Antinuke (CUT) – Dokument techniczny v1.0

Wygenerowano: 2025-09-09 19:22:08

Antinuke (CUT) – Dokument techniczny v1.0

Cel: Kompletny, wysokowydajny i bezpieczny system Antinu (pe■ne, natychmiastowe odci■cie uprawnie■ zamiast bana), cil automatycznym przywracaniem infrastruktury serwera po ataku

---

1. Streszczenie wykonawcze

- **Strategia**: Zero-trust, agresywna polityka **CUT** + automatyczny **rollback/restore**.
- **Architektura**: Hybryda **Rust (core egzekucyjny)** + **TypeScript/Node (komendy i panel)**, z **Redis/PG/S3**.
- **G■ówne funkcje**: detekcja bursta akcji destrukcyjnych, odci■cie sprawcy, globalne blokady, snapshoty delta, auto-restore, forensics, 2■man approvals, tryb maintenance.
- **SLO**: TTA (time■to■arrest) ≤ 150 ms p99 dla ■cie■ki CUT; RPO backupów na poziomie sekund (continuous);

RTO restore do spójnego stanu ≤ 60 s dla ■redniej wielko■ci serwera.

---

2. S■ownik poj■■

- **CUT** – natychmiastowe, pe■ne odci■cie uprawnie■ sprawcy (rola Quarantine, removal perms, timeout), bez bana.
- **Allowlista** – zestaw zaufanych kont/ ról dopuszczonych do wykonywania krytycznych akcji (z regu■■ M■of■N).
- **Snapshot** – pe■ny zrzut stanu (role, kana■y, overwrites, ustawienia, emoji, webhooki).
- **Delta/Journal** – przyrostowy dziennik pojedynczych zmian struktury (event■driven).
- **ELEVATED/ATTACK** – stany podwy■szonego ryzyka/aktywnego incydentu.

---

3. Wymagania

3.1 Funkcjonalne

- Detekcja masowych/niebezpiecznych akcji (role, kana■y, bany, webhooki, emoji, ustawienia).
- Akcja **CUT** + globalne "lock switches" (ban/kick, manage webhooks, itp.).
- Auto■rollback/restore infrastruktury po incydencie.
- 2■man approvals dla operacji destrukcyjnych poza incydentem.
- Tryb **maintenance** (podwy■szone progi, tylko logi).
- Panel www (alerty, timeline, przyciski Panic/Approve/Restore, edycja polityk).

3.2 Niefunkcjonalne

- Opó■nienie ■cie■ki CUT ≤ 150 ms p99; deterministyczne czasy.
- Odporno■■ na rate■limity Discorda (kolejki priorytetowe, backoff, idempotencja).
- Bezpiecze■stwo: mTLS mi■dzy komponentami, rotacja sekretów, 2FA dla personelu.
- Audytowalno■■ (hashy incydentów, logi nieusuwalne, eksport do S3/SIEM).

---

4. Polityka bezpiecze■stwa (CUT)

- **Zasada**: Pierwsze trafienie w regu■■ krytyczn■ ⇒ automatyczne **CUT** (bez bana) i

natychmiastowy rollback zmian.
- **Quarantine role**: Rola systemowa z deny na *wszystkie* krytyczne permsy oraz overwrites globalne (deny Send/Connect), z jednym kana■em „#appeals”.
- **Global locks**: Tymczasowe odebranie `Ban/Kick`, `Manage Webhooks`, `Manage Roles/Channels` poza allowlist■.
- **Timeout**: 60 min dla kont ludzkich (boty – bez timeoutu, tylko izolacja).
- **Owner edge■case**: brak mo■liwo■ci odebrania w■a■cicielowi; zamiast tego PANIC (global locks + restore) i eskalacja.

---

5. Scenariusze i stany

- **NORMAL** → progi standardowe, shadow alerts opcjonalnie.
- **ELEVATED** → wykryto anomalie niskiego/■redniego ryzyka; wzmo■one logi, opcjonalne mi■kkie lockdowny.
- **ATTACK** → regu■a krytyczna trafiona; **CUT**, PANIC, auto■restore, powiadomienia.

---

6. Architektura systemu

```
[Discord Gateway/HTTP]
■
▼
Rust Core (twilight + tokio)
■■ Event Ingest & Audit Sync
■■ Rules Engine (windows, z■score, honeypots)
■■ CUT Executor (quarantine, locks, timeout)
■■ Snapshot Manager (full + delta)
■■ Restore Orchestrator (diff, ID remap, rate queues)
■■ Forensics & Incident Journal
■■ Admin API (axum/tonic, gRPC/HTTP, SSE)
■■ Policy Runtime (WASM, opcjonalne)
■ ▲
■ ■ gRPC/HTTP (mTLS)
▼ ■
Edge/Commands (Node)
■■ discord.js (slash)
■■ BFF API (Fastify/NestJS)
■■ Next.js Panel (UI)
Stores: Redis (counters/locks), PostgreSQL (config/incidents), S3 (snapshots)
Obs: tracing + Prometheus + OpenTelemetry
```

---

7. Komponenty (szczegó■y)

7.1 Rust Core

- **Gateway**: `twilight-gateway` (shardy), filtr zdarze■, backpressure.
- **Rules Engine**: sliding windows (5–120 s), per■user/guild/global liczniki w Redis; regu■y kompozycyjne (AND/OR), severity, akcje.
- **CUT Executor**: atomowe sekwencje: lock → attach quarantine → remove risky roles → global locks → overwrites → timeout → log.
- **Snapshot Manager**: full snapshot co 20 min + continuous journal (debounce 2 s); podpisy hash.
- **Restore Orchestrator**: diff snapshot+Δ vs live; fazy: roles → channels → overwrites → assets → settings → (opcjonalnie) member■roles.
- **Admin API**:

- gRPC: `Cut`, `Restore`, `Approve`, `SetPolicy`, `StreamEvents`.
- HTTP: `/health`, `/metrics`, `/incidents`, SSE `/events`.
- **Policy Runtime (WASM)**: ■adowanie regu■ z polityki, hot■swap bez restartu core.

7.2 Edge/Commands (Node)

- **discord.js**: slashy (`/panic`, `/approve`, `/restore`, `/simulate`, `/policy get/set`, `/maintenance start/stop`).
- **BFF API**: autoryzacja Discord OAuth2 (panel), RBAC (Security/Owner), proxy do gRPC.
- **Next.js Panel**: alerty live (SSE), timeline incydentu, edycja polityk (schema■validated), przyciski akcji.

7.3 Bazy i kolejki

- **Redis**: klucze liczników, locki rozproszone (mutex CUT), pub/sub dla UI.
- **PostgreSQL**: konfiguracje, incydenty, approvals, artefakty forensics, health.
- **S3**: snapshoty i dzienniki delty (JSON, podpisy, wersjonowanie).

---

8. Przep■ywy krytyczne

8.1 ■cie■ka CUT (pseudosekwencja)

1) Wykrycie regu■y krytycznej → `acquireGuildLock(CUT)`.

2) Dodaj rol■ `@Quarantined` (wysoko), zdejmij role z permsami ryzyka.

3) W■■cz globalne locki (ban/kick/webhook/manage roles/channels poza allowlis

4) Timeout (je■li user, nie bot).

5) Wymu■ overwrites (deny Send/Connect) globalnie poza `#appeals`.

6) Uruchom restore (je■li by■o kasowanie/edycje struktury) → fazy 1–4.

7) Zaloguj incydent + wy■lij alerty (owner/SecTeam/SIEM).

8) `releaseGuildLock`.

8.2 Backup/Restore (continuous)

- Ka■dy event strukturalny → wpis do `journal` (z debouncem) + cykliczny `snapshot`.
- Restore planuje `diff`, remapuje ID, respektuje rate■limit Discorda, zapewnia idempotencj■.

8.3 Approvals (M■of■N)

- `request → hash(payload+ts)`; dwóch ró■nych cz■onków allowlisty `/approve` → wykonanie akcji.

---

9. Detekcje i regu■y (domy■lne)

| ID | Zdarzenie | Próg | Stan | Akcja |
|---|---|---|---|---|
| R1 | Role remove (staff) | ≥2/30 s | ATTACK | CUT + restore roles |
| R2 | Channel delete | ≥2/30 s | ATTACK | CUT + restore channels |
| R3 | Ban/kick wave | ≥3/30 s | ATTACK | CUT + lock ban/kick 15 min |
| R4 | Webhook storm | ≥2/30 s lub endpoint change | ATTACK | CUT + p
| R5 | Grant admin | ≥1 (bez approval) | ATTACK | CUT |

| R6 | Role position above bot | ≥1 | ATTACK | CUT |

| R7 | Emoji/sticker purge | ≥5/60 s | ATTACK | CUT + restore assets |

| R8 | Vanity/name/icon flip | ≥1 poza maintenance | ATTACK | CUT |

| R9 | Everyone mention spike | heurystyka | ELEVATED | mention governor / soft lock |

| R10| New bot gets manage* | ≥1 | ATTACK | CUT bot + sandbox |

Progi mo■na podnosi■ w maintenance; polityka jest edytowalna (policy■as■code).

---

10. Konfiguracja – schema (YAML)

```
mode: strict-cut | observe | custom
approvals:
m_of_n: [2, 3]
scope: [grant_admin, delete_channel>1, edit_role_high, webhook_change]
maintenance:
enabled: false
raise_thresholds_by: 3
honeypots:
roles: ["do-not-touch", "audit-anchor"]
channels: ["#__config", "#__vault"]
windows:
short: 30s
medium: 60s
long: 120s
rules:
role_remove_staff: { threshold: ">=2/30s", action: CUT }
channel_delete: { threshold: ">=2/30s", action: CUT+ROLLBACK }
ban_wave: { threshold: ">=3/30s", action: CUT+GLOBAL_LOCK(ban_kick,15m) }
webhook_storm: { threshold: ">=2/30s or endpoint_changed", action:
CUT+PURGE(webhooks_recent) }
grant_admin: { threshold: ">=1", require_approval: true, action_on_bypass: CUT }
role_position_above_bot: { threshold: ">=1", action: CUT }
emoji_purge: { threshold: ">=5/60s", action: CUT+RESTORE(emojis) }
vanity_or_icon_flip: { threshold: ">=1", action: CUT }
cut_action:
quarantine_role: "@Quarantined"
timeout: 60m
global_locks: ["ban_kick", "manage_webhooks"]
channel_overwrites:
deny_all_but: ["#appeals"]
backups:
mode: continuous
snapshot_interval: 20m
debounce_window: 2s
store: { s3: true, path: s3://antinuke/{guild_id}/{date}/ }
track_member_roles: true
retention: 72h
restore:
auto_on_attack: true
restore_member_roles: last_15m
verify_positions: strict
rate_limit: { create_channel: 8/s, edit_overwrites: 15/s }
logging:
security_channel: "#security-log"
dm_owner: true
evidence_export: true
```

---

11. API (gRPC/HTTP)

11.1 Protobuf (skrót)

```
service Antinuke {
rpc Cut(CutRequest) returns (CutResult);
rpc Restore(RestoreRequest) returns (RestoreResult);
rpc Approve(ApproveRequest) returns (ApproveResult);
rpc SetPolicy(SetPolicyRequest) returns (PolicyVersion);
rpc StreamEvents(StreamRequest) returns (stream SecurityEvent);
}
message CutRequest { string guild_id=1; string offender_id=2; string reason=3; }
message CutResult { string incident_id=1; bool executed=2; }
```

11.2 HTTP (Admin API)

- `GET /health` → 200 OK.
- `GET /metrics` → Prometheus.
- `GET /incidents?guild_id=...` → lista.
- `POST /restore` { guild_id, scope?, since? }.
- `GET /events` (SSE) → strumie■ SecurityEvent.

---

12. Model danych (PostgreSQL)

```
-- Gildie i polityki
CREATE TABLE guilds(
guild_id TEXT PRIMARY KEY,
name TEXT, policy JSONB, policy_version TEXT,
created_at TIMESTAMPTZ DEFAULT now()
);
-- Incydenty i zdarzenia
CREATE TABLE incidents(
id UUID PRIMARY KEY, guild_id TEXT, started_at TIMESTAMPTZ,
status TEXT, offender_id TEXT, reason TEXT, severity TEXT,
evidence_hash TEXT
);
CREATE TABLE incident_events(
id BIGSERIAL PRIMARY KEY, incident_id UUID,
ts TIMESTAMPTZ, kind TEXT, payload JSONB
);
-- Snapshoty
CREATE TABLE snapshots(
id UUID PRIMARY KEY, guild_id TEXT, ts TIMESTAMPTZ,
version TEXT, s3_url TEXT, sha256 TEXT
);
CREATE TABLE deltas(
id BIGSERIAL PRIMARY KEY, guild_id TEXT, ts TIMESTAMPTZ,
kind TEXT, payload JSONB, sha256 TEXT
);
-- Approvals
CREATE TABLE approvals(
id UUID PRIMARY KEY, guild_id TEXT, action_hash TEXT,
approver_id TEXT, ts TIMESTAMPTZ
);
```

12.1 Redis – klucze

- `cnt:{guild}:{user}:{event}` → licznik + TTL (okno).
- `lock:{guild}:CUT` → mutex.

- `map:oldid:newid:{guild}` → mapowanie ID przy restore.
- `feed:{guild}` → pub/sub alertów.

---

13. Snapshot – schema (JSON)

```
{
"version": "v1",
"guild": {"id": "...", "name": "...", "settings": {"mfa_level": 2,
"verification_level": 3}},
"roles": [{"id": "123", "name": "Admin", "position": 18, "permissions": "...",
"color": 0}],
"channels": [{
"id": "456", "name": "general", "type": "GUILD_TEXT", "parent_id": "cat1",
"overwrites": [{"type": "role", "id": "123", "allow": "...", "deny": "..."}]
}],
"categories": [{"id": "cat1", "name": "Public"}],
"emoji": [{"id": "e1", "name": "smile", "hash": "..."}],
"webhooks": [{"id": "w1", "channel_id": "456", "name": "build"}],
"vanity": {"code": "myserver"}
}
```

---

14. Algorytmy

14.1 Diff/Restore

1. Za■aduj `snapshot_latest` + replay `delta` do czasu T■ε.

2. Zbuduj graf zale■no■ci (kategorie → kana■y → overwrites; role → pozycje → c

3. Wyznacz brakuj■ce/zmienione elementy.

4. **Faza 1: utwórz role w kolejno■ci pozycji; zbuduj map■ `old→new`.**

5. **Faza 2: odtwórz kategorie/kana■y; stosuj overwrites po zmapowaniu ID.**

6. **Faza 3: odtwórz emoji/webhooki; ustawienia gildii.**

7. **Faza 4: (opcjonalnie) przypisania ról cz■onkom (ca■o■■/ostatnie 15 min).**

8. Weryfikacja: liczno■ci i checksumy.

14.2 Rate■limit & Idempotencja

- Kolejki z priorytetem: `CUT > restore > normal`.
- Token bucket per■route Discorda; backoff na 429 (exponential + jitter).
- Idempotency key: `cut:{guild}:{offender}:{tsWindow}` i `restore:{guild}:{snapshotId}`.

---

15. Bezpiecze■stwo operacyjne

- mTLS (rustls) mi■dzy komponentami; certyfikaty rotowane.
- Sekrety w SOPS/Vault; minimalny zakres tokenów.
- RBAC w panelu: Owner, SecTeam, Observer.
- 2FA wymagana dla ról moderacyjnych na serwerze.

---

16. Observability

- `tracing` z korrelacj■ `incident_id`.

- Prometheus: metryki m.in. `cut_latency`, `restore_rto`, `discord_429`, `rules_fired_total`.
- OpenTelemetry → Jaeger/Tempo (trace'y end■to■end).

---

17. Testy i symulacje

- Jednostkowe (regu■y, diff), property■based (`proptest`).
- Integracyjne (fake Discord HTTP), chaos tests (rate■limit storms).
- Komenda `/simulate nuke` – generuje burst zdarze■ i raportuje TTA/RTO.

---

18. Deployment

- Docker/Compose na start; Helm chart (K8s) produkcyjnie.
- Sharding gateway (twilight) wg wielko■ci serwerów.
- Zasoby: Core ~150–300 MB RAM, Edge ~200–400 MB, Redis/PG wg skali.

---

19. Runbooki (skrót)

- **Panic**: `/panic` → global locks + slowmode.
- **Triage**: sprawd■ incident timeline, potwierd■ regu■■.
- **Restore**: `/restore last` lub zakresowy; obserwuj RTO i 429.
- **Maintenance**: `/maintenance start 30m`, po zako■czeniu snapshot + `stop`.

---

20. Ryzyka i obej■cia

- Utrata permsów bota → alert + instrukcja r■cznego nadania najwy■szej roli.
- Outage Redis/PG/S3 → tryb degradacji (tylko CUT + minimalny restore z cache).
- Discord API zmiany → pin wersji, testy e2e przy aktualizacji libów.

---

21. Roadmap

- v1: CUT, core regu■y, continuous backup, restore, panel podstawowy.
- v1.1: policy WASM, granular member■roles restore, SIEM integracja.
- v1.2: ML heurystyki anomalii, auto■tuning progów, multi■guild dashboard.

---

22. Za■■czniki

- Schematy ERD, przyk■adowe polityki (YAML), wzór raportu incydentu, przyk■adowe dashboardy Prometheus/Grafana.