How TRIPWIRE works

A wire is one order hung on an event. You fix one swap (what for what, how much, the worst price you accept), deposit the input into a program vault on Solana, and set a trigger. When the trigger is met, the keeper fires that one swap to your wallet. Until then the wire waits armed. You can cancel it at any time.

TRIPWIRE is not an exchange, not a bot that buys news ahead of others, not an AI that manages money and not a prediction market. The one thing that can happen to a deposit is the swap its owner set.

Trigger kinds and who judges them

Triggers are judged off chain by the project's keeper and watchers. A wrong trip still runs only the owner's own swap, inside the owner's worst price.

ODDS WIRE

"YES on this prediction-market question is above (or below) X%." A number compared with a line, with no AI involved. The watcher reads the market's YES price from the server and the wire trips only after **three consecutive reads** on the far side of the line, so a single spike does not trip it. All three confirming reads come from one read path, and the receipt names that path.

RELEASE WIRE

A figure from a scheduled release (a rate decision, an inflation print, an earnings figure), read from its fixed official source page and compared with a line. The source page and the extracted number go into the receipt.

SENTENCE WIRE

A yes/no sentence about the news, picked by the project and open to everyone. An AI model reads candidate articles from the fixed source list; the wire trips only when **two or more different listed sources** state the fact. One source can be wrong, so one is never enough. The receipt lists both sources and their times.

What the keeper can and cannot do

The keeper key can call `trip` on an armed wire, close expired wires (returning the input to the owner) and sweep the burn bucket. It cannot change the output asset, the amount, the

destination or the worst price, and it cannot withdraw a deposit.

On every trip the program itself checks:

- Only the configured keeper can call it, and only on a wire that is still armed. A wire trips at most once: a trip closes it.
- The fee is moved from the vault into the two buckets before the swap; the swap may spend at most the input minus the fee.
- The swap program is pinned in the program's code (Jupiter on mainnet). The wire's own signature is the only one the program adds, and the swap may not touch the program's config or buckets.
- The output is measured by the program from what actually arrived, then moved to the owner's own token account for the output mint (derived by the program, never passed in freely). The owner's balance change must be at least the wire's minimum output. The keeper supplies no number this check uses.
- Any input left over goes back to the owner, and the wire and its vault are closed to the owner.
- Any failed check fails the whole transaction and the wire stays armed.

The admin (the project's dev wallet) can change the config (fee, limits, holding threshold, the keeper key), pause new arming and withdraw the operating bucket only. The fee has a hard cap of 1.00% in the program. A pause stops new wires only: cancel, trip and expiry closing stay open.

Worst price and minimum output

When you arm, you set a worst price: the least output you accept per one unit of input. The app turns it into a **minimum output** in the output token's base units, from the input that reaches the swap after the fee, rounded up. That minimum is stored in the wire on chain.

If the route cannot deliver the minimum when the trigger is met, the trip transaction fails whole and the wire stays armed and shows "waiting on price". The keeper tries again while the trigger holds. On a large headline other traders move too; the worst price is what bounds your fill.

Cancel and expiry

You can cancel an armed wire at any time. The whole input comes back to your wallet in the same transaction, with no fee, and the wire's rent comes back too.

Every wire has an expiry you choose: **30 days** by default, at most **365 days**. After the expiry anyone can close the wire (the keeper does it), which returns the input to the owner. Arming and cancelling cost only network fees.

Receipts and fingerprints

Every tripped wire has a public receipt: the trigger text, the sources read and the values seen, the event time, the fire time and the gap between them, the input spent, the fee, the output received against the minimum output, the effective price against the worst price, and the transaction.

The trigger's text stays off chain until its wire trips. Only its SHA-256 fingerprint is stored in the wire on chain. The receipt publishes the body, so anyone can recompute the fingerprint:

- Take the trigger body exactly as published in the receipt, byte for byte, with no trailing newline added.
- Compute `sha256(utf8(body))` and write it as lowercase hex, for example `printf '%s' "$BODY" | sha256sum`.
- Compare it with the receipt's fingerprint and with the `trigger_hash` in the wire's `Armed` event on chain.

See the receipts.

Fee and its split

A wire that trips pays an execution fee of **0.30% of its input**, taken in the input asset. Nothing is charged on arming, cancelling or expiry.

- **0.20%** of the input goes to the burn bucket inside the program. When the bucket holds at least 5 USDC of value, the keeper swaps it into $TRIPWIRE and burns exactly what the swap delivered, in one transaction.
- **0.10%** of the input goes to the keeper's operating bucket, which pays for running the keeper.

Limits at launch

- Per wire: **1 to 1,000 USDC**, or **0.01 to 5 SOL**. Inputs are USDC and SOL (SOL is wrapped in the arming transaction).
- Expiry: default 30 days, at most 365 days.
- AI calls: at most 500 per UTC day for news judgments, across the whole service.

The admin can raise the limits. The live values are read from chain below and on proof.

Place in line

When several wires hang on the same event, they fire in order of each owner's $TRIPWIRE balance read at the moment of the trip, larger balances earlier. My wires shows each armed wire's place in line.

What is on chain and what is on the server

On chain (Solana program, readable by anyone): the config (admin, keeper key, fee, burn share, limits, holding threshold, the $TRIPWIRE mint); each wire (owner, input mint and amount, output mint, minimum output, trigger fingerprint, trigger kind, expiry); each wire's vault balance; the burn and operating buckets; and the program's events for arm, trip, cancel, expiry and burn.

On the server: the trigger bodies (private until their wire trips), the event list, the source list, the readings the watchers take, the judging of triggers, the receipts' evidence and the daily AI count. Chain state the site shows as settled (armed, tripped, balances) is read from chain.

When the keeper is down

Nothing trips while the keeper is down. Deposits stay in their vaults and cancel works, because cancel is signed by the owner alone. The keeper's state, each watcher's last read and the AI budget are shown on status.

Before mainnet

Before mainnet the site runs on devnet. Every flow is real there and odds are the real market values, but the swap leg runs through a stand-in pool, since Jupiter and stock tokens do not exist on devnet. Devnet records are marked devnet.

Third-party APIs

- Swaps are routed through Jupiter's **Metis Swap API**.
- Odds values are read from Jupiter's **Prediction Market API**. When that path is unavailable, the watcher reads the market provider's own API; one confirmation never mixes the two.
- Screens that quote or run a swap, or show an odds value read from Jupiter, carry the line "Powered by Jupiter" with the name of the API used.
- TRIPWIRE uses these APIs under their published terms. No partnership with, or endorsement by, Jupiter or any market provider is implied.

Live config, read from chain

Program id	`43C7e2a912yyeVE3prEogq5j2iPMrQQRmKXzSFJ4BX7b`
Upgrade authority	`CZzwkZ4s856yqZjJ58KBT2GPV1Tp4rq5Zbx1mgBpMCbm`
Execution fee (on trip only)	`0.30% of the input`
Burn bucket share	`0.20% of the input`
Operating bucket share	`0.10% of the input`

Per-wire limit	1 USDC to 1,000 USDC	Per-wire limit	0.01 SOL to 5 SOL
Read from chain		2026-11-25 01:14 UTC (slot 525981493) on devnet	

Source list

Sentence and release wires are judged only from these sources.

- Al Jazeera - Wire service, read by rss
- Ars Technica - Wire service, read by rss
- BBC News Business - Wire service, read by rss
- BBC News Technology - Wire service, read by rss, typically 27 min 1 s behind
- Bank of England news - Official source, read by rss
- Business Wire (company releases) - Company, read by rss, typically 6 min 31 s behind
- CNBC Technology - Wire service, read by rss
- CNBC Top News - Wire service, read by rss, typically 2 min 18 s behind
- CoinDesk - Wire service, read by rss
- Cointelegraph - Wire service, read by rss
- Decrypt - Wire service, read by rss, typically 1 min 29 s behind
- European Central Bank press - Official source, read by rss
- Federal Reserve press releases - Official source, read by rss
- Fox Business - Wire service, read by rss
- GlobeNewswire (company releases) - Company, read by rss
- Jupiter Prediction API - Official source, read by json
- Kalshi public API - Official source, read by json
- MarketWatch Top Stories - Wire service, read by rss, typically 2 min 16 s behind
- NVIDIA Newsroom - Company, read by rss
- PR Newswire (company releases) - Company, read by rss, typically 2 min 1 s behind
- Polymarket CLOB API - Official source, read by json
- TechCrunch - Wire service, read by rss, typically 33 s behind
- The Block - Wire service, read by rss
- The Guardian Business - Wire service, read by rss
- The New York Times Business - Wire service, read by rss
- The New York Times Technology - Wire service, read by rss
- The Verge - Wire service, read by rss, typically 32 s behind
- The Wall Street Journal Markets - Wire service, read by rss, typically 3 min 1 s behind
- The White House news - Official source, read by rss
- U.S. Bureau of Economic Analysis releases - Official source, read by rss

- U.S. Bureau of Labor Statistics releases - Official source, read by rss