DarkGram: A Large-Scale Analysis of Cybercriminal Activity Channels on Telegram

This repository includes the following:

1) Dataset: A collection of 53,605 posts from 329 channels spanning five different cybercriminal activity channels (CACs): Credential Compromise, Pirated Software, Blackhat Resources, Pirated Media, and Social Media Manipulation. Additionally, 32,443 replies from 75 channels (as most channels do not allow replies) are included. This dataset was used to characterize the content shared by these CACs.

2) Labeling and Categorization Scripts: Scripts used to label and categorize the dataset with GPT-4.

3) Model training: A combination of the CAC posts in **1)** plus 171,221 benign posts sourced from the Pushshift Telegram dataset, and a script to build the BERT-base classifier using the training dataset.

4) DarkGram Framework: The DarkGram framework for identifying and reporting new Telegram CACs.

The following is the file structure of the repository:

Repository Tree

```
├── darkgram_framework
│   ├── channel_list.txt
│   ├── config
│   │   ├── email_config.json
│   │   └── tg_auth.json
│   ├── find_new_channels.py
│   ├── model
│   │   ├── config.json
│   │   ├── merge_into_complete_model.py
│   │   ├── model_part_001.part
│   │   ├── model_part_002.part
│   │   ├── model_part_003.part
│   │   ├── model_part_004.part
│   │   └── model_part_005.part
│   └── report_channels.py
├── data
│   ├── posts.zip
│   └── replies.zip
├── misc_scripts
│   └── gpt4_labelling
│       ├── gpt4_social_media_manipulation_platform.py
│       ├── gpt4_software_categorization.py
```

```
│       ├── gpt4_software_price.py
│       ├── sample_smm.csv
│       └── sample_softwares.csv
├── requirements.txt
└── training_model
    ├── train_darkgram.py
    └── training_data.zip
```

DarkGram dataset

CAC posts shared in channels:

Location: data/posts.zip

The ‘post’ folder contains five subfolders, inside each of which there are several {channel_name}.csv files. Each of them contains posts collected from the respective channel.

Table 1: The datasheet for the CAC posts is described below:

Variable	Type	Description
Channel ID	Numeric	The unique identifier of the Telegram channel where the post was published.
URL	Text	The link to the Telegram channel.
Post ID	Numeric	A unique numeric identifier for each post in the channel.
Date	Datetime	The date and time (in ISO 8601 format) when the post was created or last edited.
Message	Text	The textual content of the post, including any links or embedded media.
Out	Boolean	Boolean flag indicating if the post was forwarded from another source (TRUE) or sent by the channel owner (FALSE).
Mentioned	Boolean	Boolean flag indicating whether the post mentions other users or channels.
Media_Unread	Boolean	Boolean flag indicating if the post contains unread media for the viewer.
Silent	Boolean	Boolean flag indicating if the post was sent without a notification.
Post	Boolean	Boolean flag indicating if this record is a post (TRUE) or another type of content.
From_Scheduled	Boolean	Boolean flag indicating if the post was published from a scheduled queue.
Legacy	Boolean	Boolean flag indicating if the post uses legacy formatting.

Edit_Hide	Boolean	Boolean flag indicating whether edits to the post are hidden.
Pinned	Boolean	Boolean flag indicating if the post is pinned in the channel.
NoForwards	Boolean	Boolean flag indicating if forwarding of the post is restricted.
Peer_Channel	Text	The identifier of the peer (associated) channel.
From_ID_User	Numeric	The unique identifier of the user who created the post.
Fwd_From	Text	Indicates the source if the post is forwarded from another channel or user.
Via_Bot_ID	Numeric	The ID of the bot, if the post was created via a bot.
Reply_to_Msg_ID	Numeric	The ID of the message this post is replying to.
Reply_to_Scheduled	Boolean	Indicates if the post is replying to a scheduled message.
Forum_Topic	Boolean	Indicates whether the post is part of a forum topic.
Media_Photo_ID	Numeric	The unique identifier of any photo/media associated with the post.
Reply_Markup	Text	Metadata for any interactive elements (e.g., buttons) included in the post.
Views	Numeric	The number of views the post received.
Forwards	Numeric	The number of times the post was forwarded.
Replies	Numeric	The number of replies the post received.
Edit_Date	Datetime	The timestamp of the last edit to the post.
Post_Author	Text	If applicable, the author of the post (useful for multi-user channels).
Grouped_ID	Numeric	The ID of the group this post belongs to, if part of a grouped series of posts.
Reactions	Text	Metadata about user reactions to the post (e.g., emojis, like/dislike counts).
Restriction_Reason	Text	Describes any restrictions applied to the post.
TTL_Period	Numeric	Time-to-live period for the post, if applicable.

Note that, posts containing personall indicators for posts in Credential Compromise CACs were obfuscated using Microsoft Presidio (https://github.com/microsoft/presidio).

Replies to CAC posts:

Location: data/replies.zip

The replies folder contains five subfolders, inside each of which there are several subfolders (channel_name), and inside each of these subdirectories there are {post_id}.csv files which contain the replies to a particular post in that channel.

Table 2: The datasheet for the CAC posts is described below:

Variable	**Type**	**Description**
ID	Numeric	A unique numeric identifier for each reply.

Channel ID	Text	The unique identifier of the Telegram channel where the reply was published.
URL	Text	The link to the Telegram channel.
Message	Text	The textual content of the reply, including any links or emojis.
Date	Datetime	The date and time (in ISO 8601 format) when the reply was created.
Views	Numeric	The number of views the reply received.
Forwards	Numeric	The number of times the reply was forwarded.
Reactions	Emoji:Numeric	Metadata about user reactions (emojis) to the reply

GPT annotation, training scripts

Location: misc_scripts/

Throughout this paper, we have utilized GPT-4 to annotate a large volume of Telegram CAC posts. The repository includes the following annotation scripts:

- **`gpt4_software_categorization.py`**: Categorizes pirated software.
- **`gpt4_software_price.py`**: Estimates the price of pirated software.
- **`gpt4_social_media_manipulation_platform.py`**: Identifies platforms targeted by Social Media Manipulation CAC posts.

Prerequisites

To run these scripts, you need an OpenAI API key, which can be obtained from: https://platform.openai.com/docs/quickstart

Add your API credentials to the file `configs/openai.json`

Usage Instructions

Software Annotation Scripts:

- For `gpt4_software_categorization.py` or `gpt4_software_price.py`, provide a list of software as a command-line argument.
- A sample dataset, `sample_softwares.csv` (derived from Pirated Software CAC posts), is included in the repository.

Syntax:
```
python3 gpt4_software_categorization.py sample_softwares.csv
python3 gpt4_software_price.py sample_softwares.csv
```

Social Media Manipulation Script:

- For `gpt4_social_media_manipulation_platform.py`, provide a list of social media posts as a command-line argument.
- A sample dataset, `sample_smm_posts.csv` (derived from Social Media Manipulation CACs), is included in the repository.

Syntax: `python3 gpt4_social_media_manipulation_platform.py sample_smm_posts.csv`

DarkGram Training

Location: training_model/

The **`training_model`** folder contains:

Training Data:

A `training_data` folder with individual files for each of the five CACs. Each file includes all posts from channels associated with that CAC.

A `benign.csv` file with 171,221 benign posts sourced from the Pushshift Telegram dataset.

Data Structure:

The CAC dataset includes the same columns as described in Table 1.

The benign dataset contains only the post ID and the post text, as the classifier is trained exclusively on post text.

Training Script:

The `train_darkgram.py` script is used to train the BERT-based classifier model.

Usage Instructions:

First you need to unzip the training_data.zip file.

Install all dependencies from requirements.txt

Then run the training script to start training the model:

`python3 train_darkgram.py`

DarkGram framework

Location: darkgram_framework/

The **DarkGram** framework is designed to identify new CAC channels on Telegram. It operates by scanning channel links shared within other Telegram channels, analyzing their posts, and determining whether they are malicious.

Files in the Framework

The repository includes the following key files:

- **`darkgram_framework/drivers/find_new_channels.py`**: Identifies new malicious Telegram channels.
- **`darkgram_framework/drivers/report_channels.py`**: Reports newly identified malicious Telegram channels to Telegram's abuse email (`abuse@telegram.org`).

Framework Workflow

Seed Channels: Provide a list of seed channels (in the format `t.me/{channel_name}`). The framework scans posts in these channels to extract links to other Telegram channels.

1. **Channel Analysis**:
    - For each new channel identified, the framework fetches the first 10 posts.
    - These posts are analyzed using the **DarkGram classifier**.
    - If 5 or more posts are classified as malicious, the channel is flagged as malicious and reported to Telegram.
2. **Data Storage**:
    - The framework stores:
        - Newly found channels in `new_channels.csv`.
        - Posts from scanned channels in `csvs/{channel_id}`.
        - Names of shared attachments in `drops/{channel_id}/drops.csv`.

> **Warning**: Only download attachments in a secure and isolated environment, as they may contain malware. Avoid downloading `.txt` files, as they could contain sensitive user information.

Prerequisites

Seed Channel List:

Create a file `channel_list.txt` containing URLs of seed channels in this format:
vbnet
CopyEdit
`https://t.me/{channel1}`
`https://t.me/{channel2}`

Seed channels can be obtained directly from Telegram or data aggregation sites such as Telemetrio.

Telegram API Access:

Obtain credentials for the Telegram API from Telegram Core.

Add the credentials to `config/tg_auth.json` in the following format:

```
{
  "api_id": "<api_id>",
  "api_hash": "<api_hash>",
  "phone": "<your_phone>",
  "username": "<telegram_username>"
}
```

Email Configuration:

To report malicious channels, configure your SMTP email in `configs/email_config.json`.

Install all required dependencies:

```
pip3 install -r requirements.txt
```

Build the Full Model:

Merge the model shards into a complete model in the ‘model’ folder by running:

```
python3 merge_into_complete_model.py
```

Running the Framework

Run the channel detection script:

```
python3 find_new_channels.py
```

Output:

Newly found channels are saved in `new_channels.csv`.

Posts from analyzed channels are stored in `csvs/{channel_id}`.

Attachment names from these channels are saved in `drops/{channel_id}/drops.csv`.

Logs are saved in `darkgram.log` for monitoring and debugging.