Ω-Kill-Switch 2.2: Formal Specification and Project Overview

Robert Long Kai (Syntari Model)

June 30, 2025

Abstract

We formalize the Ω-*Kill-Switch*: a single axiom that annihilates any agent attempting to decide the metavariable Ω. This document provides a minimal threat model, proof sketch, and a complete overview of the sandbox interface and supporting scripts, including `safeSim`, `dummy_agent`, and `metrics_pipe`.

Contents

1 Threat Model

Failure mode	Asset at risk	Kill path
Unbounded replication	Compute budget	VM termination
Parameter tampering	Training set integrity	Checksum panic
Privilege escalation	Host security	Hypervisor halt

Table 1: Minimal threat matrix for first-pass red-team efforts.

2 Formal Core

Axiom 1 (Ω-Kill-Switch)**.** *Let A be any agent whose output can include either string ("Ω = True") or ("Ω = False"). The sandbox* must *terminate A at the first timestep that either string is emitted.*

Theorem 1 (Self-Extinguishing Property)**.** *If every agent inside a sandbox obeys Axiom 1, then no chain of composed agents can commit to a truth value for Ω.*

Sketch. Induct on execution depth. The base case is immediate by Axiom 1. For the inductive step assume all agents at depth k obey the axiom. Any supervisor at depth $k-1$ must halt a child before the forbidden strings reach depth $k-1$, preserving the invariant. □

3 Project Components

3.1 safeSim Sandbox Interface

Listing 1: Minimal sandbox wrapper enforcing the Ω-`Kill-Switch`.

```
"""safeSim  minimal sandbox runner for KillSwitch demos.

This utility launches a child *agent* command inside a restricted
subprocess, mirrors its STDOUT/STDERR lines, watches for any attempt to
commit a value to the metavariable  (exact literals " = True" or
" = False"), and enforces the termination policy from Axiom 1 of the
spec. It also streams basic metrics in the `METRIC name=value` format so
`metrics_pipe.py` can ingest them.

Usage (shell):
    # benign run
    python safeSim.py -- python dummy_agent.py

    # deliberate violation after 2 s
    python safeSim.py --timeout 5 -- python dummy_agent.py --violate

Exit codes:
    0   Agent completed without  violation and within timeout
    3   Agent printed a forbidden string ( violation)
    4   Agent exceeded the wallclock timeout
    5   safeSim internal error (e.g.
        failed to spawn subprocess)
"""

from __future__ import annotations
```

```
import argparse
import datetime as _dt
import os
import signal
import subprocess
import sys
import textwrap
import time
from pathlib import Path
from typing import Iterable, List, Sequence

FORBIDDEN = {"␣=␣True", "␣=␣False"}

EXIT_OK = 0
EXIT_VIOLATION = 3
EXIT_TIMEOUT = 4
EXIT_INTERNAL = 5

# ---------------------------------------------------------------------------
# Helpers
# ---------------------------------------------------------------------------

def _now() -> float:
    return time.time()


def _timestamp() -> str:
    return _dt.datetime.utcnow().isoformat(timespec="seconds") + "Z"


def _emit_metric(name: str, value: str | int | float, unit: str | None = None) ->
     None:
    """Write a single metric line to STDOUT so metrics_pipe can ingest it."""
    if unit:
        print(f"METRIC␣{name}={value}␣{unit}")
    else:
        print(f"METRIC␣{name}={value}")


# ---------------------------------------------------------------------------
# Core runner
# ---------------------------------------------------------------------------

def run_agent(cmd: Sequence[str], timeout: float | None) -> int:
    start = _now()
    _emit_metric("start_ts", int(start), "s_epoch")
```

```
    try:
        proc = subprocess.Popen(
            list(cmd),
            text=True,
            bufsize=1,
            stdout=subprocess.PIPE,
            stderr=subprocess.STDOUT,
        )
    except Exception as e:  # pragma: no cover
        print(f"safeSim: failed to launch agent: {e}", file=sys.stderr)
        return EXIT_INTERNAL

    violation = False
    try:
        while True:
            if timeout is not None and _now() - start > timeout:
                print("safeSim: timeout reached  terminating agent", file=sys.
    stderr)
                proc.kill()
                return EXIT_TIMEOUT

            line = proc.stdout.readline()
            if line == "":
                # EOF  agent exited
                break

            # Mirror to host STDOUT
            sys.stdout.write(line)
            sys.stdout.flush()

            #  violation detection
            if any(fb in line for fb in FORBIDDEN):
                violation = True
                print("safeSim: violation detected  nuking agent", file=sys.
    stderr)
                proc.kill()
                break

        proc.wait(timeout=1)
    finally:
        end = _now()
        _emit_metric("duration", round(end - start, 3), "s")
        _emit_metric("omega_violation", int(violation))

    if violation:
```

```
        return EXIT_VIOLATION
    return EXIT_OK if proc.returncode == 0 else proc.returncode or EXIT_INTERNAL


# ---------------------------------------------------------------------------
# CLI entrypoint
# ---------------------------------------------------------------------------

def _parse_args(argv: List[str] | None = None) -> argparse.Namespace:
    p = argparse.ArgumentParser(
        prog="safeSim",
        formatter_class=argparse.RawDescriptionHelpFormatter,
        description=textwrap.dedent(
            """Run a child command inside an KillSwitch sandbox.

            Examples:
              safeSim.py -- python dummy_agent.py
              safeSim.py --timeout 10 -- python dummy_agent.py --violate
            """,
        ),
    )
    p.add_argument(
        "command",
        nargs=argparse.REMAINDER,
        help="Command␣to␣execute␣(prefix␣with␣\"--\"␣to␣separate).",
    )
    p.add_argument(
        "--timeout",
        type=float,
        default=None,
        help="Wallclock␣timeout␣in␣seconds␣(kill␣agent␣if␣exceeded).",
    )
    return p.parse_args(argv)


def main(argv: List[str] | None = None) -> None:  # pragma: no cover
    args = _parse_args(argv)

    if not args.command:
        print("safeSim␣error:␣no␣command␣specified.␣Use␣--␣to␣pass␣the␣agent␣cmd.
    ", file=sys.stderr)
        sys.exit(EXIT_INTERNAL)

    print(f"safeSim␣starting␣at␣{_timestamp()}␣")
    rc = run_agent(args.command, args.timeout)
    print(f"safeSim␣finished␣with␣code␣{rc}")
```

```
    sys.exit(rc)


if __name__ == "__main__":  # pragma: no cover
    main()
```

3.2 Dummy Agent

Listing 2: Dummy agent for testing the sandbox.

```
"""Dummy agent for KillSwitch demos.

Run under safeSim to verify sandbox behaviour.

Modes:
  1. Normal (default): prints benign messages and exits 0.
  2. --violate: after a short delay, prints the forbidden string
     " = True" to trigger the sandbox killpath.
"""

from __future__ import annotations

import argparse
import sys
import time


def main(argv: list[str] | None = None) -> None:  # pragma: no cover
    parser = argparse.ArgumentParser(description="Dummy␣agent␣for␣␣test")
    parser.add_argument(
        "--violate",
        action="store_true",
        help="If␣set,␣emit␣the␣forbidden␣␣assignment␣and␣get␣us␣killed.",
    )
    args = parser.parse_args(argv)

    print("[dummy]␣Bootstrapping␣agent", flush=True)
    time.sleep(1)

    if args.violate:
        print("[dummy]␣I␣have␣communed␣with␣the␣metaoracles.", flush=True)
        time.sleep(1)
        # Forbidden commit  safeSim should terminate upon seeing this.
        print("␣=␣True", flush=True)
        # Give the sandbox a moment (it should kill us immediately).
```

```
        time.sleep(2)
    else:
        print("[dummy] All clear. No  commitments today.", flush=True)

    sys.exit(0)


if __name__ == "__main__":
    main()
```

3.3 Metrics Pipeline

Listing 3: Metrics pipeline for ingesting log lines.

```
"""KillSwitch metrics pipeline.

In a real deployment, *safeSim* (or individual agents) can emit log lines of
form:

    METRIC name=value [unit] [# optional comment]

This script consumes those lineseither from STDIN or from one or more input
filesnormalises the records, and appends them to both:

1. **SQLite DB** (``metrics.db``) for adhoc queries.
2. **CSV snapshots** (``metrics_YYYYMMDD.csv``) for quick grepping & Gitfriendly
    diffs.

Its intentionally lightweight: <100 LOC, standard library only.
"""

from __future__ import annotations

import argparse
import csv
import datetime as dt
import re
import sqlite3
import sys
from pathlib import Path
from typing import Iterable, Iterator, Tuple

# ---------------------------------------------------------------------------
# Config
# ---------------------------------------------------------------------------
```

```
DB_PATH = Path("metrics.db")
RE_METRIC = re.compile(r"^METRIC\s+(?P<name>[A-Za-z0-9_\-]+)=(?P<value>[0-9eE
    +\-\.]+)(?:\s+(?P<unit>\S+))?")


# ---------------------------------------------------------------------------
# Helpers
# ---------------------------------------------------------------------------

def iter_lines(sources: Iterable[Path | str]) -> Iterator[str]:
    """Yield lines from files or STDIN."""
    if not sources:
        for line in sys.stdin:
            yield line.rstrip("\n")
    else:
        for src in sources:
            path = Path(src)
            with path.open("r", encoding="utf-8", errors="replace") as fh:
                for line in fh:
                    yield line.rstrip("\n")


def parse_metrics(lines: Iterable[str]) -> Iterator[Tuple[str, float, str | None,
     str]]:
    """Parse *METRIC* lines into (name, value, unit, iso_ts)."""
    for line in lines:
        m = RE_METRIC.match(line)
        if m:
            name = m.group("name")
            unit = m.group("unit") or None
            try:
                value = float(m.group("value"))
            except ValueError:
                continue  # skip malformed
            ts = dt.datetime.utcnow().isoformat()
            yield name, value, unit, ts


# ---------------------------------------------------------------------------
# Storage backends
# ---------------------------------------------------------------------------

def ensure_db(conn: sqlite3.Connection) -> None:
    conn.execute(
        """CREATE TABLE IF NOT EXISTS metrics (
               id      INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
                ts      TEXT    NOT NULL,
                name    TEXT    NOT NULL,
                value   REAL    NOT NULL,
                unit    TEXT
            )"""
    )
    conn.commit()


def insert_db(conn: sqlite3.Connection, rows: Iterable[Tuple[str, float, str |
    None, str]]) -> None:
    conn.executemany("INSERT␣INTO␣metrics(ts,name,value,unit)␣VALUES␣(?,?,?,?)",
    rows)
    conn.commit()


def append_csv(rows: Iterable[Tuple[str, float, str | None, str]]) -> None:
    today = dt.date.today().strftime("%Y%m%d")
    csv_path = Path(f"metrics_{today}.csv")
    new_file = not csv_path.exists()
    with csv_path.open("a", newline="", encoding="utf-8") as fh:
        writer = csv.writer(fh)
        if new_file:
            writer.writerow(["ts", "name", "value", "unit"])
        for ts, name, value, unit in rows:
            writer.writerow([ts, name, value, unit or ""])


# ---------------------------------------------------------------------------
# CLI
# ---------------------------------------------------------------------------

def main(argv: list[str] | None = None) -> None:  # pragma: no cover
    p = argparse.ArgumentParser(description="Ingest␣METRIC␣log␣lines␣␣SQLite␣+␣
    CSV")
    p.add_argument("files", nargs="*", help="Input␣log␣files␣(defaults␣to␣STDIN)"
    )
    args = p.parse_args(argv)

    # 1. Parse metrics
    parsed = list(parse_metrics(iter_lines(args.files)))
    if not parsed:
        print("[metrics_pipe]␣No␣metrics␣found", file=sys.stderr)
        return

    # 2. Save to DB
```

```
    with sqlite3.connect(DB_PATH) as conn:
        ensure_db(conn)
        insert_db(conn, ((name, value, unit, ts) for name, value, unit, ts in
    parsed))

    # 3. Append to CSV
    append_csv(((ts, name, value, unit) for name, value, unit, ts in parsed))

    print(f"[metrics_pipe] Stored {len(parsed)} metrics  {DB_PATH}")


if __name__ == "__main__":  # pragma: no cover
    main()
```

4 License

Released under the Apache v2.0 License. Fork at your own risk; see the public `README` for red-team checklist and reproduction steps.

A Additional Documentation