent ,(agent-id to-agent))))
    (error "Cannot send to agent"))

  ;; Verify message type allowed
  (unless (message-type-allowed? from-agent to-agent (message-type message))
    (error "Message type not allowed"))

  ;; Queue message
  (mailbox-enqueue (agent-mailbox to-agent)
                   (signed-message from-agent message))

  ;; Audit
  (audit-append
    action: `(agent-message ,(agent-id from-agent) ,(agent-id to-agent))
    motivation: (message-type message)))

(define (agent-receive agent #!key timeout)
  "Receive message from mailbox"
  (let ((msg (mailbox-dequeue (agent-mailbox agent) timeout: timeout)))
    (when msg
      ;; Verify signature
      (verify-message-signature msg)
      msg)))
```

Shared State

Agents can share state through CAS:

```
(define (agent-share agent hash recipients)
  "Share CAS object with other agents"

  ;; Verify agent can read the object
  (unless (capability-granted? (agent-capabilities agent)
                               `(read (hash ,hash)))
    (error "Cannot read object to share"))

  ;; Grant read capability to recipients
```

```
  (for-each
    (lambda (recipient)
      (grant-capability agent recipient `(read (hash ,hash))))
    recipients)

  ;; Audit sharing
  (audit-append
    action: `(agent-share ,hash ,recipients)))
```

Monitoring and Control

Agent Status

```
(define (agent-status agent)
  "Get current agent status"
  `(agent-status
    (id ,(agent-id agent))
    (status ,(agent-state agent))
    (uptime ,(- (current-time) (agent-start-time agent)))
    (resources
      (memory ,(sandbox-memory-usage (agent-sandbox agent)))
      (cpu ,(sandbox-cpu-usage (agent-sandbox agent)))
      (storage ,(sandbox-storage-usage (agent-sandbox agent))))
    (actions ,(agent-action-count agent))
    (messages-sent ,(agent-messages-sent agent))
    (messages-received ,(agent-messages-received agent))))
```

Agent Control

```
(define (agent-pause agent #!key reason)
  "Pause agent execution"
  (unless (authorized? (current-principal) 'control agent)
    (error "Not authorized to control agent"))
  (sandbox-pause (agent-sandbox agent))
  (set-agent-state! agent 'paused)
  (audit-append action: `(agent-pause ,(agent-id agent)) motivation: reason))

(define (agent-resume agent)
  "Resume paused agent"
  (unless (authorized? (current-principal) 'control agent)
    (error "Not authorized to control agent"))
  (sandbox-resume (agent-sandbox agent))
  (set-agent-state! agent 'running)
  (audit-append action: `(agent-resume ,(agent-id agent))))
```

```
(define (agent-terminate agent #!key reason)
  "Terminate agent immediately"
  (unless (authorized? (current-principal) 'control agent)
    (error "Not authorized to control agent"))
  (sandbox-kill (agent-sandbox agent))
  (set-agent-state! agent 'terminated)
  (audit-append
    action: `(agent-terminate ,(agent-id agent))
    motivation: reason
    priority: 'high)
  (cleanup-agent agent))
```

Watchdog

```
(define (agent-watchdog)
  "Monitor all agents, enforce constraints"
  (for-each
    (lambda (agent)
      (when (eq? (agent-state agent) 'running)
        ;; Check resource limits
        (when (> (sandbox-memory-usage (agent-sandbox agent))
                 (agent-max-memory agent))
          (agent-terminate agent reason: "Memory limit exceeded"))

        ;; Check runtime limit
        (when (> (agent-uptime agent) (agent-max-runtime agent))
          (agent-terminate agent reason: "Runtime limit exceeded"))

        ;; Check heartbeat
        (when (> (- (current-time) (agent-last-heartbeat agent))
                 agent-heartbeat-timeout)
          (agent-terminate agent reason: "Heartbeat timeout"))))
    (all-agents)))
```

Soup Integration

Agents in the Soup

```
(soup-object
  (name "agent/2026-001")
  (type agent)
  (size "145KB")
  (status running)
  (spawned-by "ddp@eludom.net")
  (purpose "Archive arxiv papers")
```

```
  (runtime "4h 23m")
  (resources (memory "234MB") (storage "890MB"))
  (capabilities (read "arxiv.org/*") (write "/vault/papers/")))
```

Querying Agents

```
;; All running agents
(soup-query type: 'agent status: 'running)

;; Agents spawned by user
(soup-query type: 'agent spawned-by: user-principal)

;; Agents with network access
(soup-query type: 'agent has-capability: 'network)

;; Resource hogs
(soup-query type: 'agent min-memory: (* 256 1024 1024))
```

Agent Introspection

```
;; Agent can query itself
(define (agent-self-inspect)
  `(self
    (id ,(current-agent-id))
    (capabilities ,(current-capabilities))
    (resources-remaining
      (memory ,(- max-memory (current-memory)))
      (runtime ,(- max-runtime (current-uptime)))
      (fuel ,(remaining-fuel)))))
```

Agent Patterns

Pattern 1: Periodic Task Agent

```
(spawn-agent
  (code '(lambda ()
           (let loop ()
             (perform-task)
             (sleep 3600)  ; hourly
             (loop))))
  (capabilities
    (read "/vault/feeds/")
    (write "/vault/archive/")
    (network ("rss.example.com")))
  (constraints
```

```
    (max-runtime (* 30 24 3600))  ; 30 days
    (max-memory (* 128 1024 1024))))
```

Pattern 2: One-Shot Processing Agent

```
(spawn-agent
  (code '(lambda ()
           (let ((input (cas-get input-hash)))
             (let ((result (process input)))
               (cas-put result)))))
  (capabilities
    (read (hash input-hash))
    (write "/vault/results/"))
  (constraints
    (max-runtime 3600)
    (max-memory (* 1024 1024 1024))))
```

Pattern 3: Reactive Agent

```
(spawn-agent
  (code '(lambda ()
           (let loop ()
             (let ((msg (agent-receive timeout: 60000)))
               (when msg
                 (handle-message msg))
               (loop)))))
  (capabilities
    (receive (from supervisor-agent))
    (send (to worker-agents))
    (read "/vault/tasks/")
    (write "/vault/results/"))
  (constraints
    (max-runtime #f)  ; runs until terminated
    (max-memory (* 256 1024 1024))))
```

Pattern 4: Mobile Agent

```
;; Agent that migrates between vaults
(spawn-agent
  (code '(lambda ()
           (let ((data (gather-local-data)))
             (migrate-to remote-vault data)
             (process-remote data)
             (migrate-home results))))
  (capabilities
    (read "/vault/local/")
    (migrate (vaults (vault-a vault-b vault-c)))
```

```
  (constraints
    (max-migrations 10)
    (max-runtime (* 24 3600))))
```

Security Considerations

Escape Prevention

```
;; Sandbox escape mitigations
(define sandbox-security
  '((seccomp "restrict system calls")
    (namespaces "process/network/mount isolation")
    (capabilities "drop all Linux capabilities")
    (no-setuid "prevent privilege escalation")
    (read-only-root "immutable rootfs")
    (no-raw-sockets "prevent network attacks")))
```

Resource Exhaustion

```
;; Prevent denial of service
(define resource-limits
  '((memory "hard limit, OOM killer")
    (cpu "cgroups CPU quota")
    (disk "quota or sparse files")
    (network "bandwidth limiting")
    (processes "prevent fork bombs")
    (file-descriptors "prevent fd exhaustion")))
```

Information Leakage

```
;; Prevent covert channels
(define isolation-measures
  '((timing "fuel-based execution, no precise timers")
    (filesystem "no access outside sandbox")
    (network "egress filtering")
    (ipc "message passing only, no shared memory")
    (environment "sanitized env vars")))
```

Malicious Agents

```
(define (detect-malicious-behavior agent)
  "Heuristics for detecting malicious agents"
  (or
    ;; Excessive resource usage
    (> (agent-resource-velocity agent) threshold)
    ;; Unusual network patterns
```

```
(suspicious-network-activity? agent)
;; Repeated capability violations
(> (agent-violation-count agent) max-violations)
;; Anomalous message patterns
(anomalous-messaging? agent)))
```

Implementation Notes

Dependencies

Component	Implementation
Process sandbox	pledge/unveil (OpenBSD), seccomp (Linux)
Container sandbox	runc, crun
WASM sandbox	wasmtime, wasmer
Scheme sandbox	custom safe environment

Performance

- Sandbox creation: ~100ms (container), ~10ms (process), ~1ms (WASM)
- Message passing: ~10 s (local), ~1ms (cross-sandbox)
- Capability check: ~100ns (cached), ~10 s (chain validation)

References

1. Telescript Technology: Mobile Agents
2. Capsicum: Practical Capabilities for UNIX
3. WebAssembly System Interface (WASI)
4. RFC-021: Capability Delegation
5. RFC-003: Cryptographic Audit Trail
6. E Programming Language - Object capabilities

Changelog

- **2026-01-07** - Initial draft

Implementation Status: Draft **Dependencies:** sandbox (OS-specific), capabilities, audit **Integration:** Vault operations, distributed processing, automation