Politecnico di Torino

Ioc on Telegram

Authors: Simone Capriolo s342664, Alessandro Pispola s343720

Contents

1 Introduction

This section introduces the motivation and structure of the project, which investigates the presence of Indicators of Compromise (IoCs) on Telegram messages and proposes a practical detection and response pipeline.

1.1 Investigating IoCs in Telegram Messages

Telegram has become a fast and informal channel where cybersecurity-related content is frequently shared. These contents may include potential Indicators of Compromise such as malicious URLs, suspicious domains, IP addresses and file hashes. Such IoCs can also appear in non-cyber-related groups, where users may unknowingly share compromised links, malicious files, or scam content. This project is built on a simple research question: *do the collected Telegram messages contain IoCs? Can we reliably identify and validate them at scale?*

The investigation is performed on the *GroupMonitoringRelease* dataset, containing messages collected from Telegram groups and stored in MongoDB. Since IoCs may appear in noisy, multilingual and unstructured text, the analysis requires a robust extraction technique followed by a validation step. For this reason, the project combines message filtering, IoC extraction, and external enrichment to validate results.

1.2 Objectives and Scope

The main objective is to detect IoCs inside Telegram messages and assess whether they are potentially malicious. The pipeline is organized into three stages. First, language models such as DistilBERT and SecureBERT are explored and fine-tuned to filter messages that could contain security-relevant content. Second, IoCs are extracted using a hybrid approach that combines the Python `iocsearcher` library with custom regular expressions. Third, extracted IoCs are enriched through VirusTotal to validate findings and to support the analysis with threat intelligence data.

In addition to the offline analysis, the project includes an operational component aimed at real-time Telegram group monitoring. A Telegram bot and a Wazuh-based SIEM/SOAR setup are used to detect IoCs posted in monitored groups. This architecture generates alerts, and triggers an automated response (user ban) when malicious IoCs are identified.

1.3 Project Overview and Design Choices

The project is organized as an end-to-end pipeline that connects offline analysis with an operational monitoring prototype. The report is structured to reflect this progression: data collection and preprocessing , NLP-based filtering, comparison between DistilBERT and SecureBERT, IoC extraction, VirusTotal enrichment, and the integration with Wazuh for automated response (Telegram Bot).

Two design choices are central. First, machine learning is used mainly as a filtering layer: it helps reduce the volume of messages to inspect and supports the comparative analysis between general-purpose and security-oriented language models. Second, the operational deployment adopts a deterministic, regex-based detection strategy because it is lightweight, easy to debug, and less prone to ignore IoCs due to model misclassification.

2 Background

This chapter provides the concepts required to understand the project pipeline. It first introduces what Indicators of Compromise (IoCs) are. It then presents threat-intelligence enrichment through VirusTotal as a validation layer to add evidence to extracted IoCs. Finally, it summarizes the SIEM/SOAR concepts used in the operational part of the project, with a focus on Wazuh and active response.

2.1 Indicators of Compromise (IoCs)

An indicator of compromise (IoC) in computer forensics is an artifact observed on a computer network or within an operating system that, with high confidence, indicates a computer intrusion [1]. Common examples include: malicious IP addresses, domains and URLs hosting malware or phishing pages, file hashes that uniquely identify malicious binaries, and references to known vulnerabilities.

IoCs are useful because they can be directly used in detection and response workflows. Once an indicator is extracted, it can be searched in logs, used to generate alerts, or blocked to reduce exposure to malicious content.

A practical challenge is that IoCs in chat messages are frequently inserted in noisy text and may be intentionally or unintentionally obfuscated (e.g., `hxxp://`, `example[.]com`) to avoid clickable links or automated moderation. For this reason, the extraction stage must be robust to variations in format and may require normalization rules before applying pattern matching.

2.2 Threat Intelligence Enrichment with VirusTotal

Detecting an IoC is not sufficient to conclude that the content is malicious. Many strings can match the same format as an IoC but still be benign (e.g., public IPs, legitimate URLs, or hashes related to clean files). For this reason, IoC extraction is typically followed by an enrichment step, where each candidate indicator is checked against external threat-intelligence sources. Enrichment adds context such as known detections, threat labels, and historical observations. In this project, VirusTotal is used as the main enrichment service to support validation. VirusTotal analyses suspicious files, domains, IPs and URLs to detect malware and other breaches, automatically sharing them with the security community [2].

VirusTotal aggregates results from multiple antivirus engines and threat-intelligence providers and returns structured information depending on the IoC type (URL, domain, IP address, or file hash). In general, the enrichment output can be

summarized into: detection signals (how many engines flag the indicator), classification labels (e.g., phishing, malware hosting, suspicious), and contextual metadata.

In our pipeline, each extracted IoC is queried on VirusTotal via API and the returned report is saved together with the message containing the IoC. This makes it possible to separate likely benign indicators from suspicious ones and to summarize the overall results. Since different engines can produce different verdicts, VirusTotal is used as supporting evidence rather than as absolute ground truth.

2.3 SIEM/SOAR Concepts

Security information and event management (SIEM) is a field within computer security that combines security information and security event management to enable real-time analysis of alerts generated by applications [3].

SOAR, for security orchestration, automation and response, is a software solution that enables security teams to integrate and coordinate separate security tools, automate repetitive tasks and streamline incident and threat response workflows [4].

In this project, these concepts are implemented using Wazuh, an open-source security platform commonly used for centralized log collection, rule-based detection and automated response. Wazuh is deployed as the central component that receives security-relevant events generated by the Telegram monitoring pipeline and turns them into structured alerts.

3 Preprocessing, Fine-Tuning & Classification

This section explores the methodology used to construct the dataset for our classification model. We detail the selection of cybersecurity-specific sources and general conversational corpora to create a balanced training environment. Furthermore, we describe the preprocessing pipeline, including entity masking, to enhance the model's generalization and reduce noise.

3.1 Training Data Sources and Dataset Design

To train a message-level classifier able to identify security-relevant content, we built a dataset combining *cyber-related* text with *non-cyber* text. The goal of this design is to teach the model to discriminate between messages about threat-intelligence discussions (where IoCs and vulnerability references are likely to appear) and generic conversational content that represents background noise.

For the cyber-related class (with **label = 1**), we used the APTNER dataset. This is a cybersecurity-oriented corpus designed for named entity recognition of threat-related entities. APTNER is provided in a token-level format. So, we reconstructed complete sentences by concatenating tokens until a sentence boundary was found. A sentence was included in our positive set if it contained at least one token with a tag different from O (i.e., at least one annotated entity). This approach assumes that the presence of entities (e.g., malware names, threat actors) directly identifies the message as cyber-related. This helps the model distinguish technical discussions from generic text. Then, to enforce the coverage of vulnerability mentions, a small set of sentences containing CVE identifiers was also injected.

For the non cyber-related class (with **label = 0**), we combined two sources of general text. The SMS Spam dataset which contains informal messages and the GoEmotions dataset which contains Reddit comments. These datasets were selected to mimic the typical style of real chats, while being unrelated to cybersecurity. Finally, the two classes were balanced through sampling to reduce bias during training and to make evaluation metrics more meaningful.

3.2 Cleaning and Masking

Before fine-tuning, all messages went through a normalization process to reduce noise and prevent the model from overfitting to specific indicators. This step begins with the removal of emojis and the collapsing of extra whitespace to ensure input consistency.

The core of our preprocessing involves a set of masking rules that replace sensitive or highly specific patterns with dedicated placeholders. This ensures the model learns the structural context of a threat rather than memorizing individual indicators. The mapping includes:

- CVE identifiers → `[CVE]`
- Telegram invite links → `[TG_LINK]`
- Generic URLs → `[URL]`
- IPv4 addresses → `[IP]`
- Common domain patterns → `[DOMAIN]`

This transformation allows the classifier to generalize across different security discussions by focusing on the semantic relevance of the masked entities.

3.3 Model exploration: DistilBERT & SecureBERT

The DistilBERT and SecureBERT models were selected to compare two complementary approaches. DistilBERT was chosen as it provides a strong baseline for text classification on short and noisy messages. SecureBERT was chosen because it is pretrained on cybersecurity-related text, and is therefore expected to represent IoC-related patterns more effectively. By comparing these models, we evaluate whether domain-specific pretraining offers some advantages over a faster generic baseline.

DistilBERT is a smaller, faster and more efficient version of BERT. It is pretrained on large-scale generic text and provides strong baseline capabilities for text classification. Its main advantages are speed, lower memory usage, and easier deployment in practical scenarios. However, since its pretraining is not specialized for cybersecurity, it may struggle with domain-specific terminology and with rare cybersecurity patterns.

SecureBERT is a cybersecurity-oriented model pretrained on security-related text sources. This kind of pretraining helps the model to represent technical language, threat reports, and IoC-like structures more effectively than generic models. In principle, this makes SecureBERT a better candidate for recognizing security-relevant content in messages that contain indicators, exploit references, or attacker terminology. The trade-off is that domain-specific models are often heavier than general-purpose alternatives and may generalize less to completely non-technical text.

3.4 Fine-tuning & Inference on Telegram messages

After constructing the fine-tuning dataset, we fine-tuned a classification head on top of the pretrained Transformer to perform **binary classification** (security-relevant vs non-security). The best checkpoint was selected based on validation performance.

The fine-tuned models are not used to directly label IoCs as malicious or benign. They are used instead only to **filter** Telegram messages (security-relevant vs non-security). This was done to reduce the amount of Telegram text processed by the IoC extraction phase. Furthermore, to increase the probability of observing IoCs in real data, we focused on **12** Telegram groups already related to cybersecurity from the *GroupMonitoringRelease* database. These are some names of the 12 groups: `Darknet`, `VirusCheck Chat`, `Only Dark`, `HTTP Injector`.

Before inference, for each message in the selected groups, the same preprocessing was applied during fine-tuning. Messages are processed in batches and, for each message, the model outputs a probability score for the positive class. A message is considered cyber-related only if its score exceeds a fixed threshold of 0.6. In such a way, the resulting dataset contains only the most security-relevant messages.

The obtained dataset was enriched with additional metadata to support threat intelligence interpretation. These fields include group information (name, topic, language, country), message identifiers, timestamp, user identifier, and a small amount of surrounding context (few preceding messages and one following message).

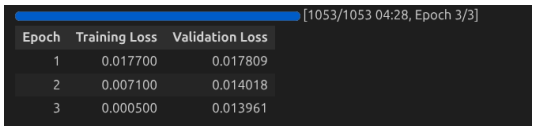

Figure 1: DistilBERT

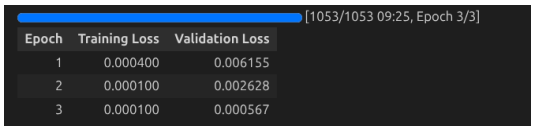

Figure 2: SecureBERT

Figure 3: Training and validation loss curves over 3 epochs for both models.

The training progress was monitored by tracking the loss reduction over 3 epochs, as shown in Figure 3. Both models demonstrate stable convergence with no signs of overfitting, as the validation loss consistently decreases alongside the training loss. Notably, SecureBERT achieves significantly lower absolute loss values (reaching a validation loss of 5.67×10^{-4}), confirming that its domain-specific pretraining allows for a more accurate adaptation to the cybersecurity classification task compared to the general-purpose baseline.

3.5 Filtering Results on Telegram Groups

Applying the filtering stage to the 12 selected Telegram groups resulted in a substantial reduction of the original message stream. Out of a total of 774,272 messages, SecureBERT retained 367,457 messages as cyber-related ($\approx 47.46\%$), while DistilBERT retained 102,654 messages ($\approx 13.26\%$).

The difference is expected and can be explained by the pretraining of the models. As already said, SecureBERT is pretrained on cybersecurity-related corpora and so it is more sensitive to security terminology and patterns. As a consequence, it tends to assign higher cyber-related probabilities to a broader set of messages.

SecureBERT vs DistilBERT (thresholded selection at `cyber_score` $\geq$ 0.60). SecureBERT retains 367,457 messages, whereas DistilBERT retains 102,654. The overlap is 85,099 messages, with a union of 385,012. Most of DistilBERT selections are also retained by SecureBERT (82.90% coverage), while DistilBERT covers only 23.16% of SecureBERT selections, indicating that DistilBERT behaves largely as a stricter subset of SecureBERT. The disagreement is strongly asymmetric: 282,358 messages are retained only by SecureBERT versus 17,555 retained only by DistilBERT. Group-level analysis shows the largest positive gaps in *Only Dark* (155,579 vs 49,962) and *Tenevoy Darknet Chat* (70,883 vs 14,518), where SecureBERT retains substantially more content. SecureBERT tends to keep a large amount of contextual or conversational content that often contains no explicit indicators, because this model was pre-trained using security-related data. As a result, it is more prone to classify a message as security relevant.

4 IoC Extraction and Threat Validation

This section describes the transformation of security-relevant Telegram messages into actionable IoCs. It is outlined the use of a hybrid extraction method, followed by VirusTotal validation for reputation signals.

4.1 Extraction pipeline

After the filtering phase, the next step was to extract candidate IoCs from the messages. The extraction stage is designed to maximize recall while reducing obvious noise.

To reduce the noise, we decided to apply the whitelisting technique: any IoC containing a whitelisted domain (e.g., `youtube.com`, `google.com`, etc.) was discarded. The reason behind this choice is that many extracted strings matched the syntactic format of an IoC but were not useful for threat analysis (e.g., links to mainstream platforms or social networks). This step allowed to focus on less common and potentially suspicious indicators. However, the trade-off is that even if the whitelisting step reduces noise, it may also discard true malicious IoCs that abuse legitimate high-reputation platforms potentially increasing false negatives.

The next stage was to use **`iocsearcher` and custom regex** to effectively extract IoCs. `Iocsearcher` is a Python library which scans raw text and returns structured matches in the form `(type, value, position, match)`. It covers common IoC categories such as URLs and IP addresses in standard form. However, real Telegram messages often contain noisy formatting that `iocsearcher` is not able to detect. For this reason, we defined a set of custom regular expressions to improve robustness on frequent IoC formats (URLs, IPv4 addresses, hashes, and emails).

Finally, the output of the IoC extraction phase was organized as a tabular dataset, in which each row corresponds to one extracted IoC. For every IoC extracted, the fields `ioc_value`, `ioc_type`, and `extraction_source` (library vs custom regex) were added to the original dataset features. To avoid repeated processing of the same IoC, we decided to remove duplicate IoCs and so to keep a single record per unique `ioc_value`. This choice reduced redundancy and the cost of enrichment through VirusTotal queries.

Extraction Results

In Table 1, are reported the results.

Metric	SecureBERT-filtered	DistilBERT-filtered
Total IoCs extracted	32,214	16,449
Discarded by whitelist	20,869	6,759
IoCs detected by `iocsearcher`	11,168	9,473
IoCs detected by custom regex	177	217
Discarded duplicates	9,679	7,869
Final unique IoCs saved	**1,666**	**1,821**

Table 1: IoC extraction results before VirusTotal enrichment.

SecureBERT retains 1666 unique IoCs and DistilBERT 1821, with an overlap of 1277 and a union of 2210 (coverage: 70.13% of Distil is also in Secure, and 76.65% of Secure is also in Distil). At group level, the largest divergence appears in *Mikrotik-Training* (483 vs 316, $\Delta = +167$), while other groups show the opposite trend with Distil retaining more. Discordant sets are sizable (389 only-Secure vs 544 only-Distil) and exhibit different message profiles.

It is interesting that SecureBERT started from a larger set of security-relevant messages but ended with less unique IoCs. This is possible because retaining more cyber-related messages does not necessarily produce more *unique* IoCs. As previously discussed, SecureBERT tends to classify messages as security-relevant more aggressively.

IoC Composition by Topic. Beyond the quantitative count, it is crucial to analyze the *type* of indicators circulating in these communities. Figure 4 illustrates the distribution of the top-6 extracted IoC types across the three main topics. This breakdown reveals distinct behavioral patterns:

- **Darknet:** This category displays a mixed profile, characterized by the high prevalence of `phoneNumber` (31.5%), `fqdn` (18.1%), and `telegramHandle` (16.8%). While contact details (phones and handles) act as "bridging" mechanisms to move negotiations to private encrypted chats, the significant presence of FQDNs is critical. These domains typically point to external illicit marketplaces or forums.

- **Software & Applications:** This topic is dominated by `url` (41.7%), consistent with the sharing of download links for tools and cracks. Notably, it also shows a significant presence of `uuid` (13.3%). In this context,

UUID-like strings likely correspond to software license keys, hardware IDs (HWID), or activation tokens exchanged in cracking communities.

- **Technologies:** As expected, this category is heavily skewed towards `url` (44.2%) and `fqdn` (23.1%), reflecting the sharing of technical resources, documentation, and external websites.

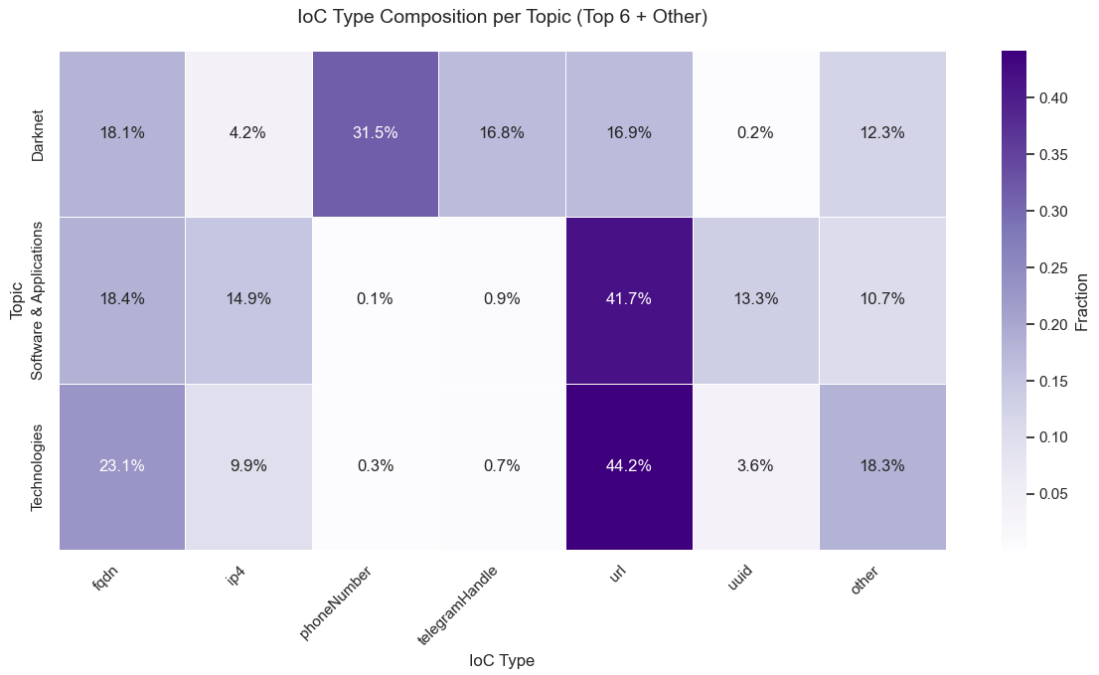

Figure 4: Distribution of Top-6 IoC types across Telegram topics (Union of extracted IoCs).

4.2 VirusTotal Enrichment Workflow and Interpretation

After extracting candidate IoCs, using VirusTotal API, a threat-intelligence enrichment was performed to assess whether the IoC were associated with known malicious activity. Since VirusTotal only supports enrichment for specific IoC, we restricted submissions to the categories handled by its API endpoints: IPv4 addresses, URLs, domains, and hashes (MD5, SHA1, SHA256).

Enrichment was performed on the 500 unique IoCs with the highest `cyber_score`. This choice was driven by the rate limits of the VirusTotal free plan (approximately 4 calls/minute). The responses were added to the dataset through these new fields: `vt_malicious` (number of engines flagging the IoC as malicious), `vt_total_engines` (total engines contributing to the analysis), `vt_scan_date` (timestamp of the latest available analysis) and `vt_permalink` (reference to the VirusTotal report).

It is crucial to note that the `vt_malicious` field is not a ground truth. In fact, it summarizes how many VirusTotal engines flagged the IoC, but false positives are still possible. For this reason, we used this field to prioritize IoC for investigation, rather than as a definitive statement of maliciousness. The results of this enrichment phase will be outlined in the next section.

To ensure reproducibility and facilitate future analysis, the final output of the pipeline is consolidated into a structured dataset. Table 2 details the complete schema, which links the original Telegram context (message text, user, group info) with the extracted threat intelligence. This comprehensive structure serves as a ready-to-use resource for both offline research and operational monitoring.

Feature	**Description**
group_id	Telegram group identifier.
chat_name	Group name.
country	Group country.
topic	Group topic/category.
context_pre	Short context preceding the target message (e.g., up to 3 previous messages).
msg_id	Telegram message identifier.
date	Message timestamp.
user_id	Sender identifier.
reply_to_msg_id	Identifier of the replied message (if the message is a reply).
text	Full message text.
context_next	Short context following the target message (e.g., the next message).
language	Group language.
cyber_score	Model score.
ioc_value	Extracted IoC string (e.g., URL, IP, domain, hash).
ioc_type	IoC category/type.
extraction_source	Source of extraction.
vt_malicious	Number of VirusTotal engines flagging the IoC as malicious.
vt_total_engines	Total number of VirusTotal engines considered in the report.
vt_scan_date	VirusTotal scan date.
vt_permalink	Link to the VirusTotal report.
vt_status	Status of the VirusTotal query (e.g., found/not found/error/rate-limited).

Table 2: Dataset Metadata

5 DistilBERT vs SecureBERT

This section presents the comparative analysis of the two models based on the threat intelligence validation performed via VirusTotal. While the previous filtering stage showed that SecureBERT retains significantly more messages than DistilBERT, this validation step aims to determine which model effectively captures actionable and malicious IoC.

5.1 Malicious IoC Overlap and Model Complementarity

The validation was performed on the top-500 extracted IoCs (ranked by `cyber_score`). A total of 52 unique malicious IoCs were identified across the union of both models. The results, visualized in Figure 5, reveal a little performance gap between the two approaches.

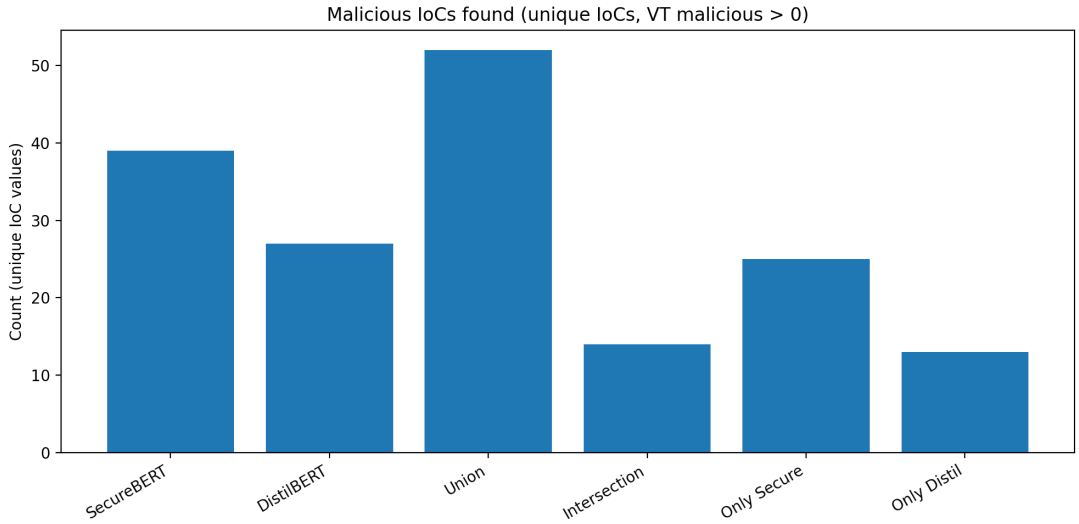

Figure 5: Unique malicious IoCs identified by Secure, Distil and their intersection.

SecureBERT identified **39** unique malicious IoCs, whereas DistilBERT identified **27**. Notably, the intersection between the two sets is limited to only **14** IoCs (26.9% of the union). This lack of overlap suggests that the models are highly complementary:

- **SecureBERT** detected 25 malicious IoCs that DistilBERT missed. This confirms the hypothesis that domain-specific pretraining allows the model to recognize security-relevant contexts that generic models might filter out as noise.

- **DistilBERT** detected 13 malicious IoCs missed by SecureBERT. These are likely found in shorter, less structured messages where the generic model's sensitivity to standard patterns (like isolated URLs) proved effective.

Consequently, to maximize detection coverage in an operational environment, a hybrid approach leveraging the union of both models is preferable to selecting a single best performer.

5.2 Distribution by Source and Type

Further analysis was conducted to understand the source and nature of the detected threats. As shown in Figure 6, the distribution of malicious IoCs is highly unbalanced across Telegram groups. Two groups alone, *Mikrotik-Training* (17 IoCs) and *HTTP Injector* (13 IoCs), account for the majority of the detections. These groups are technically oriented, often exchanging proxy lists and configuration scripts, which frequently trigger malware engines due to association with botnets or abusive infrastructure.

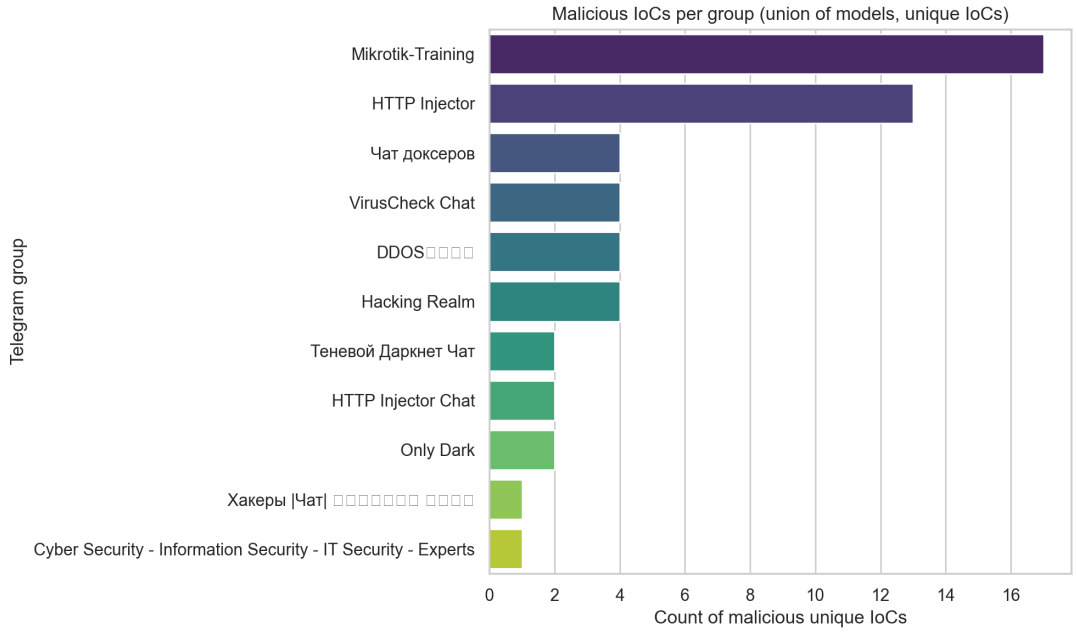

Figure 6: Distribution of malicious IoCs across monitored Telegram groups.

Finally, we analyzed the relationship between the group topic and the type of IoC detected (Figure 7). The heatmap highlights distinct behavioral patterns:

- **Darknet Topics:** The malicious indicators are predominantly Fully Qualified Domain Names (FQDNs, 60.9%). This aligns with the nature of dark-

net discussions, which often involve sharing suspicious domains or phishing links.

- **Software & Applications:** The threats are overwhelmingly IPv4 addresses (72.7%). This is consistent with the sharing of proxy servers, VPN endpoints, or C2 (Command and Control) IPs often found in groups dedicated to cracking tools and network utilities.

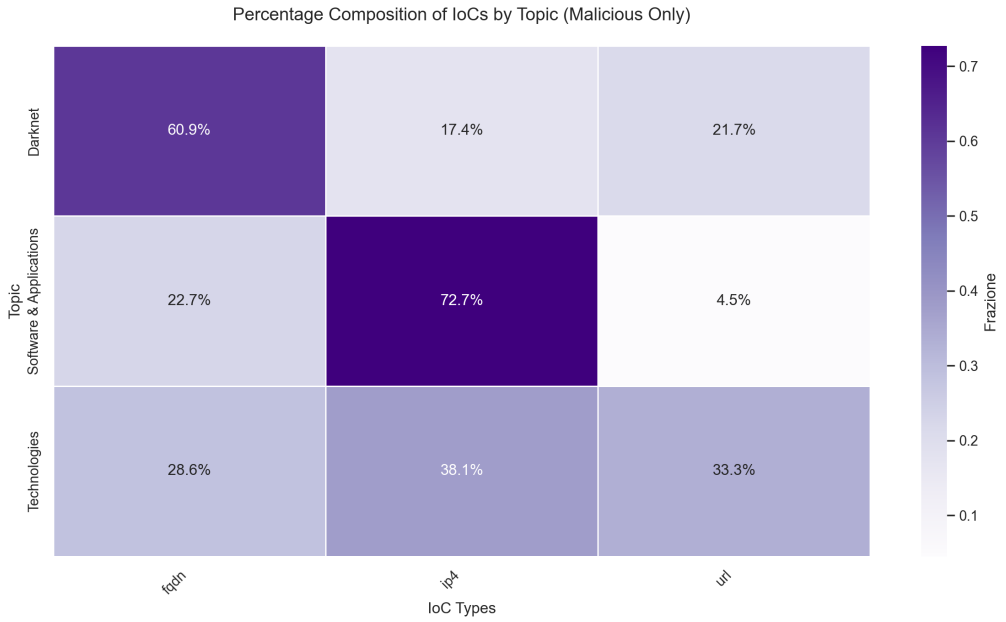

Figure 7: Percentage composition of malicious IoC types across different group topics.

6 Deployment: Telegram Bot + Wazuh

The goal of this phase was to move from an offline setting to an operational CTI workflow. The previous AI-driven pipeline told us that potentially malicious IoCs were effectively present in the monitored channels. So, it supported the decision to invest in an automated, real-time system.

Based on these findings, we implemented an end-to-end deployment that continuously monitors Telegram messages, extracts candidate IoCs, validates them through VirusTotal and triggers a response through Wazuh.

6.1 End-to-end operational flow

The system implements a CTI loop that turns Telegram messages into actionable security events. Figure 8 summarizes the full chain.

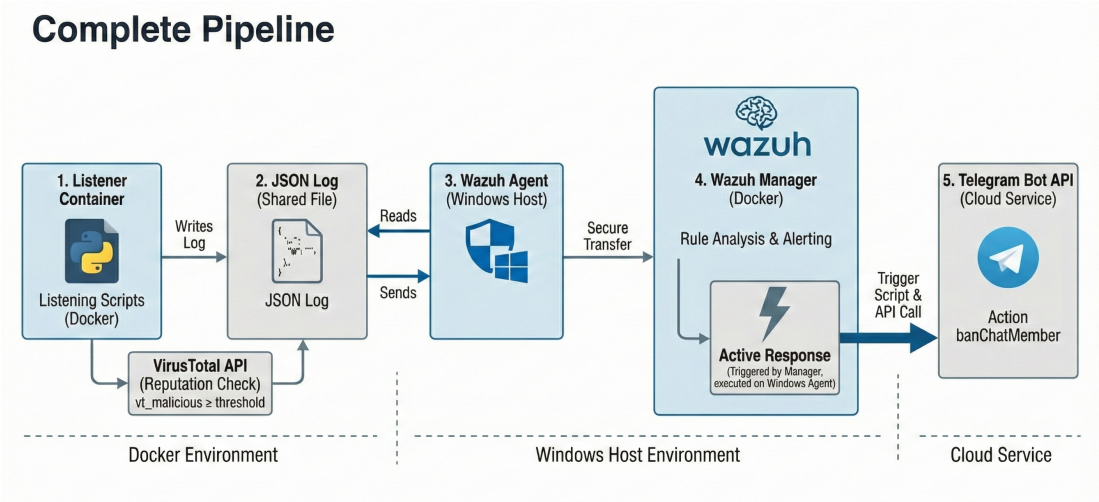

Figure 8: Complete Operational Pipeline

Python Listener + Virus Total

The listener, running inside a Docker container, relies on the **Telegram APIs** (via `Telethon` Python library) to authenticate as a client and receive new messages from the monitored group.

Every new message in the group is immediately inspected using regexes to capture IoCs. The listener uses the **VirusTotal API** to enrich extracted IoCs, obtaining a reputation signal used to validate them. From the response produced by VirusTotal, the listener extracts the number of engines labeling the IoC as `malicious` and the total number of engines. If the number of engines exceeds the threshold, the listener writes a structured security event inside the

`virustotal_results.json`. This file is crucial as it acts as a bridge, allowing the Wazuh Agent to read events generated from the listener.

Furthermore, to reduce the number of requests to Virus Total and respect rate-limiting constraints, a cache that stores the extracted IoCs is implemented. So, every time there is a new IoC, the listener performs a look-up in the cache to check if that IoC and the corresponding result are already stored. In order to guarantee the freshness of threat intelligence data and to check eventual changes on IoCs reputation, the cache is automatically cleared every 24 hours.

Wazuh Agent + Manager

The core of the detection and orchestration phase relies on the interaction between the **Wazuh Agent**, deployed on the Windows host, and the **Wazuh Manager**, running within the Docker environment. This relationship transforms logs into security actions through a multi-step communication process.

1 - Data Collection and Transmission: The Wazuh Agent configuration was modified by adding a `<localfile>` block to its `ossec.conf` file:

```
<localfile>
  <location>C:\Users\...\logs\virustotal_results.json</location>
  <log_format>json</log_format>
</localfile>
```

This allows the agent to monitor the specific output (`virustotal_results.json`) of the listener. Once a new log int he `json` file is detected, the agent forwards the data to the Wazuh Manager via a secure channel.

2 - Centralized Logic and Decision Making: When the `json` log arrives from the agent, the Wazuh Manager processes the event through its analysis engine. It determines the nature of the event by consulting a custom ruleset implemented in the `local_rules.xml` file. This logic is structured in two levels:

```
<group name="telegram,soc,">
  <rule id="100010" level="3">
    <decoded_as>json</decoded_as>
    <field name="integration_source">^telegram_sentinel$</field>
    <description>Message analyzed in $(source_chat).</description>
  </rule>

  <rule id="100011" level="12">
    <if_sid>100010</if_sid>
    <field name="virustotal.malicious" type="pcre2">^[1-9][0-9]*$</field>
    <description>
```

```
      CTI ALERT: Malicious IoC detected!
      (Score: $(virustotal.malicious)/$(virustotal.total_engines)) - $(ioc)
    </description>
    <group>active_response_trigger,</group>
  </rule>
</group>
```

The logic is the following one:

- **Rule 100010 (Event Classification):** This rule acts as a filter. It identifies any incoming log where the `json integration_source` field matches "telegram_sentinel". This value is set by the listener when it populates the `virustotal_results.json` file. The rule was set to Level 3, as it simply logs that an analysis has occurred without indicating a threat.
- **Rule 100011 (Threat Detection):** This is a child rule (indicated by `<if_sid>`) that only triggers if the parent rule is met. The engine uses a regular expression (`^[1-9][0-9]*$`) to inspect the `virustotal.malicious` field. This expression checks if the number of malicious engines is 1 or greater. If true, the alert level is elevated to 12, and the event is tagged with the `active_response_trigger` group, which is used to initiate the automated ban.

3 - Active Response Orchestration: The most critical function of the Manager is the automation of the countermeasure. To enable this, the Manager's `ossec.conf` was extended with a custom `<command>` and an `<active-response>` policy:

- **Command Definition:** The `<command>` block defines what can be executed. By naming it `win-telegram-ban` and pointing to `telegram_ban.cmd`, we created an instruction that the Manager can send to the agent.
- **The Active Response Trigger:** The `<active-response>` block defines when and where the command should run. By linking the command to `rules_id 100011` and setting the location to `local`, the Manager is instructed that whenever a malicious IoC is confirmed, it must immediately send an execution order back to the specific Agent that reported the event.

This is the `telegram_ban.cmd`:

```
@echo off
"C:\...\python.exe" "C:\...\active-response\bin\telegram_ban.py"
```

This file acts as a Windows bridge for Wazuh Active Response. When the Manager triggers the `win-telegram-ban` command, the Agent executes this script, which simply invokes the local Python interpreter and runs `telegram_ban.py` to perform the ban action.

Telegram Bot

The final stage of the SOAR pipeline is the execution of the banning script. When the Wazuh Agent executes the `telegram_ban.cmd` file, it passes the entire `json` payload to the `telegram_ban.py` Python script. The script extracts three critical variables:

- `target_user_id`: The unique identifier of the malicious sender.
- `target_chat_id`: The group where the threat was detected.
- `ioc_detected`: The specific malicious IP or URL.

To prevent bans, the script implements a local whitelist. Before calling the Telegram API, the script checks the `user_id_clean` against a predefined list of administrator and the authorized bot. If a match is found, the script logs the event as a "Whitelist Activated" and terminates without taking action.

The Ban Command and Notification: If the user is not whitelisted, the script performs the following final steps:

1. **API Call:** It sends a POST request to the `banChatMember` endpoint of the Telegram Bot API. This is a definitive administrative action that removes the user and prevents them from re-joining.
2. **Auditable Logging:** Every execution step is recorded in the local log file `telegram_ban_execution.log`. This ensures that every automated ban is traceable for post-incident review.
3. **Feedback Loop:** Upon a successful ban, the bot sends a notification message back to the group: *"BAN EXECUTED - IoC detected: [value]"*. This informs the other members that the threat has been neutralized, providing transparency to the automated security operations.

6.2 Operational examples from the Telegram pipeline

This section provides execution trace of the deployed workflow. The example is based on a message containing an IP address, which is extracted by the listener, enriched via VirusTotal, written to `virustotal_results.json`, and then ingested by the Wazuh Agent/Manager according to the rules described in Fig. 8.

1) Telegram evidence: message and automated feedback

Figure 9 shows the user message that contains the candidate IoC (the IP address), followed by the bot notification posted back to the group after the response is executed.

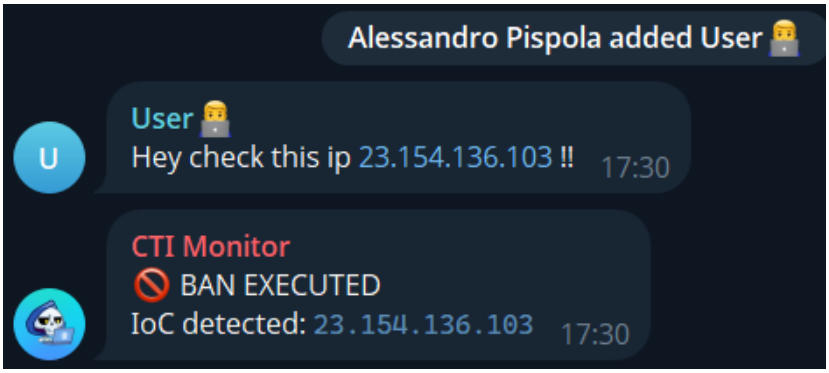

Figure 9: Telegram example of malicious IoC.

2) Dockerized listener: extraction, enrichment, and JSON write

After receiving the new message, the Dockerized listener performs regex-based IoC extraction and starts the enrichment step. Figure 10 shows the moment in which the IoC is flagged as malicious and the structured event is written to `virustotal_results.json`.

```
C:\Users\apisp\OneDrive\Desktop\PoliTo\II Anno\DPA\IoC_on_Telegram\Telegram_Listener>docker logs -f sentinel_bot
[INFO] 2026-02-04 16:26:50,580 - Connecting to 149.154.167.91:443/TcpFull...
[INFO] 2026-02-04 16:26:50,644 - Connection to 149.154.167.91:443/TcpFull complete!
[INFO] 2026-02-04 16:30:20,563 - 📩 New message from Project_DPA (ID: 6507584304)
[INFO] 2026-02-04 16:30:20,594 - 🔎 Found 1 IoC(s). Starting scan...
[WARNING] 2026-02-04 16:30:22,403 - 🚨 MALICIOUS DETECTED: 23.154.136.103 (1/93)
[INFO] 2026-02-04 16:30:22,418 - ✅ Event written to /app/logs/virustotal_results.json
```

Figure 10: IoC extraction and VirusTotal enrichment and JSON event write.

3) Structured event in `virustotal_results.json`

The listener writes a single JSON record per validated IoC. Figure 11 shows the event structure used as a bridge between the listener and the Wazuh Agent.

Figure 11: Example JSON event written by the listener.

It's useful to take a look at the `json` fields previously desrcibed:

- `integration_source`: fixed tag set by the listener to `telegram_sentinel`).
- `source_chat`: name of the monitored group.
- `chat_id`: numeric identifier of the group where the IoC was observed.
- `author_id`: numeric identifier of the sender (used as `target_user_id` by the ban script).
- `message_snippet`: short summary of the message for contextual auditing.
- `ioc`: extracted indicator value (e.g., the IP address shown in the message).
- `ioc_type`: indicator category (e.g., `ip`, `url`).
- `virustotal.malicious`: number of engines marking the IoC as malicious, used by Rule 100011.
- `virustotal.total_engines`: total number of engines considered, used to build the alert score string.

4) Active response execution on the host

Once Wazuh confirms the alert (Rule 100011), the Manager triggers the active response and the Agent executes the ban procedure locally. Figure 12 shows the ban script receiving the event, extracting the target identifiers, and performing the action. The log also highlights the whitelist safeguard: if the `user_id` matches a protected identity, the ban is cancelled.

```
[2026/02/04 17:30:26] --- SCRIPT STARTED ---
[2026/02/04 17:30:26] Waiting for data from STDIN...
[2026/02/04 17:30:26] Data received (first 100 chars): {"version":1,"origin":{"name":"node01","module":"wazuh-execd"},"command":"add","parameters":{"extra_
[2026/02/04 17:30:26] Target Acquired -> User: 6507584304, Chat: -4886220791
[2026/02/04 17:30:27] Ban Result: {'ok': True, 'result': True}
[2026/02/04 17:30:28] --- SCRIPT STARTED ---
[2026/02/04 17:30:28] Waiting for data from STDIN...
[2026/02/04 17:30:28] Data received (first 100 chars): {"version":1,"origin":{"name":"node01","module":"wazuh-execd"},"command":"add","parameters":{"extra_
[2026/02/04 17:30:28] 🛡️ WHITELIST ACTIVATED: User 8525446251 protected. Ban cancelled.
```

Figure 12: Ban script execution trace: ban attempt and whitelist protection.

5) Alert visibility in Wazuh

Finally, Figure 13 shows the alert generated by the custom ruleset in the Wazuh dashboard. The event appears with Rule ID `100011` and an elevated severity level, including the VirusTotal score and the detected IoC value.

Figure 13: Wazuh security events view.

6.3 Security considerations

A key concern of a real-time monitoring bot is the trust boundary introduced by deployment choices. If the "brain" of the bot runs on an external server, message contents and metadata may be processed and potentially stored outside the Telegram group owner's control. This raises privacy issues, since the monitoring infrastructure would have full visibility of the chat traffic.

To mitigate this risk, we designed the system to be deployable locally by the group owner or by the administrator. The full stack is distributed as a Dockerized solution (including the Wazuh single-node deployment and the Telegram listener), so it can be executed on-premise on the owner's host. In this setup, messages are still necessarily read and analyzed by the bot to support detection and banning actions, but the processing and logs remain under the direct control of the operator running the stack, avoiding third-party centralization.

In addition, the implementation follows the *need-to-know* data-minimization principles: only the useful information are kept (e.g., timestamps, stable identifiers, IoC value, and enrichment outcome), while sensitive content is minimized and can be reduced further if required. Finally, credentials such as Telegram and VirusTotal tokens are injected through environment variables rather than being hardcoded, and log retention can be configured to limit long-term exposure.

7 Conclusions

This project shows that modern social platforms such as Telegram can represent a valuable source of CTI. In practice, public and semi-public groups often act as informal channels where attack traces circulate in near real time. From a defensive (blue-teaming) perspective, these streams can be exploited to discover emerging IoCs earlier than traditional feeds. At the same time, the same ecosystem can be exploiting by attackers (red-teaming) as a delivery vector, for instance through malicious links and social-engineering messages.

As cybersecurity engineers, starting from this observation, we implemented a protection mechanism in the form of a Telegram bot that can be deployed by a group owner/admin to add an additional layer of security.

Overall, the project demonstrates how CTI collection, enrichment, alerting, and SOAR-style enforcement can be integrated into a lightweight and reproducible architecture.

7.1 Limitations

One limitation of the enrichment stage is the reliance on VirusTotal as the main reputation signal. VirusTotal aggregates multiple engines, but its results are not ground truth. As a consequence, the `vt_malicious` field should be interpreted as a indicator rather than a definitive label.

A second limitation is the restricted coverage of IoC types supported by the VirusTotal API. In our dataset, additional patterns were observed (e.g., phone numbers, telegramHandler and uuid) but only IPv4 addresses, URLs, domains, and file hashes could be enriched. This constraint reduces the ability to analyze certain social-engineering artifacts that may still be relevant in real scenarios.

From an operational standpoint, the current prototype tests a single monitored group and a limited set of extraction rules (IPv4 and URLs). Extending coverage to more groups, additional indicator types, and richer context would likely improve investigative value but also increases complexity and the risk of spurious detections.

Another limitation concerns the training strategy adopted for the text classifier. In our experiments, we fine-tuned only the classification head on top of the frozen pre-trained Transformer. While this choice reduces computational cost and training time, it also limits the model's ability to adapt its internal representations to the Telegram CTI domain. A stronger alternative would be to unfreeze and fine-tune at least the last two Transformer layers, allowing the model to improve performance.

Lastly, model capacity is also a limitation. We relied on a compact architecture such as DistilBERT which may be less expressive than larger Transformer encoders. This choice was mainly driven by practical constraints, since our available hardware resources were limited and we needed a model that could be trained and deployed reliably within a reasonable time.

7.2 Future Work

A natural extension is to improve the enrichment layer by combining multiple intelligence sources and verification mechanisms. For example, IoCs could be cross-checked with additional feeds (open CTI repositories, abuse databases, passive DNS, URL scanning services). This would reduce dependence on a single provider and improve confidence in the final verdict.

Another direction is to evolve the system into a reusable CTI asset. Indicators and correlated metadata that triggered bans could be stored in a structured repository to build a local threat ledger, enabling deep analysis, trend monitoring, and the creation of group-specific blocklists. This would transform the pipeline from a reactive mechanism into a continuously improving knowledge base.

References

[1] Wikipedia. *Indicator of compromise*. URL: https://en.wikipedia.org/wiki/Indicator_of_compromise (visited on 01/27/2026).

[2] Virus Total. *Virus Total*. URL: https://www.virustotal.com/gui/home/upload.

[3] Wikipedia. *Security information and event management*. URL: https://en.wikipedia.org/wiki/Security_information_and_event_management (visited on 01/27/2026).

[4] IBM. *What is SOAR (security orchestration, automation and response)?* URL: https://www.ibm.com/think/topics/security-orchestration-automation-response (visited on 01/27/2026).