COMPSCI 2620 Final Project: WIDE EYE

By Jesse James and Ruben Valenzuela

Spring 2025

Overview

WIDE EYE is a lightweight, scalable open-source intelligence (OSINT) system that enables distributed and task-driven collection and analysis of publicly available information through a computer network. At present, the WIDE EYE system is capable of ingesting information from RSS/Atom feeds and a prototype GDACS feed, though it has been designed so that it's extensible to any structured source or API.

At a high level, this computer network consists of three types of nodes—clients, the dispatcher, and collectors—each of which drives all of its ports, file paths, refresh intervals, and log-file locations from its own "config.json". Every process initializes Python's "logging" module to write detailed INFO/DEBUG messages to a component-specific log file, and all code carries verbose docstrings.

A client node consists of a graphical interface, a JSON configuration file, and a backend process that establishes a gRPC connection to the dispatcher for user registration/authentication (backed by a secure SQLite user table with PBKDF2-SHA256-hashed passwords), task creation, and real-time display of results in both list and topographical form.

In a WIDE EYE deployment, the dispatcher node has four core functions: client registration/authentication, source cataloging, task management, and collector management Client registration and authentication allow end-users to connect and obtain a session token. Source cataloging parses a JSON file of "known" sources to deliver available categories and locations. Task management defines a master record of all tasks and their metadata, storing them persistently in SQLite. Collector management works with task management to assign sources to collectors based on load and recent heartbeats; if any collector misses two heartbeat intervals, tasks are immediately reassigned to the next least-loaded active collector. Supporting the dispatcher, each collector node is a volunteer, stateless entity. So long as it has its own "config.json", has registered and heartbeated successfully, a collector will receive

“TaskAssignment” messages listing relevant sources and an end time; it then ingests and sends information back at its configured refresh rate.

Development Process and Technical Specifications

First Steps

Entering this project, we originally planned the collector nodes to be initialized with a source, such that each collector is responsible for only one source, and rather than matching sources to a task and distributing them to the collectors, the dispatcher would match the tasks to the collectors themselves. To determine whether or not this approach was best suited for our system, we first sought to evaluate the amount of data retrieved by different RSS feeds. After creating a list with approximately 100 sources from around the world—including major international news and local sources in places of current conflict—we validated the return values of each entry, leading to a reduction to 54 sources. Among those 54 sources, we found that some major outlets had RSS feeds with more than 100 entries, all containing recent stories. Contrarily, we also found that other sources, particularly government feeds, had fewer than 10 entries at any given time, and often included stories from months or years prior. Moreover, the feeds varied in level of detail, with some offering thorough HTML copies of summaries, and some not even offering a publication date. It was this disparity, combined with the inherently complex and risky process of collector-side source determination, that led us to the critical pivot towards a stateless collector model. Although it requires greater developer support to maintain the catalog of sources, it eliminates a number of other more grave challenges and protects both the integrity and richness of the information pool.

Client Development

Having established some peace of mind with our new collector model, we decided to start actual development by working on the client. Although arguments certainly can be made for starting with the dispatcher and/or collector, we believed that developing the client first would allow us to more consistently align the function of the system with practical use cases and user expectations later in the development cycle. This proved to be true rather quickly—originally we planned to parse a user-side task query and use some form of natural language processing to correlate the query with relevant sources. However, we found it to be more intuitive on the front end—and far easier on the back end—to tag each source in the catalog with relevant categories and locations of origin, and give the user the chance to check boxes for which entities in those groups it wanted to match against. This was immediately prioritized when later developing the dispatcher. It is worth noting, however, that after the initial layout configuration, we worked back and forth between the dispatcher, collector, and client to incorporate and validate functionality.

Regarding the technical design, the client is built around a PySide6 (official binding for Qt) interface. On startup, the “MainWindow” method loads “client/config.json” (for dispatcher address/port and log file location) and configures file-based logging. It then loads a local JSON file of country, state, and city coordinates into a lookup dictionary (for later named-entity recognition-driven (NER) RSS result mapping), and initializes a Folium map centered on the globe. The interface is composed of a "QStackedWidget" with two pages: an authentication page where users register or log in via gRPC calls ("RegisterRequest"/"LoginRequest"), and a dashboard page. The dashboard page consists of a top bar with keyword input, two searchable multi-select dropdowns for categories and locations (dynamically loaded from the source catalog and implemented with the custom "MultiSelectSearchBox" class), date-time pickers for start and end times, and buttons to add tasks or view active tasks. The main content area is divided by a "QSplitter" into a map pane—rendered by saving the Folium map to HTML in a "QWebEngineView"—and a results pane with a filterable "QListWidget". In previous projects, we used Tkinter, but the goals of this project warranted a more sophisticated approach with PySide6. Given our limited experience with the library, ChatGPT was incredibly helpful in implementing the complex visual elements, especially the map.

For communication with the dispatcher service, the client uses a gRPC stub ("ClientDispatcherStub") over a channel configured from "config.json". Authentication handlers ("on_register", "on_login") send "RegisterRequest" and "LoginRequest" messages; upon successful login, the client stores its token and writes an INFO log. Task submission ("on_add_task") constructs a "TaskRequest"—including comma-joined category and location lists plus UTC timestamps converted from Qt’s "QDateTimeEdit" widgets via "google.protobuf.Timestamp"—and invokes "StartTask". All RPCs and UI events are logged. When a task is successfully added, the client spawns a background thread to call "StreamResults" continuously, emitting Qt signals ("result_received") to marshal incoming protobuf payloads into the UI thread without blocking.

Each incoming result is JSON-decoded, stored in an in-memory dictionary keyed by task ID, and then passed through spaCy’s "en_core_web_sm" model to detect “GPE” and “LOC” entities in the headline. Detected place names are looked up in the preloaded coordinate map;

matches generate Folium markers (batched to avoid excessive refreshes via a "QTimer") and are emitted via a second custom signal ("marker_signal") to be added to the map. Meanwhile, the text and metadata of the result are displayed in the results list, where a live substring filter hides or shows individual items as the user types. All mapping and filtering events, as well as any errors, are written to "client/client.log".

The client also periodically (every five minutes via a daemon thread) refreshes the available categories and locations by calling "ListAvailableCategories" and "ListAvailableLocations"—each call logs an INFO message—and updates the multi-select dropdowns. When the user clicks “Active Tasks,” a modal "QDialog" displays all currently running tasks—each labeled with a truncated ID and its parameters—and selecting one rebuilds the Folium map and results list to show only that task’s data.

Dispatcher

The dispatcher, as the workhorse of the WIDE EYE system, required considerably more planning and intentionality than the client. First and foremost, we outlined its responsibilities—client registration/authentication, source cataloging, task management, and collector management—and structured each as its own class or module.

On startup, the dispatcher process reads "dispatcher/config.json" for client and collector gRPC ports, paths to "tasks.db" and "users.db", heartbeat timeouts, and "dispatcher/dispatcher.log". It initializes two SQLite stores: "users.db" managed by "UserManager" (securely salting and hashing passwords via PBKDF2-SHA256) and "tasks.db" managed by "TaskManager" (persistent task metadata and status). The logging subsystem is configured to record INFO/DEBUG entries with timestamps.

Client-facing logic lives in "DispatcherService" (subclassing "ClientDispatcherServicer"), decorated with "grpc_safe" to catch exceptions and translate them into gRPC errors while logging full tracebacks. "Register" and "Login" RPCs interact with "UserManager", issuing per-session tokens on success. "StartTask" parses comma-separated categories and locations, builds UTC ISO timestamps, generates a UUID, and calls "match_sources" on "dispatcher/sources.json" to find relevant feed IDs. It then persists the task as "PENDING", and for each matched source calls "CollectorManager.assign_task_balanced()", which picks the least-loaded active collector whose last heartbeat is within the configured timeout. If at least one assignment succeeds, the task flips to "DISPATCHED"; otherwise it’s marked "FAILED". Every assignment and failure is logged.

Streaming results back to clients is handled by "StreamResults", which waits on a per-task "threading.Condition". When a collector submits new data, "SubmitTaskResult" records metrics ("tasks_completed_count", "last_result_time"), appends a "TaskResult" to the queue, and logs the event before calling "cond.notify_all()". Meanwhile, an expiry sweeper thread periodically marks tasks "COMPLETED" when their end time passes, logs the transition, and notifies any blocked streams so clients can terminate gracefully.

Collector-facing logic resides in "CollectorDispatcherService" (subclassing "CollectorDispatcherServicer"). "RegisterCollector" and "LoginCollector" call into "CollectorManager", which maintains in-memory "CollectorInfo" objects (name, secret, token, heartbeat timestamps, assignments, performance counters) protected by a "threading.Lock". The "Heartbeat" RPC updates each collector’s last seen timestamp; if any collector misses two heartbeat intervals, "StreamTasks" invokes "failover_dead_collectors()", which removes the dead collector, reassigns its pending tasks to other collectors, and logs each failover event. "StreamTasks" then streams new "TaskAssignment" messages exactly once per task.

This layered, config-driven architecture—separating secure authentication, persistent tasks, in-memory coordination, and dual gRPC services—ensures that registration, authentication, source cataloging, task dispatch, load-balanced assignment, result streaming, and heartbeat-driven failover all proceed concurrently and reliably, with full traceability via file logs and docstrings.

Collector

With client and dispatcher in place, we shifted focus to the collector, whose role is to execute dispatched task assignments by fetching and forwarding OSINT data. Our primary implementation lives in "collector/collector.py", which reads "collector/config.json" for dispatcher address/port, heartbeat intervals, RSS refresh rates, and its log file. On startup it configures "logging" to "collector/collector.log", then prompts the operator to register or log in via "RegisterCollector" and "LoginCollector" calls. Successful logins yield session tokens that the collector retains.

A dedicated heartbeat thread issues "HeartbeatRequest" every configured interval; each heartbeat is logged, and any failures are recorded at ERROR. The main loop calls "StreamTasks", which also triggers the dispatcher’s failover detection when a collector has gone silent. For each "TaskAssignment", a daemon thread waits until the specified start time, then loops until end time, invoking "_collect_rss" on each source URL. Using the "feedparser" library, it parses feeds, skips malformed entries, and maintains a per-task, per-source “seen” set keyed by "(task_id, source_url)" to avoid duplicates. Each new entry is wrapped in a JSON payload and sent as "CollectorTaskResult"; successes (and any submission failures) are logged at INFO/WARNING.

We also prototyped a "gdacs_collector.py" using "aio-georss-gdacs" over an asyncio TCP server—reading its own config and logging to "collector/gdacs.log"—but it remains a demonstration of the system’s extensibility rather than part of the production datapath. Both collectors share a common, docstring-driven structure that makes it trivial to add new data-collection methods (APIs, file watches, social media) by extending a simple handler registry.

Testing

To ensure correctness and reliability across our distributed pipeline, we adopted a multi-pronged testing strategy:

1. Smoke Scripts

- “collector_dispatcher_connection.py" logs in as a client, submits a two-minute task, and prints streamed results—validating end-to-end client→dispatcher→collector→dispatcher flow with real user auth and heartbeats.
- “dispatcher_client_connection.py" implements a dummy dispatcher stub so the client can be tested in isolation, verifying "Register", "Login", "StartTask", and "StreamResults" against a controlled server.

2. Unit Tests

- "test_user_manager.py" verifies secure user registration, duplicate-username rejection, and password hashing/authentication.
- "test_task_manager.py", "test_source_catalog.py", and "test_collector_manager.py" validate persistent tasks, catalog matching, collector registration, heartbeat counting, balanced assignment, expiration detection, and purging logic. They also simulate missed heartbeats and assert that "failover_dead_collectors()" reassigns tasks as expected.

3. Integration Tests

- "test_dispatcher_integration.py" spins up "DispatcherService" with temporary databases and a live "CollectorManager"; it tests real "Register"/"Login", "StartTask", and confirms that tasks persist as "DISPATCHED" with correct collector assignments.
- "test_client_integration.py" uses pytest fixtures and Qt's "QApplication" to monkey-patch the gRPC stub, set UI inputs (keywords, multi-select, date/time), invoke "on_add_task()", and assert that the constructed "TaskRequest" matches the token, comma-joined filters, and UTC timestamps.

4. Manual Verification

- Running the live RSS collector against known feeds confirms that "collector.log" records every heartbeat, assignment, and submission, and handles parsing errors gracefully.
- Simulating a collector crash by stopping its process triggers immediate failover events in "dispatcher.log", ensuring tasks never stall.

This combined suite ensures that each component behaves as intended, RPC interfaces remain stable, and the full end-to-end pipeline handles failures gracefully under real-world conditions.

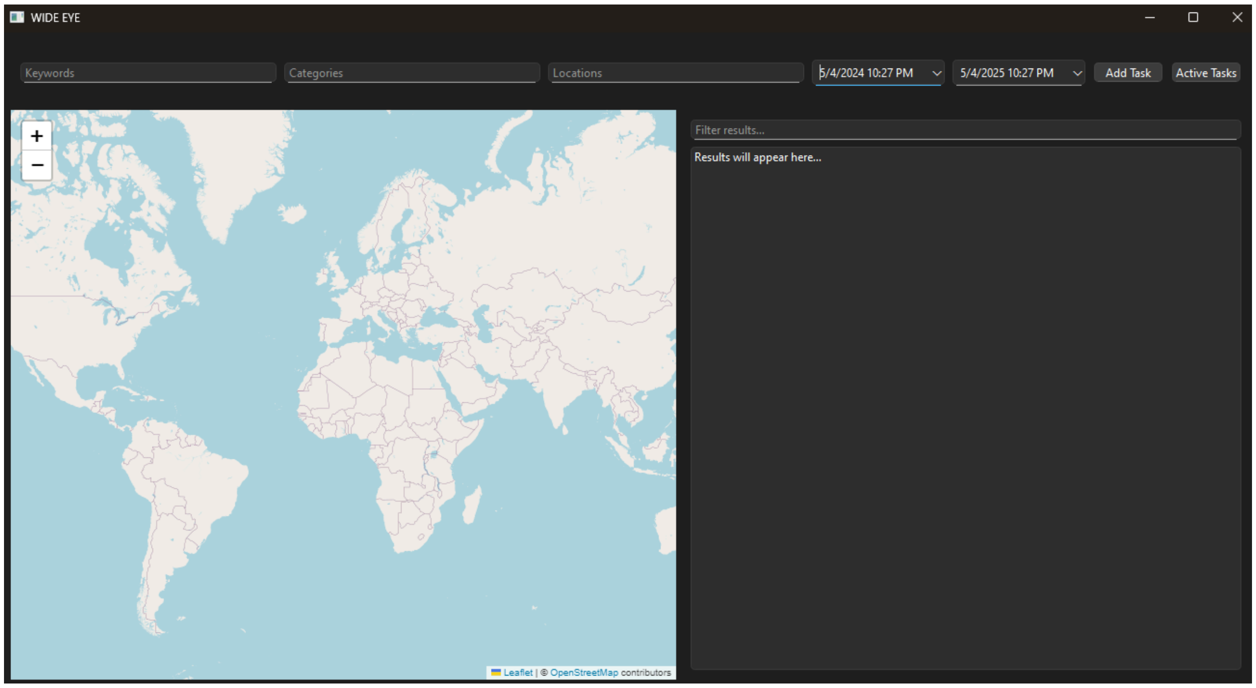
Lessons Learned

1. Configuration-first design pays dividends: By moving all ports, file paths, intervals, and log destinations into JSON config files, we made environment-specific tweaks trivial and reduced the risk of hidden magic values.
2. Verbose documentation and centralized logging accelerate debugging: The scale of this project, and particularly the three different types of nodes, definitely surpassed our previous experience. With full docstrings and file-based logs at INFO/DEBUG level though, we were able to keep up with changes and design effects. This pairs well with incremental testing.
3. Modular, stateless collectors scale effortlessly: Adding the GDACS prototype—or any future data source—requires minimal changes to the core gRPC orchestration code, validating our decision to keep collectors lightweight and driven entirely by RPC commands.

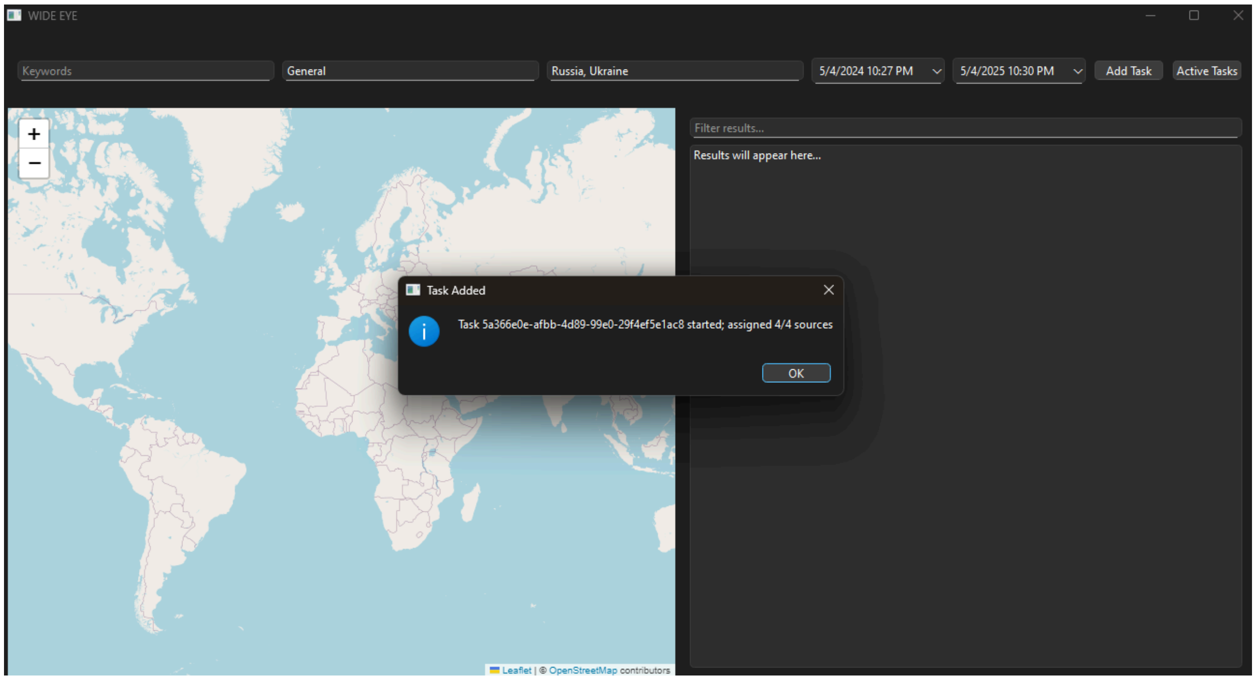
4. Balancing initial specifications with iterative feedback creates focus and flexibility: At many points, we had to deviate from our initial plan, either due to material constraints or lack of practicality. By designing with iteration in mind, we were able to move forward and adapt.

Together, these design choices have made WIDE EYE a robust, maintainable OSINT platform that can grow organically as new sources and use cases emerge.

Visual Examples



Example 1: GUI before any tasking


WIDE EYE
Keywords
Add Task
Active Tasks
Filter results...
Results will appear here...
Task Added
OK
Leaflet | © OpenStreetMap contributors

General

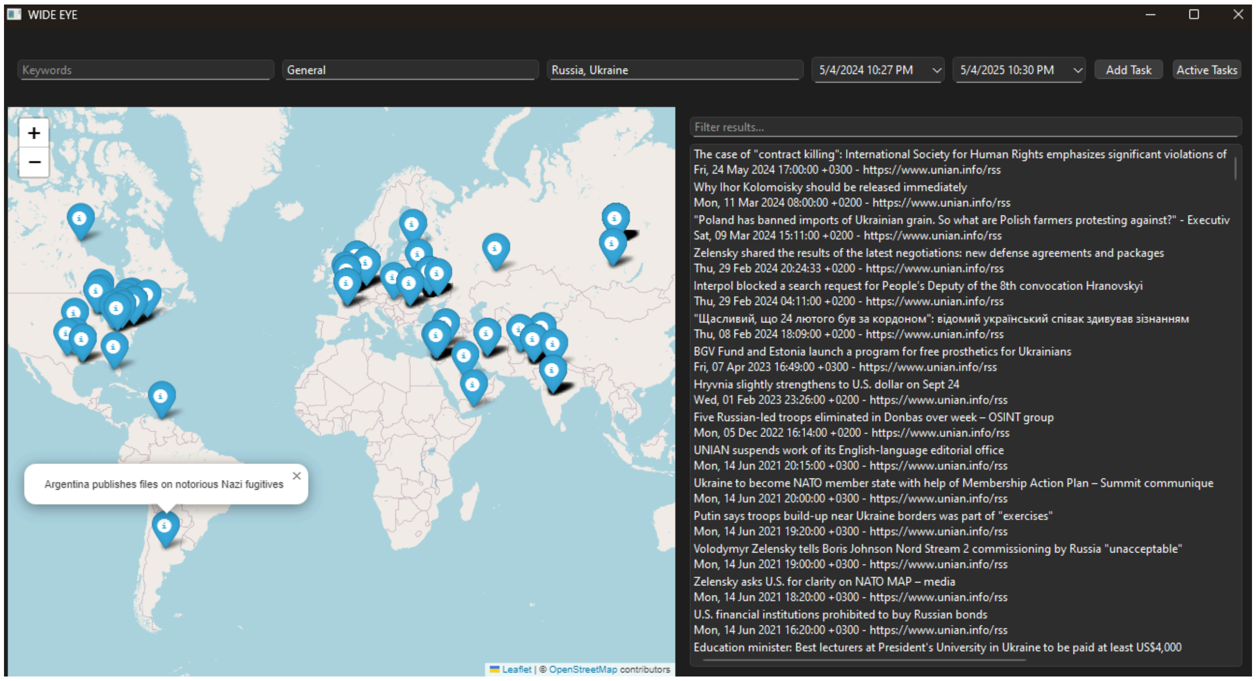
Russia, Ukraine

5/4/2024 10:27 PM

5/4/2025 10:30 PM

Task 5a366e0e-afbb-4d89-99e0-29f4ef5e1ac8 started; assigned 4/4 sources

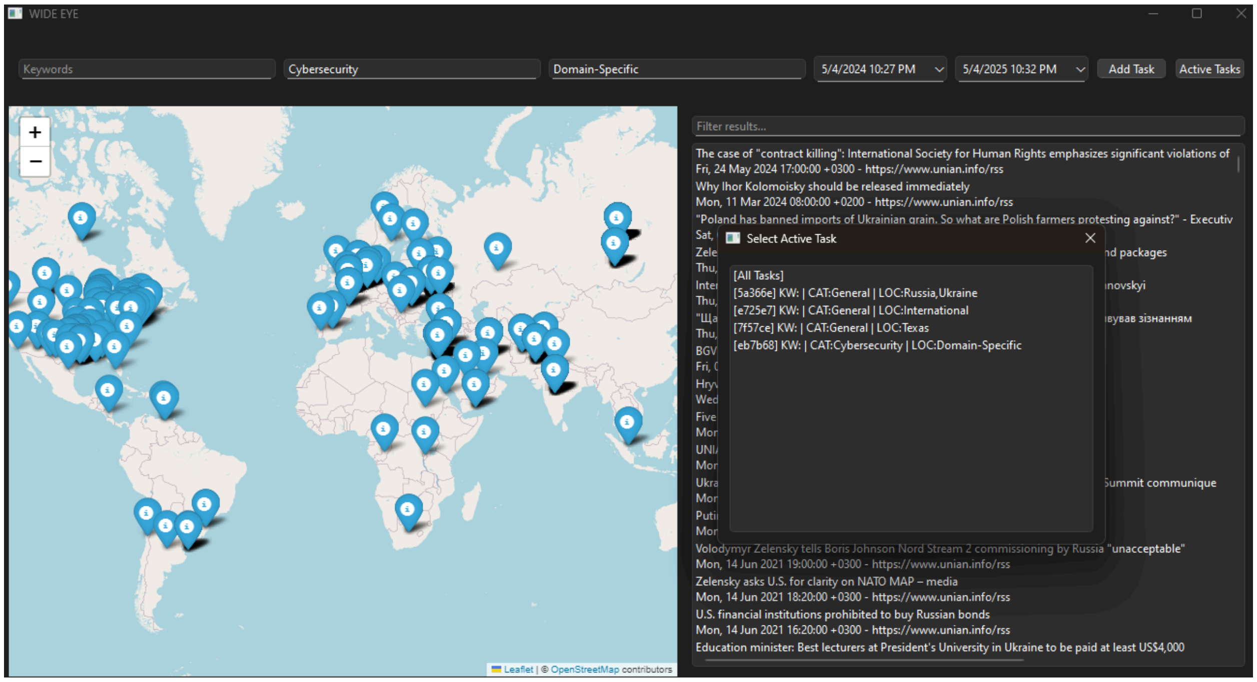
Example 2: First task (Russia/Ukraine general info) initiated



Filter results...

The case of "contract killing": International Society for Human Rights emphasizes significant violations of
Fri, 24 May 2024 17:00:00 +0300 - https://www.unian.info/rss
Why Ihor Kolomoisky should be released immediately
Mon, 11 Mar 2024 08:00:00 +0200 - https://www.unian.info/rss
"Poland has banned imports of Ukrainian grain. So what are Polish farmers protesting against?" - Executiv
Sat, 09 Mar 2024 15:11:00 +0200 - https://www.unian.info/rss
Zelensky shared the results of the latest negotiations: new defense agreements and packages
Thu, 29 Feb 2024 20:24:33 +0200 - https://www.unian.info/rss
Interpol blocked a search request for People's Deputy of the 8th convocation Hranovskyi
Thu, 29 Feb 2024 04:11:00 +0200 - https://www.unian.info/rss
"Щасливий, що 24 лютого був за кордоном": відомий український співак здивував зізнанням
Thu, 08 Feb 2024 18:09:00 +0200 - https://www.unian.info/rss
BGV Fund and Estonia launch a program for free prosthetics for Ukrainians
Fri, 07 Apr 2023 16:49:00 +0300 - https://www.unian.info/rss
Hryvnia slightly strengthens to U.S. dollar on Sept 24
Wed, 01 Feb 2023 23:26:00 +0200 - https://www.unian.info/rss
Five Russian-led troops eliminated in Donbas over week – OSINT group
Mon, 05 Dec 2022 16:14:00 +0200 - https://www.unian.info/rss
UNIAN suspends work of its English-language editorial office
Mon, 14 Jun 2021 20:15:00 +0300 - https://www.unian.info/rss
Ukraine to become NATO member state with help of Membership Action Plan – Summit communique
Mon, 14 Jun 2021 20:00:00 +0300 - https://www.unian.info/rss
Putin says troops build-up near Ukraine borders was part of "exercises"
Mon, 14 Jun 2021 19:20:00 +0300 - https://www.unian.info/rss
Volodymyr Zelensky tells Boris Johnson Nord Stream 2 commissioning by Russia "unacceptable"
Mon, 14 Jun 2021 19:00:00 +0300 - https://www.unian.info/rss
Zelensky asks U.S. for clarity on NATO MAP – media
Mon, 14 Jun 2021 18:20:00 +0300 - https://www.unian.info/rss
U.S. financial institutions prohibited to buy Russian bonds
Mon, 14 Jun 2021 16:20:00 +0300 - https://www.unian.info/rss
Education minister: Best lecturers at President's University in Ukraine to be paid at least US$4,000

Example 3: Results from first task populate map and result window



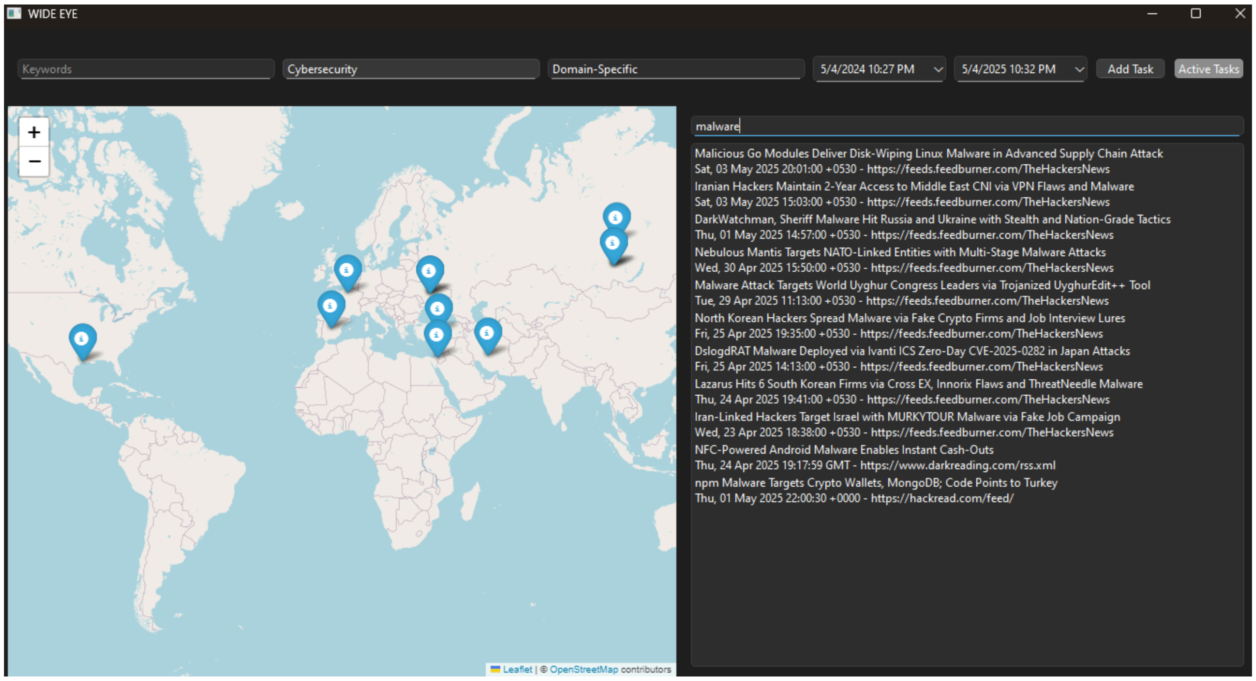
Filter results...

The case of "contract killing": International Society for Human Rights emphasizes significant violations of
Fri, 24 May 2024 17:00:00 +0300 - https://www.unian.info/rss
Why Ihor Kolomoisky should be released immediately
Mon, 11 Mar 2024 08:00:00 +0200 - https://www.unian.info/rss
"Poland has banned imports of Ukrainian grain. So what are Polish farmers protesting against?" - Executiv
Sat,
Zele nd packages
Thu,
Inter novskyi
Thu,
"Ща вував зізнанням
Thu,
BGV
Fri,
Hryv
Wed
Five
Mon
UNI
Mon
Ukra Summit communique
Mon
Puti
Mon
Volodymyr Zelensky tells Boris Johnson Nord Stream 2 commissioning by Russia "unacceptable"
Mon, 14 Jun 2021 19:00:00 +0300 - https://www.unian.info/rss
Zelensky asks U.S. for clarity on NATO MAP – media
Mon, 14 Jun 2021 18:20:00 +0300 - https://www.unian.info/rss
U.S. financial institutions prohibited to buy Russian bonds
Mon, 14 Jun 2021 16:20:00 +0300 - https://www.unian.info/rss
Education minister: Best lecturers at President's University in Ukraine to be paid at least US$4,000

Select Active Task

[All Tasks]
[5a366e] KW: | CAT:General | LOC:Russia,Ukraine
[e725e7] KW: | CAT:General | LOC:International
[7f57ce] KW: | CAT:General | LOC:Texas
[eb7b68] KW: | CAT:Cybersecurity | LOC:Domain-Specific

Example 4: Four tasks simultaneously populating map, prior to filtering



Malicious Go Modules Deliver Disk-Wiping Linux Malware in Advanced Supply Chain Attack
Sat, 03 May 2025 20:01:00 +0530 - https://feeds.feedburner.com/TheHackersNews
Iranian Hackers Maintain 2-Year Access to Middle East CNI via VPN Flaws and Malware
Sat, 03 May 2025 15:03:00 +0530 - https://feeds.feedburner.com/TheHackersNews
DarkWatchman, Sheriff Malware Hit Russia and Ukraine with Stealth and Nation-Grade Tactics
Thu, 01 May 2025 14:57:00 +0530 - https://feeds.feedburner.com/TheHackersNews
Nebulous Mantis Targets NATO-Linked Entities with Multi-Stage Malware Attacks
Wed, 30 Apr 2025 15:50:00 +0530 - https://feeds.feedburner.com/TheHackersNews
Malware Attack Targets World Uyghur Congress Leaders via Trojanized UyghurEdit++ Tool
Tue, 29 Apr 2025 11:13:00 +0530 - https://feeds.feedburner.com/TheHackersNews
North Korean Hackers Spread Malware via Fake Crypto Firms and Job Interview Lures
Fri, 25 Apr 2025 19:35:00 +0530 - https://feeds.feedburner.com/TheHackersNews
DslogdRAT Malware Deployed via Ivanti ICS Zero-Day CVE-2025-0282 in Japan Attacks
Fri, 25 Apr 2025 14:13:00 +0530 - https://feeds.feedburner.com/TheHackersNews
Lazarus Hits 6 South Korean Firms via Cross EX, Innorix Flaws and ThreatNeedle Malware
Thu, 24 Apr 2025 19:41:00 +0530 - https://feeds.feedburner.com/TheHackersNews
Iran-Linked Hackers Target Israel with MURKYTOUR Malware via Fake Job Campaign
Wed, 23 Apr 2025 18:38:00 +0530 - https://feeds.feedburner.com/TheHackersNews
NFC-Powered Android Malware Enables Instant Cash-Outs
Thu, 24 Apr 2025 19:17:59 GMT - https://www.darkreading.com/rss.xml
npm Malware Targets Crypto Wallets, MongoDB; Code Points to Turkey
Thu, 01 May 2025 22:00:30 +0000 - https://hackread.com/feed/

Example 4: Filter by cybersecurity task, followed by filter on “malware” keyword in results