Penetrum Security -- The Difference

Penetrum Security Analysis of TikTok versions 10.0.8 - 15.2.3

TikTok is a mobile application in which users can view, upload, like, duet, and share videos with one another. This mobile application seems harmless, with 66% of it's users worldwide being under the age of 30 years old, and within the US 60% being within the age range of 16-24 years old (https://mediakix.com/blog/top-tik-tok-statistics-demographics/). Seems like fun making silly videos and memes, correct?

However, 37.70% of the known IP addresses linked to TikTok are Chinese. On TikTok's ISP's privacy policy, they declare that they harvest and share your data with third-party vendors and business partners (https://rule.alibaba.com/rule/detail/2034.htm#AA). What if I told you that TikTok harvests an excessive amount of data and that this can all be proven right now? In this whitepaper, we here at Penetrum are going to prove that there's an excessive amount of data harvesting, some vulnerabilities in TikTok's code, as well as a few things that may make you feel pretty uncomfortable. Buckle up folks, it's about to get pretty wild. (All research will be publically available for all to see at https://penetrum.com/research/)

Overview:

- 37.70% of known ip addresses linked to TikTok that were found inside of APK source code are linked to Alibaba.com; a Chinese sanctioned ISP located in Hangzhou.
- Alibaba's privacy policy states that they share and distribute personal information of its users
- TikTok in itself is a security risk due to the following reasons;
    - Webview, and remote webview enabled by default
    - Application appears to take commands over text and receives them piping them directly into Java as an OS command
    - The application that uses Java reflection while decreasing VM load time can also be taken advantage of by malicious users and has a CVE score of 8.8
    - This application has been observed to log sensitive information such as;
        - Device information
        - User GEOlocation
        - Monitors user activity

Links to Chinese IP addresses

Now it's not a secret that TikTok is a Chinese mobile application, but just how much of this application is sanctioned and controlled by the Chinese? Well, according to our research, 23/61 IP addresses (or 37.70%) are stationed outside of the US with a majority of them being stationed inside of China and hosted by an ISP named Alibaba. Using the API from IPvoid and source code from the APK we were able to extract around 61 unique IP addresses. From there we used IPVoid's API to discover the information associated with them. Below is just a single example of what we received back:

```
IP: 161.117.70.89
Hostname:
---
Detections Count: 0
Detected By:
---
Country: CN (China)
Continent: AS (Asia)
Region: Zhejiang
City: Hangzhou
Latitude: 30.2936496735
Longitude: 120.161422729
ISP: Alibaba.com Singapore E-Commerce Private Limited
---
Is Proxy: False
Is Web Proxy: False
Is VPN: False
Is Hosting: True
Is Tor: False
```

On the plus side, the IP address isn't detected to be doing any nefarious activities, but that doesn't change the fact that it is still inside of a Chinese controlled ISP. Now what we ask you to do is to keep this information in mind while we continue and come back to this shortly.

Alibaba's Privacy Policy

Having mentioned above about the Chinese connections, we need to get into exactly how Alibaba protects, and uses end user private information. To read fully on how it is processed and how it is handled you can follow this link: https://rule.alibaba.com/rule/detail/2034.htm#AA, which will take you to their disclaimers page. Let us highlight a few things about the disclaimers that stuck out to us personally;

- *you will be asked to provide certain contact information that is necessary for the registering for a Platform account on behalf of a Buyer or Seller, including name, address, phone number, email address, job title and department;*
- *We will collect details of user activities, transactions and interactions on the Platform including information relating to the types and specifications of the products and services purchased, pricing and delivery information, dispute and complaint records, communications between users and any information disclosed in any discussion forum.*
- *We may disclose (or provide access to) personal information to the following categories of recipients:*
    - *Third party business partners, service providers and/or affiliates of Alibaba.com engaged by us or working with us to assist us to provide services to you or who otherwise process personal information for purposes described in this Privacy Policy or notified to you when we collect your personal information. Categories of these partners or service providers include:*
    - *cloud computing service providers to provide cloud storage services;*
    - *third party rating / reviewing service providers to carry out reviews of our services with customers if you choose to participate in reviewing or rating Alibaba products and/or services;*

Now all of this seems pretty straightforward. That is, until you realize that Alibaba was forced to shut down their servers due to a massive data leak in or around July 2019 (https://www.breakingasia.com/news/china-alibaba-shut-down-server-after-data-leak/). A total of 899GB+ of data was exposed to cyber criminals for over half a month. In that time Alibaba was (*we assume*) cooperating with TikTok on a daily basis. This not only puts Chinese citizens in jeopardy, but American citizens as well. Before this investigation, a few of us had been using TikTok for well over a year, and we never heard anything about this 899GB+ data leak from their Chinese hosting ISP… how about you? Something from this leak that we would like to personally highlight is the following paragraph (keep this in mind for later):

If credit evaluation reports from the mobile loan apps weren't bad enough, Anurag's team also discovered 4.6 million unique entries of device data, including GPS locations, full lists of mobile

contacts, SMS logs, IMSI numbers, IMEI numbers, device models and versions, stored app data from previous installations, and memory data (composition and content of mobile phone memory).

TikTok and What Data is Collected

It is known that mobile applications collect data, which they use to generate income and for targeted advertising for the end user. However, when does extracting data hit the threshold of too much? Is it necessary for a mobile application to harvest the IMEI number of a phone, it's screen resolution, or the SIM card provider information? Is it normal for an application to have a section that enables tracking, collects GPS coordinates, and more? In this section we will get into what data is collected by TikTok. (All code seen here is from version 15.2.3).

```
/* access modifiers changed from: package-private */
/* renamed from: `  reason: contains not printable characters */
public final void m72(Context context, Intent intent) {
    String stringExtra = intent.getStringExtra("shouldMonitor");
    if (stringExtra != null) {
        AFLogger.afInfoLog("Turning on monitoring.");
        AppsFlyerProperties.getInstance().set("shouldMonitor", stringExtra.equals("true"));
        m61(context, (String) null, "START_TRACKING", context.getPackageName());
        return;
    }
    AFLogger.afInfoLog("****** onReceive called *******");
    AppsFlyerProperties.getInstance().setOnReceiveCalled();
    String stringExtra2 = intent.getStringExtra("referrer");
    AFLogger.afInfoLog("Play store referrer: ".concat(String.valueOf(stringExtra2)));
    if (stringExtra2 != null) {
        if ("AppsFlyer_Test".equals(intent.getStringExtra("TestIntegrationMode"))) {
            SharedPreferences.Editor edit = com.ss.android.ugc.aweme.keva.d.a(context, "appsflyer-data", 0).edit();
            edit.clear();
            if (Build.VERSION.SDK_INT >= 9) {
                edit.apply();
            } else {
                edit.commit();
            }
            AppsFlyerProperties.getInstance().f80 = false;
            AFLogger.afInfoLog("Test mode started..");
            this.f35 = System.currentTimeMillis();
        }
        SharedPreferences.Editor edit2 = com.ss.android.ugc.aweme.keva.d.a(context, "appsflyer-data", 0).edit();
        edit2.putString("referrer", stringExtra2);
        if (Build.VERSION.SDK_INT >= 9) {
            edit2.apply();
        } else {
            edit2.commit();
        }
```

The above image is pulled from the source code of TikTok (you can see all the code and the ones specified by downloading our research at https://penetrum.com/research/), the above source code contains a tracker named AppsFlyer. AppsFlyer is, according to their website, a "*enterprise CRM-like SaaS platform that allows app developers to store, own, analyze, and control their customers data*" (https://www.appsflyer.com/product/security-and-privacy/). That being said, it is clearly stated in the code that they use it for monitoring. To what extent? We are not sure, due to obfuscation and anti-VM precautions taken by TikTok after version 10.0.8. What we can derive from the code is that when the variable *shouldMonitor* is set to true the application

enables a tracker. One of the tracking methods AppsFlyer uses is location data, which is used to process user location to produce location based advertisements.

```
    static Location m114(Context context) {
        Location location;
        Location location2;
        try {
            LocationManager locationManager = (LocationManager) context.getSystemService("location");
            if (m113(context, new String[]{"android.permission.ACCESS_FINE_LOCATION",
"android.permission.ACCESS_COARSE_LOCATION"})) {
                location = locationManager.getLastKnownLocation("network");
            } else {
                location = null;
            }
            if (m113(context, new String[]{"android.permission.ACCESS_FINE_LOCATION"})) {
                location2 = locationManager.getLastKnownLocation("gps");
            } else {
                location2 = null;
            }
            if (location2 == null && location == null) {
                location2 = null;
            } else {
                if (location2 != null || location == null) {
                    if (location == null) {
                    }
                }
                location2 = location;
            }
            if (location2 != null) {
                return location2;
            }
            return null;
        } catch (Throwable unused) {
            return null;
        }
    }
```

Continuing with the location tracking, the code above is also pulled from the APK source code. The requested permission *android.permission.ACCESS_FINE_LOCATION* is used to allow the API "*to determine as precise a location as possible from the available location providers, including the Global Positioning System (GPS) as well as WiFi and mobile cell data.*" (https://developers.google.com/maps/documentation/android-sdk/location). The protection level of this permission is labeled as "*dangerous*" and another permission (which is also requested by the application) is given as an alternative to the use of this one.

```
public void doLoadLibrary(String str) {
}

public String getAbClient() {
    return "";
}

public String getAbFeature() {
    return "";
}

public String getAbVersion() {
    return "";
}

public String getAppId() {
    return "-1";
}

public String getAppName() {
    return "";
}

public String getBypassBOEJSON() {
    return null;
}

public String getCarrierRegion() {
    return "";
}

public String getChannel() {
    return "";
}

public String getDeviceId() {
    return "";
}

public String getDevicePlatform() {
    return "android";
```

Continuing on from here we come across this code, which is a file filled with data harvesting. It collects everything from the current OS version to running network events (WiFi SSID changes, etc), and even the IMEI number of the associated phone. This is extremely alarming to us due to what was said in the above data leak "*including GPS locations, full lists of mobile contacts, SMS logs, IMSI numbers, IMEI numbers, device models and versions, stored app data from previous installations, and memory data"*, now we at Penetrum cannot make a direct connection to this breach and TikTok, but it seems very suspicious that the data collected is mentioned in a breach that happened to their ISP provider. We understand that data is needed to be collected for the developers to thrive and continue producing good code for the application. But, where do we draw the line when it comes to too much data?

```
try {
    Object obj2 = f88446a;
    if (obj2 == null) {
        k.a("obMSimTelephonyManager");
    }
    Class<?> cls = Class.forName(obj2.getClass().getName());
    Class[] clsArr = {Integer.TYPE};
    Method method = cls.getMethod("getDeviceId", (Class[]) Arrays.copyOf(clsArr, 1));
    Object[] objArr = new Object[1];
    if (Build.VERSION.SDK_INT == 21) {
        objArr[0] = Long.valueOf(a(i));
    } else {
        objArr[0] = Integer.valueOf(b(i));
    }
    Object obj3 = f88446a;
    if (obj3 == null) {
        k.a("obMSimTelephonyManager");
    }
    Object invoke = method.invoke(obj3, Arrays.copyOf(objArr, 1));
    if (invoke != null) {
        str = invoke.toString();
    }
    if (str != null) {
        String lowerCase = str.toLowerCase();
        k.a((Object) lowerCase, "(this as java.lang.String).toLowerCase()");
        if (k.a((Object) lowerCase, (Object) TEVideoRecorder.FACE_BEAUTY_NULL)) {
        }
        StringBuilder sb = new StringBuilder("imei is :");
        if (str == null) {
            k.a();
        }
        sb.append(str);
        return str;
    }
    Method method2 = cls.getMethod("getImei", (Class[]) Arrays.copyOf(clsArr, 1));
    Object obj4 = f88446a;
    if (obj4 == null) {
        k.a("obMSimTelephonyManager");
    }
    Object invoke2 = method2.invoke(obj4, Arrays.copyOf(objArr, 1));
    if (invoke2 == null) {
```

The above code taken from the TikTok APK, shows the collection of cellphone data, specifically the IMEI of the cell phone. The IMEI number of a phone is literally created to identify the phone. The IMEI when used by trackers is usually used to determine whether an application is re-installed on a phone and give an analysis of other applications that are installed on the phone. Essentially, it creates an extremely realistic and graphic fingerprint of your phone which can be used to determine everything you have installed. Getting information like this has been considered "*controversial*" in multiple circumstances. Alongside the IMEI is the IMSI which is used to follow users while getting a new phone, basically while transferring your SIM data to a new phone, the IMSI number stays with you. Tracking this number gives the tracker a solid understanding of your habits when it comes to phones and can give you more targeted advertisements, and even create a profile on you for your next potential phone. We were not able

to say with 100% certainty that the IMSI number is also extracted, but it seems relevant to the cause of the tracker to also extract the IMSI number with the IMEI.

Now we are not saying that TikTok is using these things for nefarious purposes in any way, we at Penetrum believe that everyone should have the right to know what data is being harvested by companies and would like to give our readers a clearer understanding of what happens when you download the mobile application TikTok. From our understanding and our analysis it seems that TikTok does an excessive amount of tracking on it's users, and that the data collected is partially if not fully stored on Chinese servers with the ISP Alibaba. It seems slightly coincidental to us that Alibaba's data breach specifically states that the breach itself included that IMEI, IMSI, phone numbers, and user data specifically pertaining to phones was breached as well as other personal information.

Security Concerns of Downloading TikTok

If the above information wasn't enough to get you thinking about what TikTok is doing with your information, or that the security needs to be seriously looked into pertaining to TikTok. Maybe us talking directly about the insecurities that arose during our analysis will get you thinking. We will discuss the following insecurities:

- Execution of OS commands
- Insecure cryptography usage
- Potential SQL injection code from user defined variables
- Storing of API tokens
- Webview enabled by default along with insecure webview enabled

Lets dive in by first examining the OS execution done by TikTok:

```
package com.bytedance.j;

import d.e.c;
import d.e.o;
import d.f.b.k;
import java.io.BufferedReader;
import java.io.Closeable;
import java.io.InputStreamReader;
import java.io.Reader;
import java.util.List;

public final class b {

    /* renamed from: a  reason: collision with root package name */
    public static final b f24548a = new b();

    private b() {
    }

    /* JADX WARNING: Code restructure failed: missing block: B:10:0x0039, code lost:
        throw r1;
     */
    /* JADX WARNING: Code restructure failed: missing block: B:5:0x0032, code lost:
        r1 = move-exception;
     */
    /* JADX WARNING: Code restructure failed: missing block: B:9:0x0036, code lost:
        d.e.c.a(r0, r3);
     */
    public static List<String> a(String str) {
        k.b(str, "cmd");
        Process exec = Runtime.getRuntime().exec(str);
        k.a((Object) exec, "process");
        Closeable bufferedReader = new BufferedReader(new InputStreamReader(exec.getInputStream()));
        List<String> a2 = o.a((Reader) (BufferedReader) bufferedReader);
        c.a(bufferedReader, (Throwable) null);
        return a2;
    }
}
```

The above code (and all code shown) was also taken from TikTok version 15.2.3. This code is used to execute OS commands from inside of TikTok and is used a few times within the application to perform operating system commands. We are not entirely sure if the code is user defined or not because our analysis was prevented from viewing activities performed on any application version higher than 10.0.8. Now executing OS commands is normal and understandable, but executing them from user input is less acceptable. More research will need to be done in order to make a concrete determination if TikTok executes from user input, but we are confident that there is some issue on what is executed as an OS command.

Not only does the application execute OS commands, but it also uses insecure cryptographic algorithms, such as MD5:

```
/* renamed from: ˏ  reason: contains not printable characters */
public static String m168(String str) {
    try {
        MessageDigest instance = MessageDigest.getInstance("MD5");
        instance.reset();
        instance.update(str.getBytes("UTF-8"));
        return m167(instance.digest());
    } catch (Exception e2) {
        StringBuilder sb = new StringBuilder("Error turning ");
        sb.append(str.substring(0, 6));
        sb.append(".. to MD5");
        AFLogger.afErrorLog(sb.toString(), e2);
        return null;
    }
}
```

The hashing algorithm MD5 has been deprecated since at least 2011 along with SHA1. MD5 has been known to be a very weak hashing algorithm due to the speed and the ability to make thousands of hashes very quickly in a bruteforce attempt.

Not only do they use insecure hashing algorithms, but there is potential for a user defined SQL query as well:

```
    public final void a(SQLiteDatabase sQLiteDatabase, a aVar) {
        if (aVar != null && !a(sQLiteDatabase, aVar.f21219a)) {
            super.a(sQLiteDatabase, aVar);
            try {
                sQLiteDatabase.execSQL("delete from " + this.f21222b + " where _id" + " in (select _id" + " from " + this.f21222b
+ " order by insert_time" + " desc limit 1000 offset 500" + ")");
            } catch (Exception unused) {
            }
        }
    }
```

Everyone knows that you never give a user access to your raw SQL commands, this can be a very bad thing, very quickly. The above code does exactly that. The arguments passed into the code are user defined (we are not 100% sure where they are defined but think it has something to do with the profile editor), once executed they do not seem to go through any type of sanitization and are executed to delete a column from a table.

TikTok seems to have a knack for hardcoding their information into the applications so they will not have to retrieve it from elsewhere, for example:

```
    private int b(c cVar) {
        g gVar;
        try {
            gVar = g.b((CharSequence) this.f25317c + "/issue/" + cVar.l + "/attachments").a("X-Atlassian-Token", "no-
check").b(this.f25318e, this.f25319f);
            try {
                List<String> list = cVar.m;
```

The above code is an example of TikTok hard coding API tokens into its code. Hardcoding tokens is not that big of deal, until you realize that they're hardcoded into plaintext and are accessible throughout the entire application, that's when it starts becoming a bigger deal than it should be.

Alongside all these issues we discovered that TikTok uses webview and reflection alot, when done correctly these do not bring up any real security threats, however:

j.java

```java
package com.bytedance.android.monitor.webview;

import android.graphics.Bitmap;
import android.os.Build;
import android.text.TextUtils;
import android.view.View;
import android.webkit.ValueCallback;
import android.webkit.WebView;
import com.bytedance.android.monitor.webview.c;
import java.util.HashMap;
import java.util.Map;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

public final class j implements c, e {

    /* renamed from: a  reason: collision with root package name */
    private static c f20425a = null;

    /* renamed from: b  reason: collision with root package name */
    private static e f20426b = null;

    /* renamed from: c  reason: collision with root package name */
    private static String f20427c = "ttlive_web_view_tag";

    /* renamed from: d  reason: collision with root package name */
    private static String f20428d = "ttlive_web_view_last_url_tag";
```

```java
package com.ss.android.ugc.aweme.crossplatform.business;

import android.content.Context;
import android.os.Bundle;
import android.text.TextUtils;
import android.webkit.JavascriptInterface;
import android.webkit.WebView;
import com.google.b.c.aw;
import com.ss.android.deviceregister.d;
import com.ss.android.ttve.nativePort.TEVideoRecorder;
import com.ss.android.ugc.aweme.IAccountUserService;
import com.ss.android.ugc.aweme.ac.a.j;
import com.ss.android.ugc.aweme.ac.a.l;
import com.ss.android.ugc.aweme.bi.b;
import com.ss.android.ugc.aweme.common.i;
import com.ss.android.ugc.aweme.crossplatform.activity.q;
import com.ss.android.ugc.aweme.crossplatform.b.a;
import com.ss.android.ugc.aweme.crossplatform.base.c;
import com.ss.android.ugc.aweme.crossplatform.business.BusinessService;
import com.ss.android.ugc.aweme.crossplatform.platform.webview.SingleWebView;
import com.ss.android.ugc.aweme.lancet.f;
import com.ss.android.ugc.aweme.setting.model.WebShareMode;
import com.ss.android.ugc.aweme.share.aj;
import com.ss.android.ugc.aweme.share.improve.pkg.WebSharePackage;
import com.ss.android.ugc.aweme.sharer.ui.SharePackage;
import com.zhiliaoapp.musically.df_photomovie.R;
import d.f.b.k;
import d.m.p;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.json.JSONException;
import org.json.JSONObject;

abstract class AbsShareBusiness extends BusinessService.Business {
```

```
package com.bytedance.android.livesdk.browser.d;

import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.Color;
import android.net.Uri;
import android.net.http.SslError;
import android.os.AsyncTask;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentActivity;
import android.support.v7.widget.am;
import android.text.TextUtils;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.webkit.CookieManager;
import android.webkit.GeolocationPermissions;
import android.webkit.SslErrorHandler;
import android.webkit.WebChromeClient;
import android.webkit.WebResourceResponse;
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.FrameLayout;
import android.widget.ImageView;
import android.widget.TextView;
import c.a.ab;
import c.a.ae;
import c.a.e.e.b.af;
import c.a.e.e.e.r;
import c.a.t;
import c.a.x;
```

The above code is just an example of how many times webview and reflection is used within TikTok. Reflection can in theory make the latency and load on the Java VM quicker, but the use of webview also has a very serious security risk associated with it that is ranked as an 8.8 CVE score. During the course of analyzing this application, there were several concerning areas relating to webview and its insecure use of SSL/TLS like ignoring SSL/TLS errors all together, reflection, or *REMOTE_DEBUGGING* as captured here:

```
    private synchronized void m173(String str, PackageManager packageManager) {
        AppsFlyerProperties instance = AppsFlyerProperties.getInstance();
        AppsFlyerLib instance2 = AppsFlyerLib.getInstance();
        String string = instance.getString("remote_debug_static_data");
        if (string != null) {
            this.f189 = new JSONObject(string);
        } else {
            this.f189 = new JSONObject();
            m174(Build.BRAND, Build.MODEL, Build.VERSION.RELEASE, instance.getString("advertiserId"),
instance2.f33, instance2.f42);
            m171("4.8.20.170", instance.getString("AppsFlyerKey"), instance.getString("KSAppsFlyerId"),
instance.getString("uid"));
            try {
                int i = packageManager.getPackageInfo(str, 0).versionCode;
                m177(str, String.valueOf(i), instance.getString("channel"),
instance.getString("preInstallName"));
            } catch (Throwable unused) {
            }
            instance.set("remote_debug_static_data", this.f189.toString());
        }
        try {
            this.f189.put("launch_counter", this.f198);
        } catch (JSONException e2) {
            a.b(e2);
        }
    }
```

As well as here:

```
public final WebResourceResponse shouldInterceptRequest(WebView webView, String str) {
        com.bytedance.android.livesdk.browser.e.a.b();
        if (!TextUtils.isEmpty(str)) {
            WebResourceResponse a2 = com.bytedance.android.livesdk.ab.j.j().d().d().a(str, webView);
            if (a2 != null) {
                if (TextUtils.equals("text/html", a2.getMimeType())) {
                    c.this.C = System.currentTimeMillis();
                    c.this.D.put("intercept_delay", Long.valueOf((c.this.C - c.this.B) / 1000));
                }
                if (c.c(str)) {
                    com.bytedance.android.livesdk.browser.e.a.a(c.this.f9358a, str, 0);
                }
                j.b().a(webView, str, true);
                if (c.b(str)) {
                    c.this.a(webView, str);
                }
                return a2;
            }
            if (c.c(str)) {
                com.bytedance.android.livesdk.browser.e.a.a(c.this.f9358a, str, 1);
            }
            j.b().a(webView, str, false);
        }
        return super.shouldInterceptRequest(webView, str);
    }
```

Allowing user defined commands to be executed within webview has the potential to lead to arbitrary files being loaded on the device that is hosting the application. Which in theory can lead to malware being loaded from inside the application, chained with remote debugging to see what fails in your malware. It also allows a very big window for attackers to not only upload, but execute, and debug their malware as well(in almost real time). To us here at Penetrum this seems like an extreme security risk that shouldn't be taken lightly. It is also good to note that webview is implemented in such a way, within the application, that it ignores SSL errors which means the authenticity of the sender/client cannot be established…. Meaning anyone who figures out how to enable and leverage this capability of this app, can use it or execute man in the middle attacks. This can be seen within our static reports at https://penetrum.com/research/.

Conclusion

At Penetrum, we strive to provide the most detailed, transparent, and accurate security analysis and audits that are within our ability. We also strive to develop the most ambitious, yet practical cybersecurity tools and use them in the field. After extensive research, we have found that not only is TikTok a massive security flaw waiting to happen, but the ties that they have to Chinese parties and Chinese ISP's make it a very vulnerable source of data that still has more to be investigated. Data harvesting, tracking, fingerprinting, and user information occurs throughout the entire application. As a US company, we feel that it is our responsibility to raise awareness of this extensive data harvesting to TikTok's 1 billion users.

For more in depth information on this issue, check out our research at https://penetrum.com/research/.

We appreciate you taking the time to look through our research and appreciate you taking the time out of your schedule to read our whitepaper. We'll see you then.

Penetrum - The Difference