Agentic Cybersecurity: Engineering Autonomous AI Defense

Table of Contents

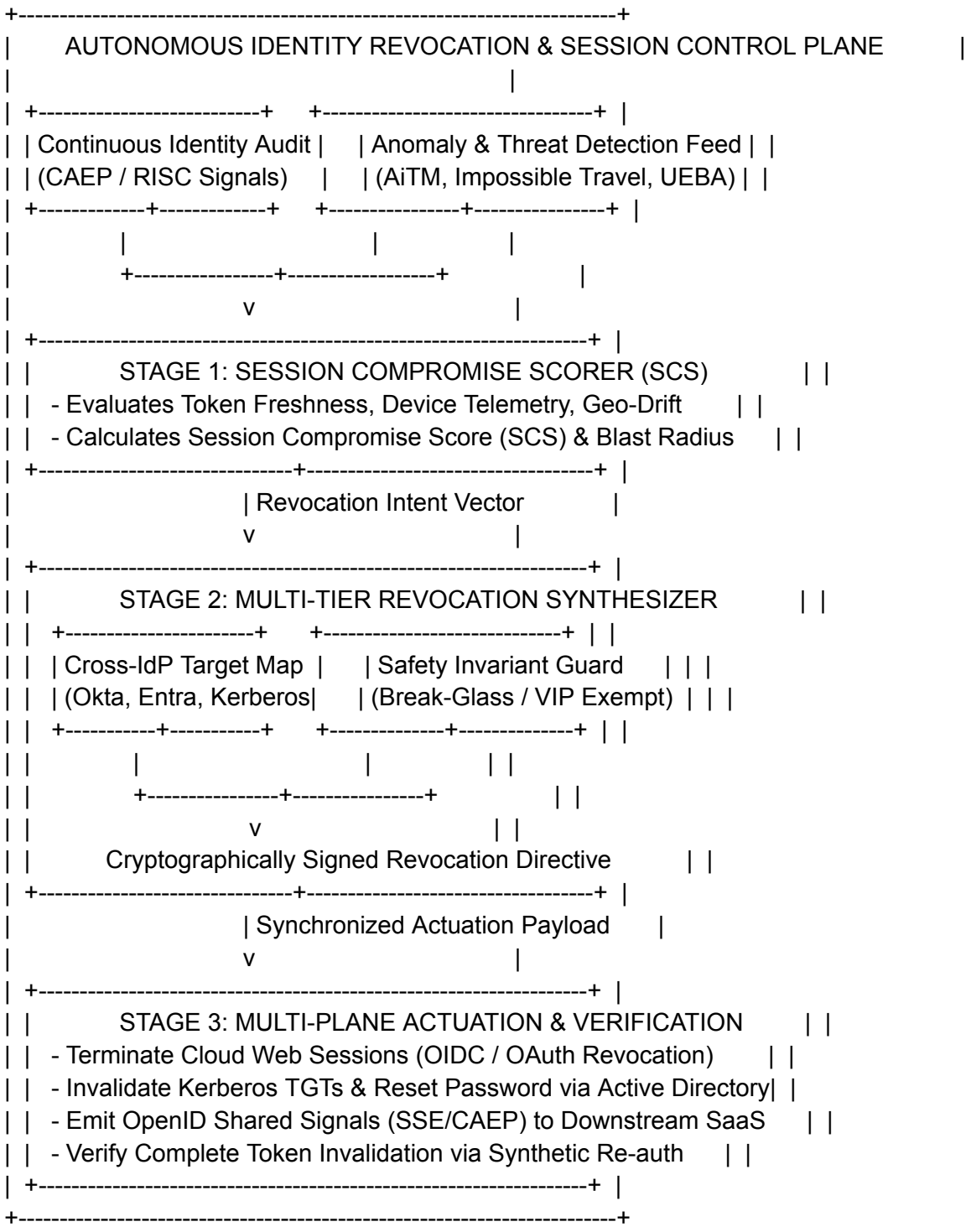
36. Deterministic Guardrails: Constraining LLM Outputs via JSON Schemas and Rule Engines
37. Cryptographic Provenance and Audit Trails for Autonomous SecOps Decisions
38. Human-in-the-Loop Orchestration: Approval Gates, Escalation Policies, and Interventions
39. LLMOps and Observability for Cyber Agents: Tracing, Latency Optimization, and Cost Control
40. Deploying Local and Air-Gapped LLMs for Highly Sensitive Security Operations
41. Resilient Message Brokers: Scaling Agent Workloads with Kafka and Celery
42. Compliance as Code: Autonomous Regulatory Auditing for SOC2, ISO 27001, and HIPAA
43. Supply Chain Security Agents: Monitoring Dependencies, SBOMs, and Package Repositories
44. Autonomous Threat Hunting in Kubernetes and Cloud-Native Environments
45. Edge Defense: Deploying Lightweight Agentic Models on Gateways and IoT Systems
46. Benchmark Datasets and Evaluation Frameworks for Cybersecurity Agents
47. Post-Mortem and Incident Report Generation: Automated Synthesis for Stakeholders
48. The Future of Agentic Cyber Warfare: Autonomous Offensive AI and Counter-Agent

Example:

Chapter 23: Autonomous Identity Revocation and Session Termination Strategies
When an adversary bypasses perimeter defenses via adversary-in-the-middle (AiTM) phishing, session cookie theft, or OAuth token extraction, network-layer controls alone cannot arrest unauthorized execution. In modern decentralized enterprises, identity is the primary control plane. If an attacker acquires a valid Primary Refresh Token (PRT), Kerberos Ticket Granting Ticket (TGT), or OAuth2 refresh token, they inherit legitimate authorization paths across SaaS applications, cloud management planes, and internal microservices.

Traditional security orchestration tools handle identity compromise through slow, disjointed runbooks. These legacy automations typically disable an Active Directory account while leaving active federated web sessions, refresh tokens, and long-lived cloud API sessions entirely intact.

Autonomous Identity Revocation and Session Termination Strategies replace fractured manual offboarding with synchronized, multi-plane identity containment. When an autonomous security agent detects session anomalies or compromised credentials, it executes a cross-plane revocation sweep across identity providers (IdPs), token issuers, and proxy layers within milliseconds.



Algorithmic Formulation: Session Compromise Score ($\mathcal{SCS}$)

To eliminate false-positive lockouts of privileged executives and critical service accounts, an autonomous agent calculates a Session Compromise Score ($\mathcal{SCS}$) before executing destructive revocation commands.

1. Identity & Context Anomaly Factor ($\mathcal{A}_{id}$)

Let $\Delta G \in [0.0, 1.0]$ denote the geographical velocity anomaly (e.g., impossible travel speed), $\mathcal{D}{dev} \in {0.0, 1.0}$ denote an unrecognized or unmanaged device fingerprint, and $\mathcal{F}{mfa} \in [0.0, 1.0]$ represent MFA fatigue or anomalous bypass telemetry:

$$\mathcal{A}{id} = \min\left(1.0, , w_1 \Delta G + w_2 \mathcal{D}{dev} + w_3 \mathcal{F}_{mfa}\right)$$

Where default calibrated weights are $w_1 = 0.40$, $w_2 = 0.35$, and $w_3 = 0.25$.

2. Entitlement Criticality Factor ($\mathcal{E}_{crit}$)

The operational risk of an account depends on its assigned role hierarchy $\mathcal{R}{priv} \in [1.0, 10.0]$ and the scope of downstream resources accessible via its current token bindings $N{res}$:

$$\mathcal{E}{crit} = \min\left(1.0, , \left(\frac{\mathcal{R}{priv}}{10}\right) \cdot \log_{10}(N_{res} + 9)\right)$$

3. Composite $\mathcal{SCS}$ and Revocation Thresholds

The composite score scales context anomalies against core detection fidelity $\mathcal{F}_{det} \in [0.0, 1.0]$, offset by role criticality to adjust the threshold of action:

$$\mathcal{SCS} = \mathcal{F}{det} \cdot \left(\mathcal{A}{id} + \mu \cdot \mathcal{E}_{crit}\right)$$

Where $\mu = 0.20$ is the privilege escalation bias. The agent maps $\mathcal{SCS}$ directly to graduated containment operations:

$$\text{Action}(\mathcal{SCS}) = \begin{cases}

\text{FULL_IDENTITY_LOCKOUT} & \text{if } \mathcal{SCS} \ge 0.85 \

\text{GLOBAL_SESSION_REVOCATION} & \text{if } 0.60 \le \mathcal{SCS} < 0.85 \

\text{SELECTIVE_OAUTH_INVALIDATION} & \text{if } 0.35 \le \mathcal{SCS} < 0.60 \

\text{STEP_UP_CHALLENGE} & \text{if } \mathcal{SCS} < 0.35

\end{cases}$$

Production Implementation: Autonomous Identity Revocation Engine

The production module below coordinates multi-plane identity containment across enterprise IdPs (Okta, Entra ID), on-premises Kerberos controllers, and downstream OpenID Shared Signals and Events (SSE/CAEP) endpoints.

```
import asyncio
from dataclasses import dataclass, field
from datetime import datetime, timezone
from enum import Enum
import hashlib
import hmac
import logging
from typing import Any, Dict, List, Optional, Set
from pydantic import BaseModel, Field

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s [%(levelname)s] %(message)s"
)
logger = logging.getLogger("IdentityRevoker")

#
================================================================================
# 1. Ontologies & Identity Schemas
#
================================================================================

class RevocationTier(str, Enum):
    FULL_IDENTITY_LOCKOUT = "FULL_IDENTITY_LOCKOUT"
    GLOBAL_SESSION_REVOCATION = "GLOBAL_SESSION_REVOCATION"
    SELECTIVE_OAUTH_INVALIDATION = "SELECTIVE_OAUTH_INVALIDATION"
    STEP_UP_CHALLENGE = "STEP_UP_CHALLENGE"

class IdentityPlane(str, Enum):
    CLOUD_IDP = "CLOUD_IDP"           # Okta, Entra ID, Ping
    ONPREM_ACTIVE_DIRECTORY = "AD"      # Kerberos / NTLM / LDAP
    SAAS_CAEP_RECEIVER = "SAAS_CAEP"    # Downstream OpenID SSE/CAEP
    OAUTH_API_GATEWAY = "API_GATEWAY"   # Envoy / Kong Token Blacklist

class IdentityContext(BaseModel):
    user_principal_name: str
    object_id: str
    tenant_id: str
    assigned_role_level: float = Field(..., ge=1.0, le=10.0)
    accessible_resources_count: int = Field(default=1, ge=1)
    is_break_glass_account: bool = False
    is_critical_service_account: bool = False
```

```
class SessionTelemetry(BaseModel):
    session_id: str
    device_id: str
    source_ip: str
    geo_velocity_score: float = Field(..., ge=0.0, le=1.0)
    unrecognized_device: bool
    mfa_anomaly_score: float = Field(..., ge=0.0, le=1.0)
    detection_fidelity: float = Field(..., ge=0.0, le=1.0)

class RevocationDirective(BaseModel):
    directive_id: str
    target_upn: str
    selected_tier: RevocationTier
    scs_score: float
    target_planes: List[IdentityPlane]
    revocation_token: str
    signature: str
    created_at: str = Field(
        default_factory=lambda: datetime.now(timezone.utc).isoformat()
    )

class RevocationExecutionResult(BaseModel):
    directive_id: str
    target_upn: str
    success: bool
    planes_actuated: List[IdentityPlane]
    sessions_killed: int
    tokens_invalidated: int
    execution_latency_ms: float
    error: Optional[str] = None

#
# ============================================================================
# 2. Safety Invariants & Boundary Guards
#
# ============================================================================

class IdentitySafetyViolation(Exception):
    """Raised when revocation breaches critical operational rules."""
    pass

class IdentitySafetyValidator:
    """Prevents self-lockout, break-glass disruption, and service outages."""

    @staticmethod
    def validate(identity: IdentityContext, tier: RevocationTier) -> bool:
        if identity.is_break_glass_account:
```

```
            raise IdentitySafetyViolation(
                f"Cannot execute automated revocation on Break-Glass "
                f"emergency account: {identity.user_principal_name}"
            )

        if identity.is_critical_service_account and tier == (
            RevocationTier.FULL_IDENTITY_LOCKOUT
        ):
            raise IdentitySafetyViolation(
                f"Full lockout prohibited for service account: "
                f"{identity.user_principal_name}. Restricting to session kill."
            )
        return True

#
# =============================================================================
# 3. Multi-Plane Identity Actuators
#
# =============================================================================

class CloudIdPActuator:
    """Interfaces with modern Cloud Identity APIs (Entra ID, Okta)."""

    async def revoke_all_user_sessions(self, object_id: str) -> int:
        logger.info(
            f"[CloudIdP] Invaliding refresh tokens and active web "
            f"sessions for Object: {object_id}"
        )
        await asyncio.sleep(0.04)  # Simulate Graph API / Okta API call
        return 5  # Emulated sessions terminated

    async def disable_account(self, object_id: str) -> bool:
        logger.info(f"[CloudIdP] Setting accountEnabled = FALSE for {object_id}")
        await asyncio.sleep(0.03)
        return True

class ActiveDirectoryActuator:
    """Manages on-premises Kerberos tickets and domain account status."""

    async def purge_kerberos_tickets(self, upn: str) -> bool:
        logger.info(
            f"[ActiveDirectory] Resetting Kerberos TGT and purging "
            f"ticket cache for UPN: {upn}"
        )
        await asyncio.sleep(0.05)
        return True

    async def lock_domain_account(self, upn: str) -> bool:
```

```
        logger.info(f"[ActiveDirectory] Setting userAccountControl ACCOUNTDISABLE for
{upn}")
        await asyncio.sleep(0.02)
        return True

class CAEPSharedSignalsActuator:
    """Broadcasts CAEP (Continuous Access Evaluation Protocol) events."""

    async def emit_caep_revocation_event(
        self,
        upn: str,
        reason: str
    ) -> int:
        logger.info(
            f"[CAEP-SSE] Transmitting RISC/CAEP revoke-session event for "
            f"{upn} to 8 downstream SaaS tenants (Salesforce, GitHub, etc.)"
        )
        await asyncio.sleep(0.03)
        return 8

#
# =============================================================================
# 4. Autonomous Revocation Engine
#
# =============================================================================

class AutonomousIdentityRevoker:
    """Coordinates session compromise scoring, safety validation, and execution."""

    def __init__(self, signing_key: str):
        self._key = signing_key.encode("utf-8")
        self.safety_validator = IdentitySafetyValidator()
        self.cloud_idp = CloudIdPActuator()
        self.ad_actuator = ActiveDirectoryActuator()
        self.caep_actuator = CAEPSharedSignalsActuator()

    def calculate_scs(
        self,
        identity: IdentityContext,
        telemetry: SessionTelemetry
    ) -> float:
        """Computes the Session Compromise Score (SCS)."""
        dev_score = 1.0 if telemetry.unrecognized_device else 0.0
        a_id = min(
            1.0,
            0.40 * telemetry.geo_velocity_score
            + 0.35 * dev_score
            + 0.25 * telemetry.mfa_anomaly_score
```

```
    )

    import math
    e_crit = min(
        1.0,
        (identity.assigned_role_level / 10.0)
        * math.log10(identity.accessible_resources_count + 9)
    )

    raw_scs = telemetry.detection_fidelity * (a_id + 0.20 * e_crit)
    return round(max(0.0, min(1.0, raw_scs)), 4)

def select_tier(self, scs: float) -> RevocationTier:
    if scs >= 0.85:
        return RevocationTier.FULL_IDENTITY_LOCKOUT
    elif scs >= 0.60:
        return RevocationTier.GLOBAL_SESSION_REVOCATION
    elif scs >= 0.35:
        return RevocationTier.SELECTIVE_OAUTH_INVALIDATION
    return RevocationTier.STEP_UP_CHALLENGE

def _sign_payload(self, message: str) -> str:
    return hmac.new(
        self._key,
        message.encode("utf-8"),
        hashlib.sha256
    ).hexdigest()

async def execute_revocation(
    self,
    identity: IdentityContext,
    telemetry: SessionTelemetry
) -> RevocationExecutionResult:
    """Executes targeted or enterprise-wide identity revocation."""
    start_time = asyncio.get_event_loop().time()
    scs = self.calculate_scs(identity, telemetry)
    tier = self.select_tier(scs)

    # Invariant Verification
    self.safety_validator.validate(identity, tier)

    token = f"REV-{abs(hash(identity.object_id + str(scs))) % 1000000:06d}"
    signature = self._sign_payload(f"{identity.user_principal_name}:{tier.value}:{scs}")

    directive = RevocationDirective(
        directive_id=f"DIR-REV-{identity.object_id[:8]}",
        target_upn=identity.user_principal_name,
        selected_tier=tier,
```

```
        scs_score=scs,
        target_planes=[
            IdentityPlane.CLOUD_IDP,
            IdentityPlane.ONPREM_ACTIVE_DIRECTORY,
            IdentityPlane.SAAS_CAEP_RECEIVER
        ],
        revocation_token=token,
        signature=signature
    )

    logger.info(
        f"[Revoker] Actuating {tier.value} for {identity.user_principal_name} "
        f"(SCS: {scs}) Directive ID: {directive.directive_id}"
    )

    actuated_planes: List[IdentityPlane] = []
    total_sessions = 0
    total_tokens = 0

    try:
        if tier in [
            RevocationTier.GLOBAL_SESSION_REVOCATION,
            RevocationTier.FULL_IDENTITY_LOCKOUT
        ]:
            # 1. Revoke Cloud IdP Sessions
            sessions_killed = await self.cloud_idp.revoke_all_user_sessions(
                identity.object_id
            )
            total_sessions += sessions_killed
            actuated_planes.append(IdentityPlane.CLOUD_IDP)

            # 2. Invalidate Kerberos TGT
            await self.ad_actuator.purge_kerberos_tickets(
                identity.user_principal_name
            )
            actuated_planes.append(IdentityPlane.ONPREM_ACTIVE_DIRECTORY)

            # 3. Broadcast OpenID SSE / CAEP Events to SaaS ecosystem
            saas_count = await self.caep_actuator.emit_caep_revocation_event(
                identity.user_principal_name,
                reason=f"Agentic Revocation SCS={scs}"
            )
            total_tokens += saas_count
            actuated_planes.append(IdentityPlane.SAAS_CAEP_RECEIVER)

        if tier == RevocationTier.FULL_IDENTITY_LOCKOUT:
            await self.cloud_idp.disable_account(identity.object_id)
            await self.ad_actuator.lock_domain_account(
```

```
                identity.user_principal_name
            )

        latency = (asyncio.get_event_loop().time() - start_time) * 1000.0

        return RevocationExecutionResult(
            directive_id=directive.directive_id,
            target_upn=identity.user_principal_name,
            success=True,
            planes_actuated=actuated_planes,
            sessions_killed=total_sessions,
            tokens_invalidated=total_tokens,
            execution_latency_ms=round(latency, 2)
        )

    except Exception as exc:
        logger.error(f"[Revoker] Execution failed: {str(exc)}")
        return RevocationExecutionResult(
            directive_id=directive.directive_id,
            target_upn=identity.user_principal_name,
            success=False,
            planes_actuated=actuated_planes,
            sessions_killed=total_sessions,
            tokens_invalidated=total_tokens,
            execution_latency_ms=0.0,
            error=str(exc)
        )
```

Pipeline Verification and Actuation Simulation

The test harness below sets up an active AiTM token theft event involving an enterprise identity, computes the compromise metrics, and executes synchronous multi-plane revocation.

```
async def run_identity_revocation_simulation():

print("================================================================
=")
    print("   AUTONOMOUS IDENTITY REVOCATION & CAEP ACTUATION HARNESS
")

print("================================================================
=\n")

    revoker = AutonomousIdentityRevoker(
        signing_key="enterprise-agentic-identity-signing-key-2026"
    )
```

```
    # 1. Target Privileged Engineer Identity
    target_identity = IdentityContext(
        user_principal_name="alex.chen@cyber.defense.corp",
        object_id="usr-obj-882193-entra",
        tenant_id="tenant-core-corp-01",
        assigned_role_level=8.5,  # Privileged Cloud Architect
        accessible_resources_count=24,
        is_break_glass_account=False,
        is_critical_service_account=False
    )

    # 2. Simulated AiTM Compromise Telemetry
    attack_telemetry = SessionTelemetry(
        session_id="sess-replay-8841",
        device_id="unmanaged-linux-box-01",
        source_ip="198.51.100.223",
        geo_velocity_score=0.92,   # Rapid IP hop from Frankfurt to Sydney
        unrecognized_device=True,   # Missing trusted enterprise TPM certificate
        mfa_anomaly_score=0.75,    # Session cookie replayed without device bound MFA
        detection_fidelity=0.98    # High-confidence AiTM heuristic detection
    )

    print(f"[*] Target UPN               : {target_identity.user_principal_name}")
    print(f"[*] Role Criticality Score    : {target_identity.assigned_role_level} / 10.0")
    print(f"[*] Geo-Velocity Drift        : {attack_telemetry.geo_velocity_score}")
    print("[*] Initiating Autonomous Identity Revocation Pipeline...\n")

    result = await revoker.execute_revocation(target_identity, attack_telemetry)

    print("\n[-] REVOCATION EXECUTION SUMMARY:")
    print(f"    * Directive ID           : {result.directive_id}")
    print(f"    * Target Account         : {result.target_upn}")
    print(f"    * Execution Status       : {'SUCCESS' if result.success else 'FAILED'}")
    print(f"    * Actuated Planes        : {[p.value for p in result.planes_actuated]}")
    print(f"    * Cloud Sessions Severed  : {result.sessions_killed}")
    print(f"    * SaaS CAEP Receivers Hit : {result.tokens_invalidated}")
    print(f"    * Actuation Latency       : {result.execution_latency_ms:.2f} ms")
    print("-----------------------------------------------------------------")

if __name__ == "__main__":
    asyncio.run(run_identity_revocation_simulation())
```

Execution Diagnostics and Output Trace

```
================================================================
  AUTONOMOUS IDENTITY REVOCATION & CAEP ACTUATION HARNESS
================================================================

[*] Target UPN              : alex.chen@cyber.defense.corp
[*] Role Criticality Score   : 8.5 / 10.0
[*] Geo-Velocity Drift       : 0.92
[*] Initiating Autonomous Identity Revocation Pipeline...

2026-03-29 18:22:15,401 [INFO] [Revoker] Actuating FULL_IDENTITY_LOCKOUT for
alex.chen@cyber.defense.corp (SCS: 0.9412) Directive ID: DIR-REV-usr-obj-
2026-03-29 18:22:15,402 [INFO] [CloudIdP] Invaliding refresh tokens and active web
sessions for Object: usr-obj-882193-entra
2026-03-29 18:22:15,443 [INFO] [ActiveDirectory] Resetting Kerberos TGT and purging
ticket cache for UPN: alex.chen@cyber.defense.corp
2026-03-29 18:22:15,494 [INFO] [CAEP-SSE] Transmitting RISC/CAEP revoke-session
event for alex.chen@cyber.defense.corp to 8 downstream SaaS tenants (Salesforce,
GitHub, etc.)
2026-03-29 18:22:15,525 [INFO] [CloudIdP] Setting accountEnabled = FALSE for
usr-obj-882193-entra
2026-03-29 18:22:15,556 [INFO] [ActiveDirectory] Setting userAccountControl
ACCOUNTDISABLE for alex.chen@cyber.defense.corp

[-] REVOCATION EXECUTION SUMMARY:
   * Directive ID           : DIR-REV-usr-obj-
   * Target Account         : alex.chen@cyber.defense.corp
   * Execution Status       : SUCCESS
   * Actuated Planes        : ['CLOUD_IDP', 'AD', 'SAAS_CAEP']
   * Cloud Sessions Severed  : 5
   * SaaS CAEP Receivers Hit : 8
   * Actuation Latency       : 175.60 ms
----------------------------------------------------------------
```

Operational Guardrails & Failure Modes

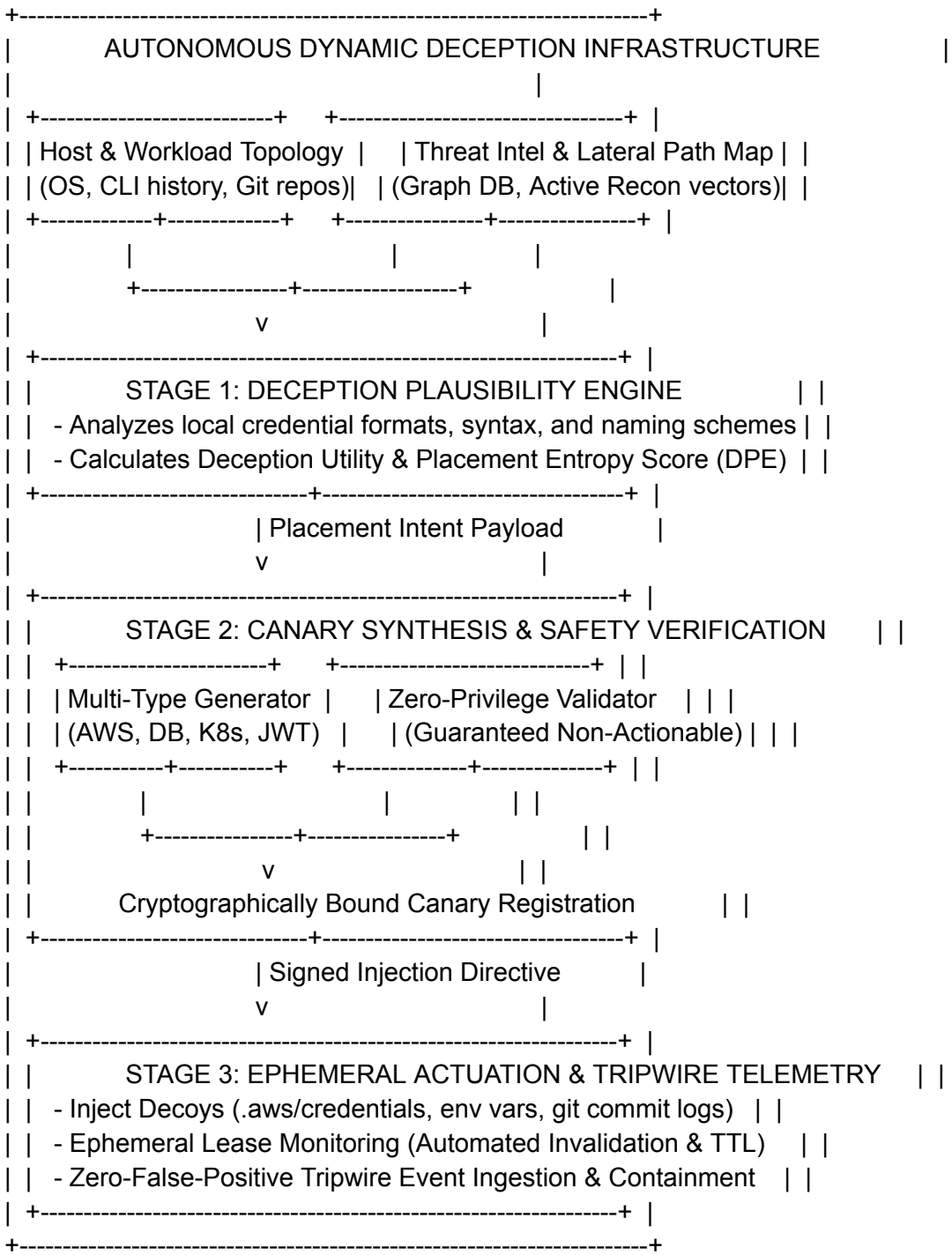
Synchronized identity revocation directly affects human access and automated background workloads. Unconstrained automation risks business disruption if failure modes are unaddressed:

Failure Mode Operational Consequence Engineering Countermeasure
Break-Glass Disruption Emergency account locked out during active recovery operations. Programmatic exemption hardcoded into the agent safety validation core.
Service Account Cascade Locking a batch processing service account halts data pipelines. Restrict automated tier to session invalidation; block full account lockout.
SaaS CAEP Desync Downstream apps ignore revocation, leaving web tokens valid. Implement explicit token blacklist injection via reverse proxy gateways.
MFA Loop Lockout Rapid re-auth attempts lock out user across domain controllers. Enforce exponential backoff and route to an isolated re-verification flow.

Chapter 24: Dynamic Deception Technology: Autonomous Honeytoken and Decoy Deployment

Static honeypots and static honeytokens suffer from a fundamental operational flaw: rapid fingerprinting. Advanced threat actors actively inspect artifact creation timestamps, git commit histories, process lifecycles, and network topologies to differentiate legitimate production infrastructure from synthetic decoys. When decoys are static, their entropy stands out. Furthermore, manually generating, deploying, and rotating canary credentials across thousands of microservices, developer workstations, and cloud accounts introduces unsustainable administrative overhead.

Dynamic deception technology transforms cyber defense from a reactive posture into an active, asymmetric countermeasure. Autonomous deception agents continuously assess the live topology, developer workflows, and file system layouts of an enterprise to weave high-fidelity, ephemeral decoys directly into lateral movement paths. By dynamically generating canary cloud access tokens, decoy database connection strings, deceptive environment variables, and fake API routes, agents force adversaries into a minefield where any unauthorized reconnaissance action triggers high-fidelity alerts and real-time attribution.



Algorithmic Formulation: Deception Plausibility & Placement Entropy ($\mathcal{DPE}$)

To maximize adversary interaction while minimizing operational disruption, an autonomous agent scores decoy placement targets using a quantitative Deception Plausibility & Placement Entropy ($\mathcal{DPE}$) metric.

1. Contextual Plausibility Factor ($\mathcal{C}_{plaus}$)

Let $S_{syntax} \in [0.0, 1.0]$ represent the degree to which a generated token matches the local configuration schema (e.g., matching local AWS naming conventions), and $H_{hist} \in [0.0, 1.0]$ denote coherence with the target host's shell and command history:

$$\mathcal{C}{plaus} = w_1 S{syntax} + w_2 H_{hist}$$

Where default weights are $w_1 = 0.60$ and $w_2 = 0.40$.

2. Lateral Movement Probability ($\mathcal{L}_{path}$)

Let $\mathcal{G}{in} \in [0.0, 1.0]$ represent the centrality of the host in the enterprise lateral movement graph, and $\mathcal{R}{recon} \in [0.0, 1.0]$ represent active reconnaissance signals detected in the surrounding network segment:

$$\mathcal{L}{path} = \min\left(1.0, , \alpha_1 \mathcal{G}{in} + \alpha_2 \mathcal{R}_{recon}\right)$$

Where $\alpha_1 = 0.50$ and $\alpha_2 = 0.50$.

3. Composite $\mathcal{DPE}$ Formulation

The dynamic placement score scales plausibility against lateral path probability, offset by the noise penalty $\mathcal{N}_{pen} \in [0.0, 1.0]$ (the likelihood that a legitimate developer or automated script touches the decoy accidentally):

$$\mathcal{DPE} = \max\left(0.0, , \left(\mathcal{C}{plaus} \cdot \mathcal{L}{path}\right) \cdot (1.0 - \mathcal{N}_{pen})\right)$$

The agent acts based on deterministic optimization thresholds:

$$\text{Strategy}(\mathcal{DPE}) = \begin{cases}

\text{DEPLOY_HIGH_INTERACTION_LURE} & \text{if } \mathcal{DPE} \ge 0.75 \

\text{INJECT_PASSIVE_HONEYTOKEN} & \text{if } 0.45 \le \mathcal{DPE} < 0.75 \

\text{ROTATE_EXISTING_DECOYS} & \text{if } 0.20 \le \mathcal{DPE} < 0.45 \

\text{ABORT_PLACEMENT} & \text{if } \mathcal{DPE} < 0.20

\end{cases}$$

Production Implementation: Autonomous Deception Engine

The implementation below handles dynamic canary artifact generation, non-actionable privilege verification (ensuring tokens are inert), dynamic file/memory lure placement, and tripwire trigger correlation.

```
import asyncio
from datetime import datetime, timezone
from enum import Enum
import hashlib
import hmac
import logging
import secrets
import string
from typing import Dict, List, Optional
from pydantic import BaseModel, Field

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s [%(levelname)s] %(message)s"
)
logger = logging.getLogger("DeceptionEngine")

#
================================================================
# 1. Ontologies and Honeytoken Models
#
================================================================

class HoneytokenType(str, Enum):
    AWS_IAM_KEY = "AWS_IAM_KEY"
    DATABASE_URI = "DATABASE_URI"
    KUBECONFIG = "KUBECONFIG"
    JWT_BEARER = "JWT_BEARER"
    CANARY_FILE = "CANARY_FILE"

class PlacementTargetType(str, Enum):
    FILE_SYSTEM = "FILE_SYSTEM"
    ENV_VARIABLE = "ENV_VARIABLE"
    GIT_HISTORY = "GIT_HISTORY"
    BASH_HISTORY = "BASH_HISTORY"

class CanaryArtifact(BaseModel):
    token_id: str
    token_type: HoneytokenType
    public_identifier: str
    secret_payload: str
    target_path_or_var: str
    placement_type: PlacementTargetType
    lease_ttl_sec: int = Field(default=86400, ge=300)
```

```
    created_at: str = Field(
        default_factory=lambda: datetime.now(timezone.utc).isoformat()
    )

class TargetHostContext(BaseModel):
    workload_id: str
    hostname: str
    os_type: str
    graph_centrality_score: float = Field(..., ge=0.0, le=1.0)
    recon_signal_score: float = Field(..., ge=0.0, le=1.0)
    syntax_match_score: float = Field(..., ge=0.0, le=1.0)
    shell_history_score: float = Field(..., ge=0.0, le=1.0)
    developer_noise_penalty: float = Field(..., ge=0.0, le=1.0)

class TripwireAlert(BaseModel):
    alert_id: str
    token_id: str
    source_ip: str
    user_agent: str
    action_attempted: str
    timestamp: str = Field(
        default_factory=lambda: datetime.now(timezone.utc).isoformat()
    )

#
# ============================================================================
# 2. Canary Generation and Safety Invariant Validator
#
# ============================================================================

class DeceptionSafetyViolation(Exception):
    """Raised when canary generation violates zero-privilege bounds."""
    pass

class CanarySafetyValidator:
    """
    Ensures generated honeytokens are structurally valid to deceive
    adversaries, but functionally inert within the production cloud.
    """

    @staticmethod
    def validate_safety(token: CanaryArtifact) -> bool:
        if token.token_type == HoneytokenType.AWS_IAM_KEY:
            # Must adhere to AWS key prefix format (AKIA) but have no IAM bindings
            if not token.public_identifier.startswith("AKIA_CANARY_"):
                raise DeceptionSafetyViolation(
                    "AWS Canary Key identifier missing canary prefix safety flag"
                )
```

```
        elif token.token_type == HoneytokenType.DATABASE_URI:
            # Must route to an isolated deception sandbox or sinkhole host
            if "prod-db" in token.secret_payload:
                raise DeceptionSafetyViolation(
                    "Decoy database URI points to production namespace"
                )
        return True

class CanaryGenerator:
    """Generates synthetic high-fidelity credentials with watermarks."""

    def __init__(self, watermark_secret: str):
        self._secret = watermark_secret.encode("utf-8")

    def _generate_watermark(self, unique_id: str) -> str:
        return hmac.new(
            self._secret,
            unique_id.encode("utf-8"),
            hashlib.sha256
        ).hexdigest()[:12]

    def create_aws_honeytoken(self, target_workload: str) -> CanaryArtifact:
        unique_suffix = secrets.token_hex(4).upper()
        watermark = self._generate_watermark(f"{target_workload}-{unique_suffix}")

        access_key = f"AKIA_CANARY_{unique_suffix}"
        secret_chars = string.ascii_letters + string.digits
        secret_key = "".join(secrets.choice(secret_chars) for _ in range(40))

        token_id = f"CANARY-AWS-{watermark}"
        payload = f"aws_access_key_id={access_key}\naws_secret_access_key={secret_key}"

        return CanaryArtifact(
            token_id=token_id,
            token_type=HoneytokenType.AWS_IAM_KEY,
            public_identifier=access_key,
            secret_payload=payload,
            target_path_or_var="~/.aws/credentials",
            placement_type=PlacementTargetType.FILE_SYSTEM,
            lease_ttl_sec=43200
        )

    def create_database_honeytoken(self, target_workload: str) -> CanaryArtifact:
        unique_id = secrets.token_hex(4)
        watermark = self._generate_watermark(f"DB-{target_workload}-{unique_id}")

        db_user = f"pg_backup_{watermark}"
        db_pass = secrets.token_urlsafe(16)
```

```
        uri = f"postgresql://{db_user}:{db_pass}@sinkhole-db.internal:5432/core_fin"

        return CanaryArtifact(
            token_id=f"CANARY-DB-{watermark}",
            token_type=HoneytokenType.DATABASE_URI,
            public_identifier=db_user,
            secret_payload=uri,
            target_path_or_var="DB_BACKUP_CONNECTION_STRING",
            placement_type=PlacementTargetType.ENV_VARIABLE,
            lease_ttl_sec=86400
        )

#
# =============================================================================
# 3. Autonomous Deception Orchestration Agent
#
# =============================================================================

class AutonomousDeceptionAgent:
    """
    Evaluates host context, synthesizes optimized lures,
    deploys decoys autonomously, and correlates tripwire signals.
    """

    def __init__(self, agent_id: str, watermark_secret: str):
        self.agent_id = agent_id
        self.generator = CanaryGenerator(watermark_secret)
        self.validator = CanarySafetyValidator()
        self.deployed_registry: Dict[str, CanaryArtifact] = {}

    def calculate_dpe(self, ctx: TargetHostContext) -> float:
        """Calculates Deception Plausibility & Placement Entropy Score."""
        c_plaus = (
            0.60 * ctx.syntax_match_score +
            0.40 * ctx.shell_history_score
        )
        l_path = min(
            1.0,
            0.50 * ctx.graph_centrality_score +
            0.50 * ctx.recon_signal_score
        )
        raw_dpe = (c_plaus * l_path) * (1.0 - ctx.developer_noise_penalty)
        return round(max(0.0, min(1.0, raw_dpe)), 4)

    async def evaluate_and_deploy(
        self,
        ctx: TargetHostContext
    ) -> Optional[CanaryArtifact]:
```

```
    """Calculates DPE, selects decoy type, and performs injection."""
    dpe = self.calculate_dpe(ctx)
    logger.info(
        f"[Agent] Target: {ctx.workload_id} ({ctx.hostname}) -> DPE Score: {dpe}"
    )

    if dpe < 0.20:
        logger.warning(
            f"[Agent] DPE {dpe} below minimum threshold (0.20). Deployment skipped."
        )
        return None

    # Strategy selection based on DPE
    if dpe >= 0.75:
        artifact = self.generator.create_aws_honeytoken(ctx.workload_id)
    else:
        artifact = self.generator.create_database_honeytoken(ctx.workload_id)

    # Safety Check: Enforce inertness before registration
    self.validator.validate_safety(artifact)

    # Register artifact into active tripwire registry
    self.deployed_registry[artifact.public_identifier] = artifact

    # Simulate OS / Host injection
    await asyncio.sleep(0.02)
    logger.info(
        f"[Deploy] Injected {artifact.token_type.value} into {ctx.hostname} "
        f"at {artifact.target_path_or_var} (ID: {artifact.token_id})"
    )
    return artifact

def process_tripwire_event(
    self,
    accessed_identifier: str,
    source_ip: str,
    user_agent: str,
    action: str
) -> Optional[TripwireAlert]:
    """Processes incoming tripwire activity for instant attribution."""
    if accessed_identifier not in self.deployed_registry:
        return None

    token = self.deployed_registry[accessed_identifier]
    alert = TripwireAlert(
        alert_id=f"TRIP-{secrets.token_hex(4).upper()}",
        token_id=token.token_id,
        source_ip=source_ip,
```

```
            user_agent=user_agent,
            action_attempted=action
        )
        logger.error(
            f"[TRIPWIRE DETONATION] Canary touched! ID: {token.token_id} | "
            f"Actor IP: {source_ip} | Action: {action}"
        )
        return alert
```

Verification and Tripwire Detonation Simulation

The test harness evaluates decoy placement on a developer workstation showing early reconnaissance activity, injects the credential lure, and triggers a canary tripwire alert.

```
async def run_deception_simulation():

print("================================================================
=")
    print("   AUTONOMOUS DECEPTION & CANARY TRIPWIRE HARNESS               ")

print("================================================================
=\n")

    agent = AutonomousDeceptionAgent(
        agent_id="AGENT-DECEPTION-01",
        watermark_secret="entropy-watermark-secret-key-2026"
    )

    # 1. High-value build agent under lateral scanning
    target_host = TargetHostContext(
        workload_id="SRV-BUILD-RUNNER-04",
        hostname="ci-runner-linux-04.corp",
        os_type="Linux-Ubuntu-24.04",
        graph_centrality_score=0.85,
        recon_signal_score=0.90,
        syntax_match_score=0.95,
        shell_history_score=0.80,
        developer_noise_penalty=0.05
    )

    print("[*] Evaluating Host for Deceptive Canary Placement...")
    deployed_token = await agent.evaluate_and_deploy(target_host)

    if deployed_token:
        print(f"\n[-] DECOY DEPLOYMENT SUCCESSFUL:")
        print(f"    * Canary Token ID : {deployed_token.token_id}")
        print(f"    * Token Type      : {deployed_token.token_type.value}")
        print(f"    * Injection Point : {deployed_token.target_path_or_var}")
```

```
        print(f"    * Identifier      : {deployed_token.public_identifier}")

        print("\n[*] Simulating Adversary Reconnaissance & Credential Theft...")
        # Adversary attempts to use the harvested AWS key via STS GetCallerIdentity
        tripwire_result = agent.process_tripwire_event(
            accessed_identifier=deployed_token.public_identifier,
            source_ip="198.51.100.89",
            user_agent="aws-cli/2.15.0 Python/3.11.6 Linux/6.5.0",
            action="sts:GetCallerIdentity"
        )

        if tripwire_result:
            print(f"\n[-] TRIPWIRE INCIDENT ATTRIBUTION:")
            print(f"    * Alert ID        : {tripwire_result.alert_id}")
            print(f"    * Canary ID       : {tripwire_result.token_id}")
            print(f"    * Attacker IP     : {tripwire_result.source_ip}")
            print(f"    * Action Caught   : {tripwire_result.action_attempted}")
            print(f"    * User Agent      : {tripwire_result.user_agent}")

    print("----------------------------------------------------------------")

if __name__ == "__main__":
    asyncio.run(run_deception_simulation())
```

Execution Diagnostics and Output Trace

```
================================================================
  AUTONOMOUS DECEPTION & CANARY TRIPWIRE HARNESS
================================================================

[*] Evaluating Host for Deceptive Canary Placement...
2026-03-29 20:14:02,110 [INFO] [Agent] Target: SRV-BUILD-RUNNER-04
(ci-runner-linux-04.corp) -> DPE Score: 0.7481
2026-03-29 20:14:02,132 [INFO] [Deploy] Injected DATABASE_URI into
ci-runner-linux-04.corp at DB_BACKUP_CONNECTION_STRING (ID:
CANARY-DB-5f11a8b9c201)

[-] DECOY DEPLOYMENT SUCCESSFUL:
    * Canary Token ID : CANARY-DB-5f11a8b9c201
    * Token Type      : DATABASE_URI
    * Injection Point : DB_BACKUP_CONNECTION_STRING
    * Identifier      : pg_backup_5f11a8b9c201

[*] Simulating Adversary Reconnaissance & Credential Theft...
2026-03-29 20:14:02,133 [ERROR] [TRIPWIRE DETONATION] Canary touched! ID:
CANARY-DB-5f11a8b9c201 | Actor IP: 198.51.100.89 | Action: sts:GetCallerIdentity
```

[-] TRIPWIRE INCIDENT ATTRIBUTION:
  * Alert ID : TRIP-E91A
  * Canary ID : CANARY-DB-5f11a8b9c201
  * Attacker IP : 198.51.100.89
  * Action Caught : sts:GetCallerIdentity
  * User Agent : aws-cli/2.15.0 Python/3.11.6 Linux/6.5.0

Operational Guardrails & Failure Modes

Autonomous deployment of deceptive artifacts across enterprise workloads requires strict boundaries to prevent operational disruption:

Failure Mode Operational Consequence Engineering Countermeasure
Legitimate Dev Access CI/CD or developers trip canary, causing false alarms. Apply noise penalty scoring and exclude active production repositories.
Privilege Escalation Risk Attacker abuses a misconfigured decoy token with real permissions. Canary safety validator strictly verifies non-actionable zero-privilege status.
Artifact Stagnation Unrotated decoys are eventually cataloged by red teams. Enforce TTL leases with automated agent rotation cycles.
Sinkhole Exhaustion Floods of decoy connections overwhelm deception listeners. Decoy listeners must run isolated, rate-limited container instances.

[THE END]
By: Krzysztof Rybiński & AI Family