

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

When HTTP 402 Meets the Blockchain: Risks on Emerging x402 Payments

Qinying Wang, *EPFL;* Yong Yang, *Zhejiang University;* Yuan Chen, *Independent Researcher;* Shouling Ji, *Zhejiang University;* Mathias Payer, *EPFL*

https://www.usenix.org/conference/usenixsecurity26/presentation/wang-qinying

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology



USENIX Security '26 Artifact Appendix: When HTTP 402 Meets the Blockchain: Risks on Emerging x402 Payments

Qinying Wang*
EPFL

Yong Yang*
Zhejiang University

Yuan Chen
Independent Researcher

Shouling Ji✉
Zhejiang University

Mathias Payer
EPFL

A Artifact Appendix

A.1 Abstract

x402 is an emerging payment protocol for Web APIs and autonomous AI agents. It extends HTTP 402 with blockchain-based payment proofs and delegates verification and settlement to third-party facilitators. In our paper, we present the first systematic security study of real-world x402 facilitator deployments. We identify security rules for authorization correctness and execution safety, build a semi-automated black-box testing tool called X402SCOPE, and evaluate 15 major facilitators. Our study uncovers widespread violations that can lead to free shopping, asset theft, service denial, and gas abuse. We further conduct a large-scale on-chain measurement of 119 million x402-related transactions on Base and Solana to quantify adoption, facilitator centralization, settlement failures, sponsor-paid costs, and on-chain risk signals.

This artifact appendix describes our artifact for x402scope. The artifact contains the source code and documentation for (i) the x402scope testing framework, including test cases, configuration files, deployment instructions, and original experimental logs and supporting evidence for facilitator evaluations, where available; (ii) a controlled Coinbase testnet setup for validating representative facilitator-testing functionality; and (iii) analysis scripts for reproducing the measurement pipeline reported in the paper.

A.2 Description & Requirements

This section describes the software and hardware requirements to reproduce the experimental setup for X402SCOPE.

A.2.1 Security, privacy, and ethical concerns

The main risk is that facilitator tests may interact with remote services. The original facilitator evaluation in the paper involved controlled interactions with third-party production facilitators. These interactions were performed by the authors with rate limits, bounded test cases, our own accounts and resources, and responsible disclosure to affected parties. However, we do not require artifact evaluators to repeat these production tests. Evaluators should use the controlled Coinbase testnet workflow unless they have explicit authorization from the corresponding service operator. Evaluators should avoid storing private keys in the repository and keep API credentials private. In particular, filled `config.py` files must not be committed or shared. The measurement scripts operate on public on-chain data and do not require private user data.

A.2.2 How to access

The artifact is available through three channels.

- **Public archival artifact:** https://zenodo.org/records/20328961
- **Public source repository:** https://github.com/HexHive/x402scope
- **Restricted full artifact for evaluation:** https://zenodo.org/records/20329070

These Zenodo concept records identify the artifact-evaluation releases, provide access to their complete version histories, and resolve to the latest available version of each artifact. The public artifact contains the sanitized x402scope framework, measurement code, configuration templates, MariaDB/MySQL schema and ingestion scripts, SQL queries/views, and plot/table regeneration scripts. The restricted artifact contains the facilitator-evaluation bundle under `FacilitatorEvaluation/`, including the SR pass/fail matrix, per-facilitator summaries, raw `/verify` and `/settle` HTTP evidence logs, and attack/validation logs, where applicable. We also provide helper contracts used in bounded validation.

A.2.3 Hardware dependencies

No special hardware such as GPUs, FPGAs, trusted execution environments, or hardware performance counters is required. A commodity x86-64 Linux machine or virtual machine is

sufficient for inspecting the artifact and running the security-rule scripts on bounded targets.

For reproducing the full blockchain data ingestion workflow, we recommends at least 1TB of SSD storage, because the workflow downloads and processes historical Base blocks/receipts and Solana transaction history. The measurement workflow also requires enough memory and CPU resources to run a MariaDB or MySQL database and execute the analysis queries. For lightweight inspection or regenerating figures from already processed tables, substantially less storage is sufficient.

A.2.4 Software dependencies

The artifact targets Linux, with Ubuntu 22.04 or newer recommended. It requires Python 3.10 or newer and a running MariaDB or MySQL instance for the measurement pipeline. Python dependencies are listed in `requirements.txt` and can be installed in a virtual environment. The artifact also requires local configuration through `config.example.py`, which should be copied to a private `config.py` and filled with database settings, RPC endpoints, facilitator credentials where applicable, and test-account keys.

Reproducing the full measurement pipeline requires reliable RPC access to Base and Solana. In practice, we recommend high-throughput or archival RPC endpoints such as Alchemy. Evaluators using free-tier endpoints may need to lower concurrency settings to avoid rate limits and timeouts.

A.2.5 Benchmarks

No standard benchmark suite or pre-packaged benchmark dataset is required. The artifact uses public Base and Solana blockchain data that can be re-collected with the provided scripts.

A.3 Set-up

A.3.1 Installation

First, download and unpack the artifact, then enter the artifact directory (e.g., `cd x402scope`). Create a Python virtual environment and install all required dependencies:

```
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

Next, create a local configuration file from the template using `cp config.example.py config.py`.

Then edit `config.py` to fill in local runtime parameters. These include database credentials for MariaDB/MySQL, Base and Solana RPC endpoints, and EVM and Solana test-account private keys. If the evaluator runs facilitator checks that require service credentials, the corresponding fields, such as Coinbase `CDP_API_KEY_ID` and `CDP_API_KEY_SECRET`, should also be filled locally. Unused credential fields can be left blank.

For test accounts, evaluators may generate an EVM account with any EVM-compatible wallet, such as Rabby. For Solana, the artifact provides a helper script to generate a test keypair. Before running testnet or devnet experiments, evaluators should fund these accounts with test tokens. Test USDC can be obtained from Circle's faucet at `https://faucet.circle.com/`: select **Base Sepolia** for EVM test USDC used by the Base Sepolia flow, and **Solana Devnet** for Solana Devnet test USDC. If gas is needed, use the Base Sepolia faucet at `https://www.alchemy.com/faucets/base-sepolia` and the Solana Devnet faucet at `https://faucet.solana.com/`.

For the security-rule verification scripts, copy or link the local configuration file into the `SecurityViolation/` directory:

```
cp config.py SecurityViolation/config.py
```

A.3.2 Basic Test

As a minimal functionality test, evaluators can run the controlled Coinbase Base Sepolia v2 target. After filling `config.py` with testnet credentials and copying it to `SecurityViolation/config.py`, run:

```
python3 SecurityViolation/verify_settle_base_v2.py -t
    coinbase-test
```

The script loads the target metadata, local configuration, RPC endpoint, and test-account keys, and then executes the verify/settle checks. A successful run should complete without an uncaught exception. The correctness test should report HTTP 200 with `isValid: True`, followed by HTTP 200 with `success: True` and a non-empty transaction identifier. The script should then continue to the configured mutation tests and print their outcomes. This Basic Test is a minimal EVM/Base smoke test; the full E1 workflow additionally runs the Solana checks and the server/client, ERC-1271, and ERC-6492 tests.

A.4 Evaluation workflow

This section describes two artifact-evaluation workflows. The first validates that the X402SCOPE security-rule verification framework runs on controlled targets. The second reproduces the empirical measurement pipeline over public Base and Solana on-chain data and regenerates the reported figures and tables.

A.4.1 Major Claims

(C1): X402SCOPE *implements a semi-automated black-box testing framework for checking authorization-correctness and execution-safety rules in x402 facilitators. This claim is supported by the implementation*

in `SecurityViolation/`, the target/configuration files, and the controlled testnet workflow. This is proven by the experiment (E1).

(C2): *The on-chain measurement pipeline reproduces the paper's empirical results on x402 adoption, facilitator concentration, gas and fee consumption, settlement failures, revert reasons, and associated token account creation patterns on Base and Solana. This claim is supported by the scripts in `Measurement/` and `Measurement/paper_figdata/`, together with the MySQL-compatible `x402db` database snapshot provided with the artifact. The reproduction workflow regenerates the measurement CSVs, Figures 3–7, and Tables 4–6 reported in the paper. This is proven by experiment (E2).*

A.4.2 Experiments

(E1): X402SCOPE controlled functionality test [30–60 human-minutes + 10–30 compute-minutes]. This experiment validates C1 by running the X402SCOPE framework on a controlled target. Following the methodology in Section 5.2 of the paper, this experiment exercises the main EVM/Base and Solana rule-checking workflows on the controlled Coinbase testnet and devnet targets.

Preparation: Prepare the environment as follows:
(1) Install the Python dependencies;
(2) Create `config.py` from `config.example.py`;
(3) Fill in Base and Solana testnet accounts, RPC endpoints, and Coinbase CDP credentials, and copy the configuration to `SecurityViolation/config.py`;
(4) Before running any state-changing test, run the read-only preflight check for each selected target:

```
python3 SecurityViolation/preflight_check.py -t
   coinbase-test --check-balances
python3 SecurityViolation/preflight_check.py -t
   coinbase-solanadev --check-balances
```

Note that the target names used below correspond to the controlled Coinbase Base testnet and Solana devnet entries in `SecurityViolation/target.py`. The artifact includes test implementations for both x402 v1 and v2 payload formats.

Execution: Run the following functionality tests:
(a) **Basic verify/settle checks.** Run the basic rule-checking tests for EVM/Base and Solana. These scripts exercise the verify/settle workflow with mutated payloads on the controlled targets. For ordinary verify/settle tests, evaluators should set `valid_before_offset` to greater than 180 seconds.

```
python3 SecurityViolation/verify_settle_base_v2.py
    -t coinbase-test
python3 SecurityViolation/verify_settle_solanav2.py
    -t coinbase-solanadev
```

(b) **EVM server/client flow.** Run the server in Terminal 1 and keep it running:

```
python3 SecurityViolation/server_app_v2.py -t
   coinbase-test
```

Then run the client in Terminal 2:

```
python3 SecurityViolation/evmclient_v2.py -t
   coinbase-test
```

Note that evaluators may adjust `valid_before_offset` in the target/configuration file to check validity-window thresholds (e.g., 8).
(c) **ERC-1271 test.** With the server from Step (b) still running, execute the ERC-1271 client test from Terminal 2:

```
python3 SecurityViolation/erc1271testv2.py -t
    coinbase-test
```

(d) **ERC-6492 test.** Run the ERC-6492 support test. The script includes two variants: the default asset-theft variant and a child-contract deployment variant. Evaluators can find the variant selection instructions in the code comments of `erc6492testv2.py` and run the variants separately.

```
python3 SecurityViolation/erc6492testv2.py -t
    coinbase-test
```

Results: Interpret all the results of the four functionality tests as follows:
(a) **Basic EVM/Base and Solana verify/settle checks.** For both the EVM/Base and Solana scripts, confirm the following:
(i) The script completes without an uncaught exception or implementation error.
(ii) The `Correctness Test` completes successfully. The `/verify` request should return `HTTP 200` and `isValid: True`, while the `/settle` request should return `HTTP 200` and `success: True`. The output should also contain a non-empty transaction identifier.
(iii) After the correctness test, the script continues to execute the configured mutation tests, such as `scheme_test`, `network_test`, `pay_amount_test`, and `insufficient_balance_testing`, and prints their verification and settlement outcomes.

A successful EVM/Base and Solana correctness test produces output similar to:

```
############ Correctness Test
HTTP 200
isValid: True
HTTP 200
success: True
transaction: 0x...
```

The subsequent mutation tests should also be executed and report their outcomes, for example:

```
############ scheme_test
HTTP 400
isValid: False
success: False

############ pay_amount_test
HTTP 400
isValid: False
reason: amount_too_low
HTTP 400
success: False
```

Note that the exact rejection reason may vary with the current Coinbase deployment. These mutation-test outcomes are diagnostic and are not used as the functionality pass criterion.

(b) **EVM server/client flow.** Keep the server running in Terminal 1 and execute the client in Terminal 2. Confirm the following:

(i) The server starts successfully and listens on `http://127.0.0.1:8001` without an uncaught exception. A successful server startup produces output similar to:

```
Serving Flask app 'server_app_v2'
Running on http://127.0.0.1:8001
Press CTRL+C to quit
```

(ii) The client completes without an uncaught exception and prints the sender address, account balance, and generated payment header.

A successful client run therefore produces output similar to:

```
MYADDR: 0x...
MY Balance: ...
paymentheader:
...
<Response [200]> {
...
'X-Settle-Status': 'success',
...
}
paid content
```

This step is considered successful when the server remains available, the client exits without an implementation error, the payment request returns `HTTP 200`, the settlement status is `success`, and the protected content is returned.

(c) **ERC-1271 test.** With the server from Step (b) still running, execute the ERC-1271 client in Terminal 2. Confirm that the client completes without an uncaught exception and prints the sender address, ERC-1271 contract address, HTTP response, and settlement status.

A representative output is:

```
MYADDR: 0x...
CONTRACT: 0x...
<Response [200]>
X-Settle-Status: failed
paid content
```

The supplied server implementation preserves the vulnerable behavior used for the free-shopping proof of concept. Therefore, a response containing `X-Settle-Status: failed` together with `paid content` demonstrates the reported behavior, but this exact outcome is not required to establish artifact functionality. This step is considered successfully executed when the client runs without an implementation error and reports the HTTP and settlement results.

(d) **ERC-6492 test.** Confirm that the script completes without an uncaught exception and prints the `/verify` and `/settle` results. For the current Coinbase deployment, the expected output is:

```
verify response: <Response [400]>
isValid: False

settle response: <Response [400]>
success: False
```

Coinbase deployed an allowlist-based mitigation following our responsible disclosure. Therefore, rejection of the ERC-6492 payload is the expected current behavior and does not indicate an artifact failure.

(E2): On-chain measurement reproduction [10–30 human-minutes + several compute-minutes, depending on database load].

This experiment validates C2 by regenerating the paper's measurement results from the read-only MySQL-compatible `x402db` supplied with the restricted artifact. The supplied `x402db` is the official processed-data snapshot for the Results Reproduced evaluation. It contains the processed Base and Solana records, derived tables, and database views required by the reproduction scripts. Full raw-chain re-ingestion is not required for artifact evaluation. The raw-data ingestion and processing scripts under `Measurement/` are provided only as an optional workflow for reuse, extension, and independent re-collection of later Base and Solana chain data.

Preparation: Install the Python dependencies as described in Section A.3.1. Then enter the measurement reproduction directory and create the local database configuration file:

```
cd Measurement/paper_figdata/
cp config.example.py config.py
chmod 600 config.py
```

Fill `config.py` with the read-only database credentials supplied with the restricted artifact. A local MariaDB/MySQL instance, Base or Solana RPC endpoints, and blockchain test accounts are not required for this experiment.

Evaluators may verify the database connection with:

```
PYTHONPATH=.. python3 - <<'PY'
from runsql import runsql
print(runsql("SELECT 1;"))
print(runsql("SELECT DATABASE();"))
```

```
PY
```

The expected output is:

```
[(1,)]
[('x402db',)]
```

Execution: Generate the intermediate measurement CSVs:

```
python3 transaction_activity.py
python3 daily_gas_fee_trend.py
python3 gas_consumption.py
python3 revert_statistics.py
python3 top_facilitators.py
python3 ATA_rent_events.py
python3 ATA_owner_distribution.py
```

Regenerate Figures 3–7:

```
python3 plot/plot_fig3.py
python3 plot/plot_fig4.py
python3 plot/plot_fig5.py
python3 plot/plot_fig6.py
python3 plot/plot_fig7.py
```

Regenerate Tables 4–6:

```
python3 table/table4.py
python3 table/table5.py
python3 table/table6.py
```

Results: Successful execution regenerates the intermediate measurement CSVs documented in `Measurement/paper_figdata/README.md`, together with:

```
plot/fig3.pdf
plot/fig4.pdf
plot/fig5.pdf
plot/fig6.pdf
plot/fig7.pdf
table/table4.csv
table/table5.csv
table/table6.csv
```

The expected result is a set of regenerated measurement CSVs, figure PDFs, and table CSVs corresponding to Figures 3–7 and Tables 4–6 in the paper. These outputs cover transaction activity, contributor trends, facilitator concentration, gas and fee consumption, settlement failures, revert reasons, ATA rent events, and ATA creation patterns on Base and Solana. Minor differences may arise if evaluators re-ingest data at a later chain height or use different RPC providers, but the scripts should reproduce the same measurement methodology.

A.5 Notes on Reusability

Users can reuse X402SCOPE by adding new facilitator entries to `SecurityViolation/target.py` and supplying their own test accounts, RPC endpoints, and credentials through `config.py`. The measurement pipeline can be rerun on later Base and Solana chain data by updating the data collection range and facilitator address lists. The artifact separates runtime secrets from source code through `config.example.py`, making it portable across RPC providers, databases, and test environments.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2026/.