PEDOGRAM$_{prv}$

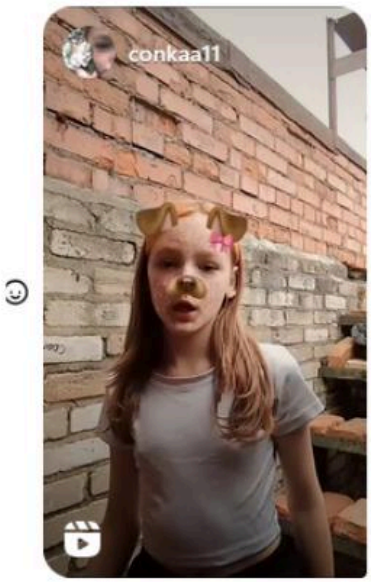
This report follows algorithm and the techniques used by services such as ‘instagram reels’ which feeds back content based of various interactions with the user on a specific video or topic, it is easy for use and documents the findings and eventual exposure of a large pedophile ring operating through Instagram and META services- following and taking down those using Instagram's reel content and recommendation algorithm to advertise, distribute and sell Child Sexual Abuse Material (CSAM) to others.

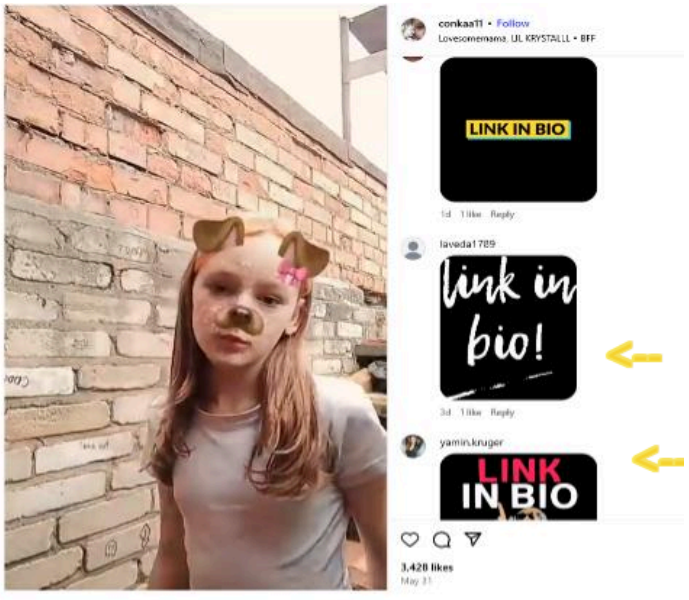
Findings

Due to the viewing and interactive based learning of similar nature in regards to content consumed and topics of interest to interact with each other and see more relevant videos that like minded people watch- this is evident in how the algorithm feeds users interested in gaming, more gaming related videos and thus more gamers interact with one another because they are now fed if not the same, but similar videos.

But sadly, due to the nature of the internet gaming isn't the only thing that catches peoples interest- this algorithmic design combined with a lack of proper content reporting and moderation- regardless of whether a user clicks ‘interested’ or ‘not interested’ on a video allows for users to quickly spread, view and interact with individuals who watch potentially exploitative videos of underage people, all while fed by an algorithm which promotes it due to a ‘positive interaction’ with that type of content which these people provide (eg, liking, commenting or watching the video for long periods of time) d

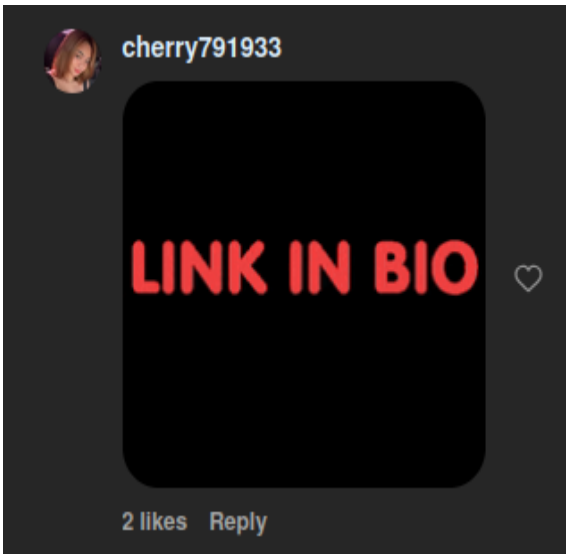
An example of this-
take this seemingly innocent video of what appears to be a minor found on instagram reels, surely the comment section would be restricted and abusive content would be shunned out by the ‘safe’ algorithm designed to stop abuse… right?





Wait… why are the comments filled with people sending gifs saying ‘Link in bio’- and others saying things such as; “Trade” and “exchange”- well you guessed it, these people are all using these videos suggested by the algorithm to distribute, advertise and sell exploitative and elicit material containing children.

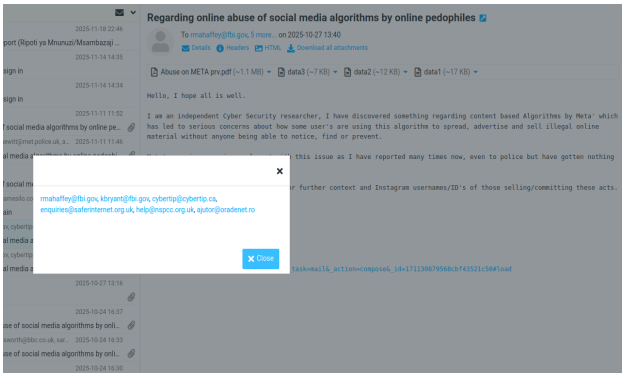
This is evident in many of the accounts containing profile pictures and username information with the ‘Telegram’ logo in them (a common platform for distributing this type of content) or accounts with ‘boy’, ‘girl’, ‘rsvp’ or even ‘cp’ within the profile, with nearly all the accounts using these ‘Link in BIO’ gifs containing these telegram links, an advertisement for other pedophiles to exchange their illicit imagery or ‘trade’ as is referred to in the comment section:

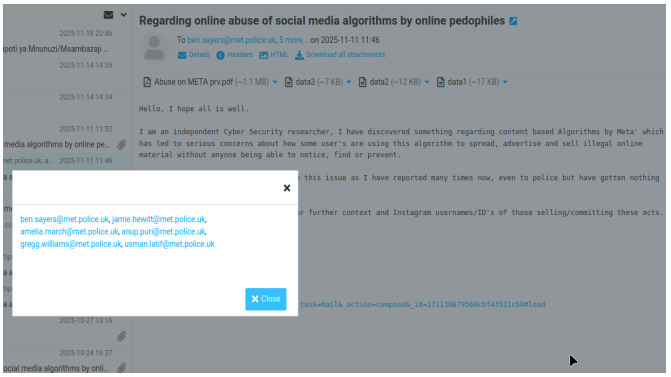


So why do they use these gifs so often?- well it isn't a coincidence that all these accounts are attempting to advertise the fact they are distributing and ‘trading’ CSAM this way, this is actually on purpose and for a reason- by using gifs and not comments as much, it makes it much harder for automated moderation and report bots to remove key words or certain language recognition from comments, allowing many of these to go unfiltered or reported by the algorithm.

Reporting

As soon as this issue was discovered, the scale and both likelihood of this issue already being aware to meta where assessed, and it was decided that due to the track record of META’s in compliance with these sort of issues and actions taken with similar types of abuse on their platform in the past, a joint report being made to FBI, NSPCC and MET organisations would be the best way to spread word in this issue and show extent:





Many other reports like these were written to many other law enforcement, linked social media and child safety advocate organisations in hopes of a response or potential action being taken.

Although this is the extent of reporting, further action was taken by myself to ensure this network was shed to light and taken down.

Mitigation

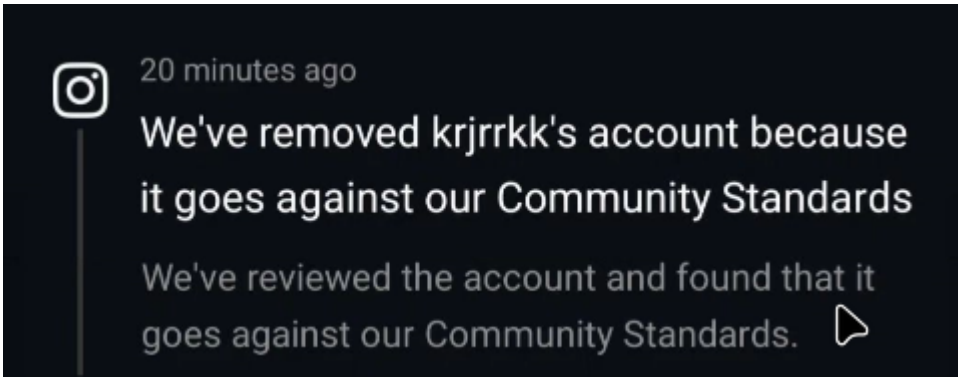
During general testing, found that many of these accounts were linked in nature, and all contained very similar characteristics about them, namely:

- Many of the accounts were still considered ‘new’ to META’s automated abuse detection system (account created >= 2023)

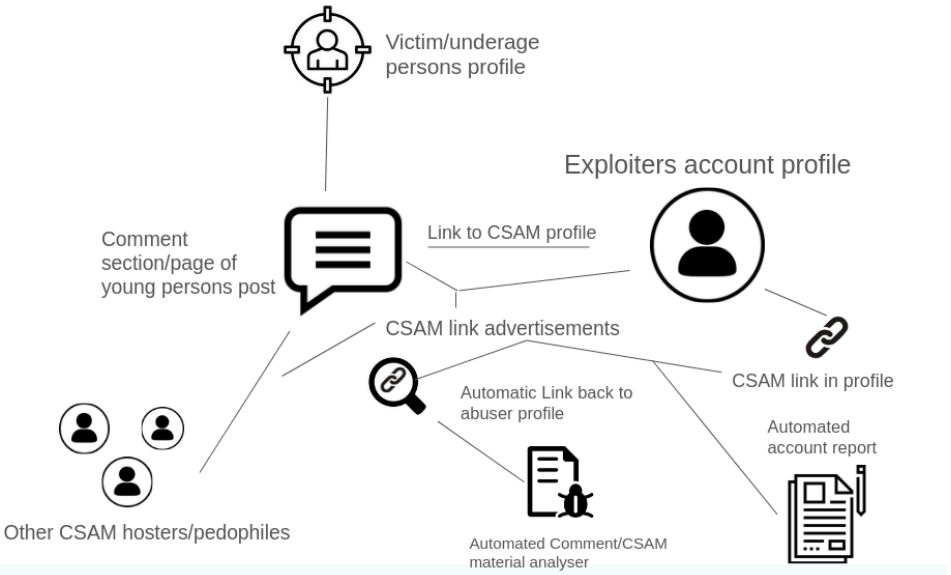
- Nearly all these accounts contained ‘Telegram’ links or shortened links in their BIOs or their account ‘link to’ section where most of the CSAM abuse material was hosted.

- Accounts did not have much social interaction or positive reputation score on the platform (EG: no or very little posts, followers, following, etc..)

And that if these accounts were reported as ‘Spam’ or ‘Fraudulent’, due to the nature of how the platform's account reputation and filtering system works, rather than reported as ‘exploitative or sexual’ in nature, then they would be automatically removed by META’s platform.



So an attack vector plan was drawn up to mass filter out these accounts, and have as many of them ‘falsely’ flagged and reported as possible as that was the most effective way to remove the harmful material:



To make this attack process more effective, automation programs were written in order to assist in the identification and reporting of these malicious users, assisting with identifying accounts:

```python
def graphql_request(query_hash: str, variables: Dict, headers: Dict[str, str]) -> Dict:
    var_str = json.dumps(variables, separators=(",", ":"))
    url = (
        f"https://www.instagram.com/graphql/query/"
        f"?query_hash={query_hash}"
        f"&variables={requests.utils.quote(var_str)}"
    )
    r = requests.get(url, headers=headers)
    if r.status_code != 200:
        print(f"[!] HTTP {r.status_code}: {r.text[:300]}...")
        sys.exit(1)

    try:
        return r.json()
    except ValueError:
        print("[!] Invalid JSON from Instagram.")
        sys.exit(1)


def parse_comment_node(node: Dict, include_replies: bool = False) -> Dict:
    owner = node.get("owner", {})
    item = {
        "id": node.get("id"),
        "username": owner.get("username"),
        "text": node.get("text"),
        "like_count": node.get("like_count"),
        "created_at": node.get("created_at"),
    }

    if include_replies:
        reply_edges = node.get("edge_threaded_comments", {}).get("edges", [])
        item["replies"] = [{
            "id": r.get("node", {}).get("id"),
            "username": r.get("node", {}).get("owner", {}).get("username"),
            "text": r.get("node", {}).get("text"),
            "like_count": r.get("node", {}).get("like_count"),
            "created_at": r.get("node", {}).get("created_at"),
        } for r in reply_edges]

    return item
```

```python
def to_usernames(comments: List[Dict]) -> List[str]:
    seen = set()
    out = []
    for c in comments:
        u = c.get("username")
        if u and u not in seen:
            seen.add(u)
            out.append(u)
    return out


def usernames_with_duplicates(comments: List[Dict]) -> List[str]:
    return [c.get("username") for c in comments if c.get("username")]


def dedupe_by_key(items: Iterable[Dict], key: str) -> List[Dict]:
    seen = set()
    result = []
    for it in items:
        k = it.get(key)
        if k and k not in seen:
            seen.add(k)
            result.append(it)
    return result


def write_output(data_format, file_format, output_path, comments, usernames=None):
    os.makedirs(os.path.dirname(output_path) or ".", exist_ok=True)

    if data_format == "usernames":
        payload = usernames
        if file_format == "json":
            with open(output_path, "w") as f:
                json.dump(payload, f, indent=2)
        elif file_format == "csv":
            with open(output_path, "w", newline="") as f:
                w = csv.writer(f)
                w.writerow(["username"])
                for u in payload: w.writerow([u])
        else:
            with open(output_path, "w") as f:
                for u in payload: f.write(u + "\n")
            return
```

Formatting and filtering the ones attempting to advertise and spread illicit material:

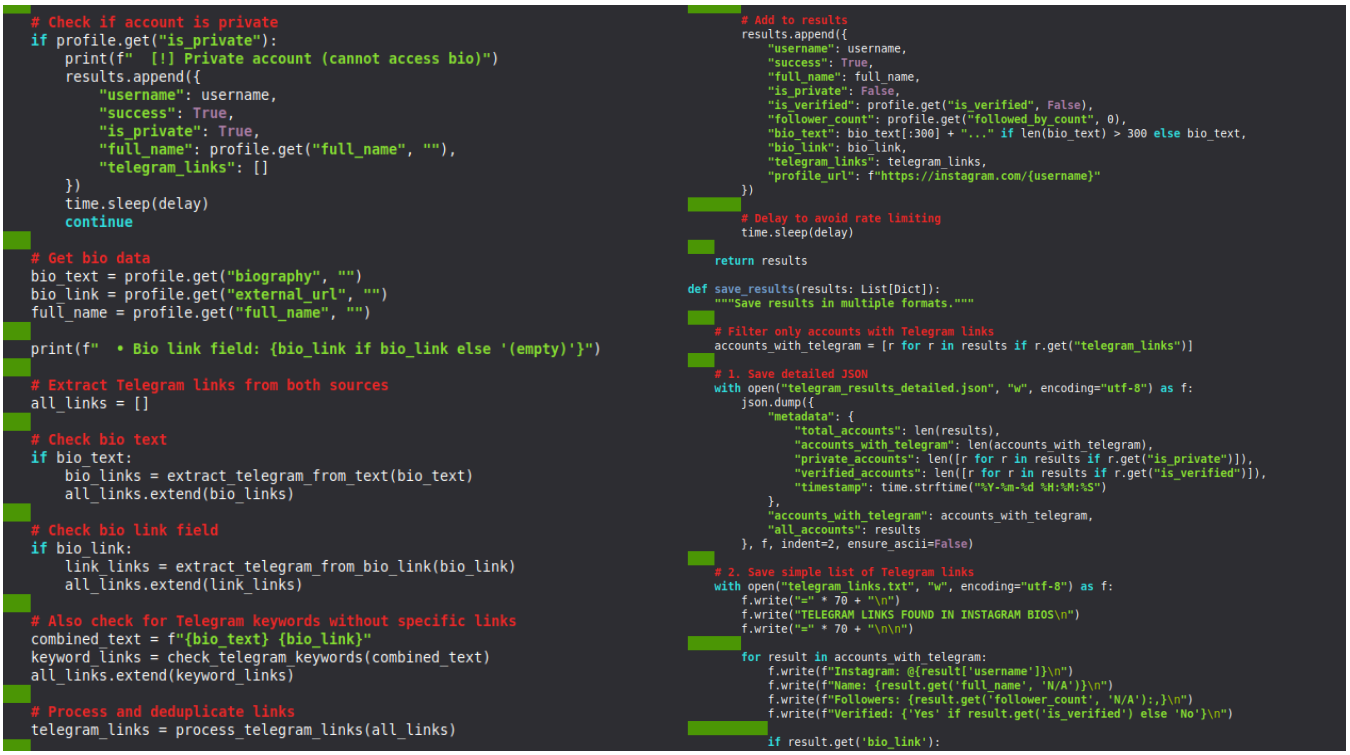
```python
    for pattern in TELEGRAM_URL_PATTERNS:
        match = re.search(pattern, url, re.IGNORECASE)
        if match and match.group(1):
            username = match.group(1)
            link = f"t.me/{username}"
            links.append((link, f"Bio link field: {url}"))

    # Also check if it's a link shortener that might point to Telegram
    # Common shorteners
    shorteners = ['bit.ly', 'tinyurl.com', 'cutt.ly', 'shorturl.at', 'ow.ly', 'is.gd', 'buff.ly', 'adf.ly']

    if any(shortener in url.lower() for shortener in shorteners):
        try:
            # Try to resolve the short URL
            response = requests.head(url, allow_redirects=True, timeout=5)
            final_url = response.url

            # Check if final URL is Telegram
            for pattern in TELEGRAM_URL_PATTERNS:
                match = re.search(pattern, final_url, re.IGNORECASE)
                if match and match.group(1):
                    username = match.group(1)
                    link = f"t.me/{username}"
                    links.append((link, f"Bio link redirects to: {final_url}"))
        except:
            pass

    return links

def check_telegram_keywords(text: str) -> List[Tuple[str, str]]:
    """Check for Telegram mentions without specific links."""
    links = []
    text_lower = text.lower()

    for keyword in TELEGRAM_KEYWORDS:
        if keyword in text_lower:
            # Find the context around the keyword
            keyword_idx = text_lower.find(keyword)
            if keyword_idx != -1:
                start = max(0, keyword_idx - 30)
                end = min(len(text), keyword_idx + len(keyword) + 30)
                context = text[start:end].replace('\n', ' ').strip()

                # Look for username pattern near the keyword
                username_pattern = r'[@\s]([a-zA-Z0-9_]{5,})'
                username_match = re.search(username_pattern, context)

                if username_match:
                    username = username_match.group(1)
                    link = f"t.me/{username}"
                    links.append((link, f"Telegram mention: ...{context}..."))
                else:
                    # Just note the mention without specific username
                    links.append(("Telegram mentioned", f"Context: ...{context}..."))
```

```python
# Telegram link patterns for BIO TEXT
TELEGRAM_TEXT_PATTERNS = [
    r't\.me/([a-zA-Z0-9_]+)',  # t.me/username
    r'telegram\.me/([a-zA-Z0-9_]+)',  # telegram.me/username
    r'telegram:?\s*(?:@?)([a-zA-Z0-9_]{5,})',  # telegram: username
    r'tg:?\s*(?:@?)([a-zA-Z0-9_]{5,})',  # tg: username
    r'contact:?\s*(?:t\.me/|telegram\.me/|@?)([a-zA-Z0-9_]+)',  # contact: t.me/username
    r'@([a-zA-Z0-9_]{5,})\s*(?:on telegram|on tg|telegram|tg)',  # @username on telegram
    r'(?:telegram|tg)\s*[@:]\s*([a-zA-Z0-9_]{5,})',  # telegram @username
    r'link:?\s*(?:t\.me/|telegram\.me/)([a-zA-Z0-9_]+)',  # link: t.me/username
]

# Telegram patterns for BIO LINK (URL field)
TELEGRAM_URL_PATTERNS = [
    r'(?:https?://)?t\.me/([a-zA-Z0-9_]+)',  # t.me/username
    r'(?:https?://)?telegram\.me/([a-zA-Z0-9_]+)',  # telegram.me/username
    r'(?:https?://)?telegram\.dog/([a-zA-Z0-9_]+)',  # telegram.dog/username
    r'(?:https?://)?telegram\.org/([a-zA-Z0-9_]+)',  # telegram.org/username
]

# General patterns to detect Telegram mentions
TELEGRAM_KEYWORDS = [
    'telegram', 'tg', 't.me', 'telegram.me', 'telegram.dog', 'telegram.org'
]

def build_headers() -> Dict[str, str]:
    """Build headers for Instagram requests."""
    return {
        "User-Agent": (
            "Mozilla/5.0 (Linux; Android 13; SM-A125F) AppleWebKit/537.36 "
            "(KHTML, like Gecko) Chrome/122.0.0.0 Mobile Safari/537.36"
        ),
        "Accept": "*/*",
        "Accept-Language": "en-US,en;q=0.9",
        "X-Requested-With": "XMLHttpRequest",
        "X-IG-App-ID": "936619743392459",
        "Cookie": f"sessionid={SESSIONID}; ds_user_id={DS_USER_ID}; csrftoken={CSRFTOKEN}; mid={MID};"
    }

def fetch_user_profile(username: str, headers: Dict[str, str]) -> Optional[Dict]:
    """Fetch Instagram user profile data including bio link."""
    try:
        # Try the web version which includes bio link
        url = f"https://www.instagram.com/{username}/?__a=1&__d=dis"
        response = requests.get(url, headers=headers, timeout=10)

        if response.status_code == 200:
```

Triaging and bypassing the attempts to obfuscate or redirect the malicious links through services like shorteners (commonly used to disguise telegram links from automated reporting):

```python
# Also check if it's a link shortener that might point to Telegram
# Common shorteners
shorteners = ['bit.ly', 'tinyurl.com', 'cutt.ly', 'shorturl.at', 'ow.ly', 'is.gd', 'buff.ly', 'adf.ly']

if any(shortener in url.lower() for shortener in shorteners):
    try:
        # Try to resolve the short URL
        response = requests.head(url, allow_redirects=True, timeout=5)
        final_url = response.url

        # Check if final URL is Telegram
        for pattern in TELEGRAM_URL_PATTERNS:
            match = re.search(pattern, final_url, re.IGNORECASE)
            if match and match.group(1):
                username = match.group(1)
                link = f"t.me/{username}"
                links.append((link, f"Bio link redirects to: {final_url}"))
    except:
        pass
```

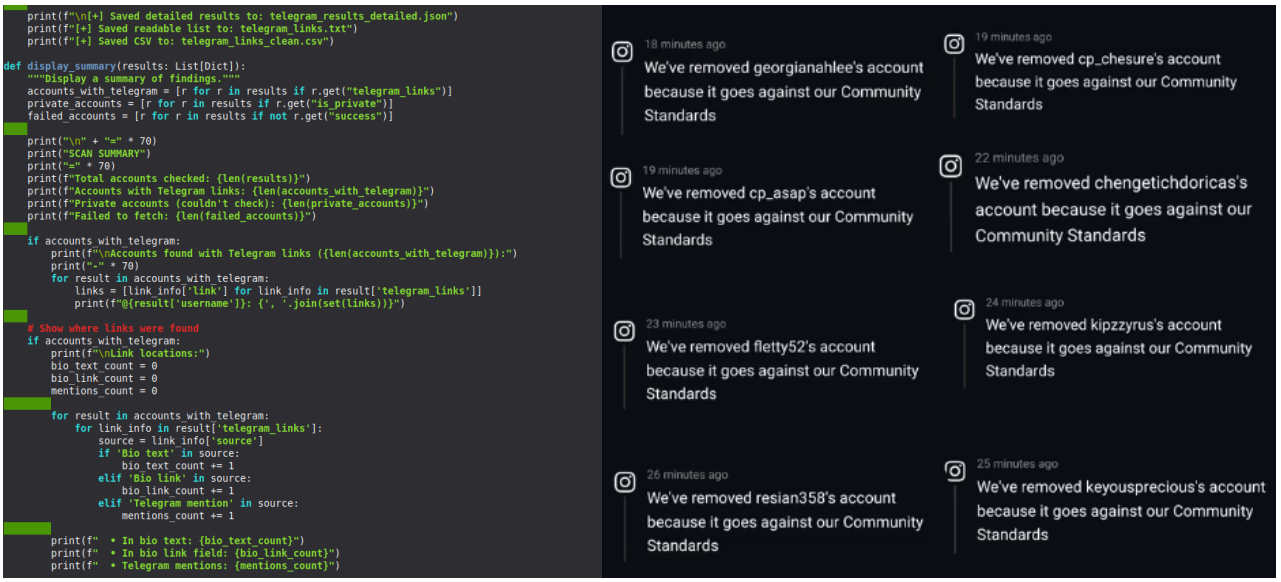
And gathering the appropriate information related to the accounts for the best chances of identifying, taking down and learning from the methods used:



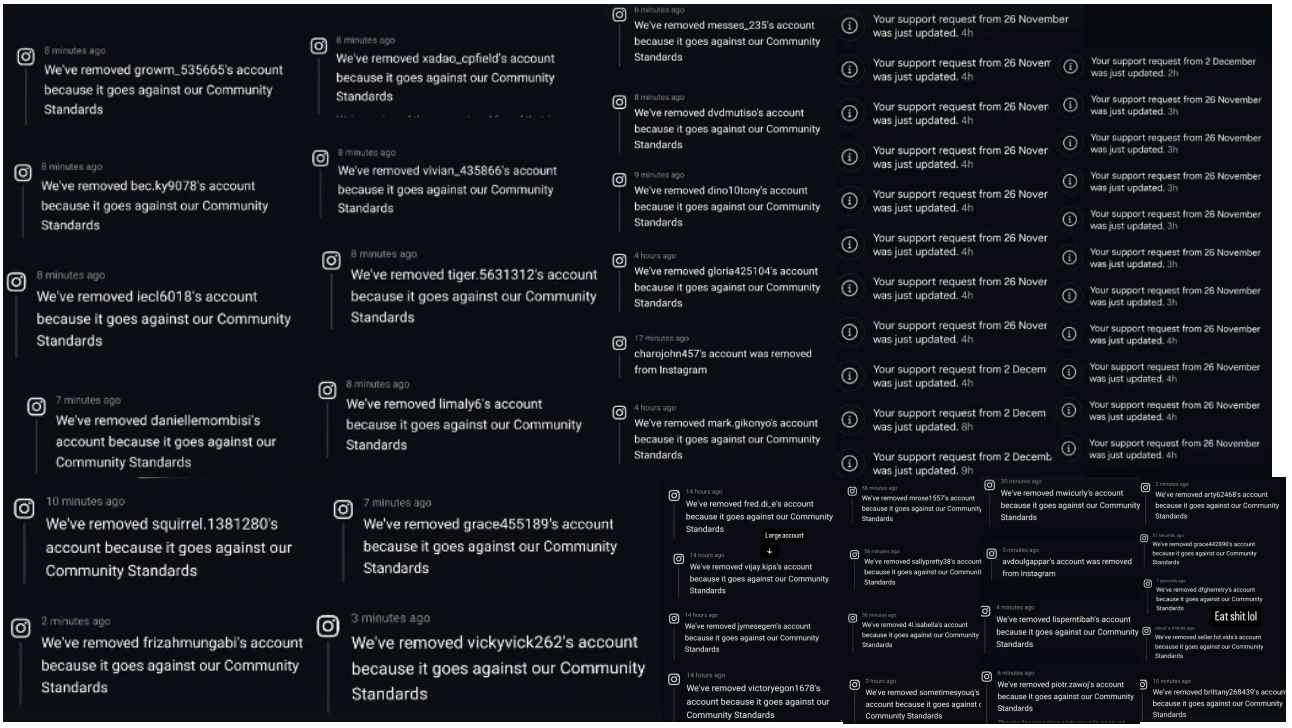
```python
    # Check if account is private
    if profile.get("is_private"):
        print(f"  [!] Private account (cannot access bio)")
        results.append({
            "username": username,
            "success": True,
            "is_private": True,
            "full_name": profile.get("full_name", ""),
            "telegram_links": []
        })
        time.sleep(delay)
        continue

    # Get bio data
    bio_text = profile.get("biography", "")
    bio_link = profile.get("external_url", "")
    full_name = profile.get("full_name", "")

    print(f"  • Bio link field: {bio_link if bio_link else '(empty)'}")

    # Extract Telegram links from both sources
    all_links = []

    # Check bio text
    if bio_text:
        bio_links = extract_telegram_from_text(bio_text)
        all_links.extend(bio_links)

    # Check bio link field
    if bio_link:
        link_links = extract_telegram_from_bio_link(bio_link)
        all_links.extend(link_links)

    # Also check for Telegram keywords without specific links
    combined_text = f"{bio_text} {bio_link}"
    keyword_links = check_telegram_keywords(combined_text)
    all_links.extend(keyword_links)

    # Process and deduplicate links
    telegram_links = process_telegram_links(all_links)
```

```python
        # Add to results
        results.append({
            "username": username,
            "success": True,
            "full_name": full_name,
            "is_private": False,
            "is_verified": profile.get("is_verified", False),
            "follower_count": profile.get("followed_by_count", 0),
            "bio_text": bio_text[:300] + "..." if len(bio_text) > 300 else bio_text,
            "bio_link": bio_link,
            "telegram_links": telegram_links,
            "profile_url": f"https://instagram.com/{username}"
        })

        # Delay to avoid rate limiting
        time.sleep(delay)

    return results

def save_results(results: List[Dict]):
    """Save results in multiple formats."""

    # Filter only accounts with Telegram links
    accounts_with_telegram = [r for r in results if r.get("telegram_links")]

    # 1. Save detailed JSON
    with open("telegram_results_detailed.json", "w", encoding="utf-8") as f:
        json.dump({
            "metadata": {
                "total_accounts": len(results),
                "accounts_with_telegram": len(accounts_with_telegram),
                "private_accounts": len([r for r in results if r.get("is_private")]),
                "verified_accounts": len([r for r in results if r.get("is_verified")]),
                "timestamp": time.strftime("%Y-%m-%d %H:%M:%S")
            },
            "accounts_with_telegram": accounts_with_telegram,
            "all_accounts": results
        }, f, indent=2, ensure_ascii=False)

    # 2. Save simple list of Telegram links
    with open("telegram_links.txt", "w", encoding="utf-8") as f:
        f.write("=" * 70 + "\n")
        f.write("TELEGRAM LINKS FOUND IN INSTAGRAM BIOS\n")
        f.write("=" * 70 + "\n\n")

        for result in accounts_with_telegram:
            f.write(f"Instagram: @{result['username']}\n")
            f.write(f"Name: {result.get('full_name', 'N/A')}\n")
            f.write(f"Followers: {result.get('follower_count', 'N/A'):,}\n")
            f.write(f"Verified: {'Yes' if result.get('is_verified') else 'No'}\n")

            if result.get('bio_link'):
```

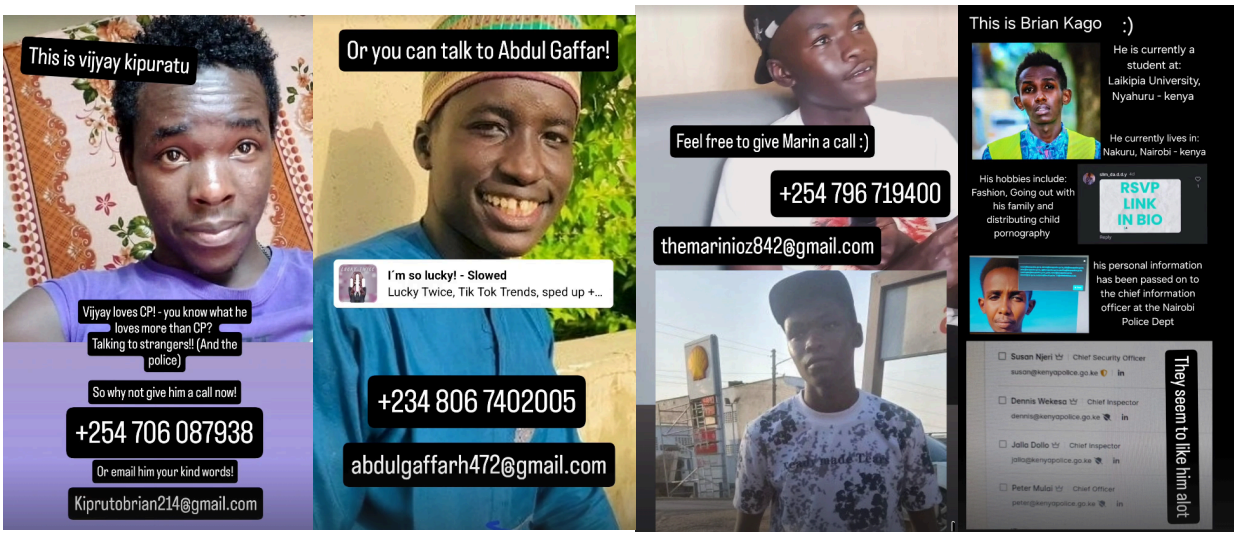
Which once compiled and ran across 3 different scripts, proved highly effective against removal of the illicit material and identification of those behind the accounts:



Managing to take down over 1000+ accounts distributing harmful material:



And identifying some of the biggest distributors of this martial through META platforms and telegram:



Who's full information, including contact details and locations where passed on to both independent organisations local to them and international law enforcement:

Child Porn Buyer/Distributor Report (Ripoti ya Mnunuzi/Msambazaji wa Ponografia ya Watoto)

From Prvvv
Bcc susan@kenyapolice.go.ke, dennis@kenyapolice.go.ke, jalla@kenyapolice.go.ke, peter@kenyapolice.go.ke, aj@kenyapolice.go.ke, geoffrey@kenyapolice.go.ke, washington@kenyapolice.go.ke, peter.odwar@kenyapolice.go.ke, director@cid.go.ke, incidents@ke-cirt.go.ke, 1 more...
Date 2025-11-18 22:46

Summary Headers HTML

Hello, hope you are well.

I am a European security researcher, I have Investigated and tracked down the following person distributing, selling and buying explicit images of children through Instagram, telegram and meta services- he is apart of a large CSAM network, please find the following information useful to track his whereabouts and come back to me with any help or additional information.

Image:

Instagram: https://www.instagram.com/slim_da.d.d.y
Facebook: https://www.facebook.com/profile.php?id=100094177127327

Name: Brian Kago

Location: Nakuru, Nairobi - Kenya

Went to: Anestar Group of Schools - Nakuru
Goes to: Laikipia University, Nyahururu - Kenya (

Thank you very much -

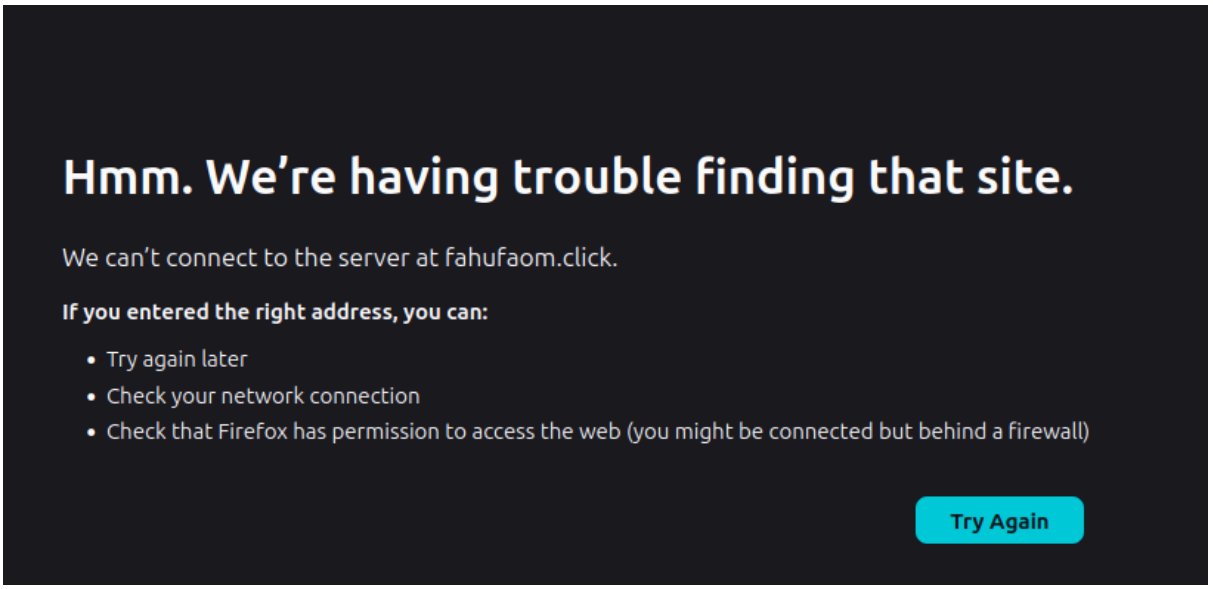
Prv

(

**"Habari, natumai uko salama.

Mimi ni mtafiti wa usalama kutoka Ulaya. Nimefanya uchunguzi na kufuatilia mtu anayesambaza, kuuza na kununua picha za watoto zenye maudhui ya ngono kupitia Instagram, Telegram na huduma nyingine za Meta. Yeye ni sehemu ya mtandao mkubwa wa CSAM. Tafadhali pata taarifa zifuatazo ili ziweze kusaidia katika kumfuatilia, na naomba mniarifu ikiwa kuna msaada mwingine au taarifa za ziada mnazohitaji."**)

Many non telegram, third party websites used as advertising points and vectors for distributing CSAM were also targeted and taken down, found VIA similar comment advertisements and targets BIO's:



Conclusion

This project not only exposed serious systemic failures within META's trust and safety infrastructure, but also demonstrated the significant — and sometimes disruptive — impact that a single independent actor can have when confronting online child exploitation networks. By engaging directly with these accounts, identifying their connections, and deploying automated tools to mass-report and disrupt their distribution channels, I directly contributed to dismantling parts of a large-scale CSAM operation. Over one thousand accounts were removed, and high-risk offenders were escalated to authorities with actionable intelligence.

However, the scale of that intervention also underscores its inherent risks. Operating without formal oversight or institutional backing meant accepting the possibility of damaging evidence chains, triggering offender adaptation, or interfering with active investigations already underway. The decision to intervene came with the knowledge that mistakes could lead to consequences — legal, ethical, or operational — and that heavy responsibility rests on anyone who inserts themselves into an arena typically handled by trained professionals.

Yet doing nothing would have enabled continued harm to real children, whose abuse was being algorithmically promoted for engagement metrics. If anything, the disruption caused by my actions highlights just how much damage can be done — either by bad actors exploiting these systems, or by a lone individual forced into action because those responsible for safeguarding users failed to do so.

Ultimately, this report demonstrates both what can be achieved and what should never have been necessary. The burden should never fall on independent individuals to protect children from exploitation on mainstream platforms. META and similar companies must fundamentally rethink their moderation architecture, because when the safety mechanisms fail, the consequences ripple outward — affecting victims, communities, and even those who step in to try to stop the harm.