A Lazarus Keylogger- PSLogger

norfolkinfosec.com/a-lazarus-keylogger-pslogger/

norfolk January 22, 2019

This blog recently referenced a late July VNCert report containing file-based IOCs affiliated with attempted intrusions against financial organizations in Vietnam. Several contextual and technical characteristics of these files tie them to recent activity typically attributed to North Korean adversaries with a specific interest in the financial sector.

This post explores the technical characteristics of one of these files, a keylogging and screengrabbing utility. Two versions of this utility have appeared in-the-wild. The first is directly identified in the VNCert alert and is a DLL injected via a modified version of the open-source PowerSploit framework. The second is a standalone executable submitted to VirusTotal by a user in Pakistan (and possibly used in an intrusion in that region).
syschk.ps1 (Vietnam)

MD5: 26466867557f84dd4784845280da1f27
SHA1: ed7fcb9023d63cd9367a3a455ec94337bb48628a
SHA256: 791205487bae0ac814440573e992ba2ed259dca45c4e51874325a8a673fa5ef6

Syschk.ps1 contains three primary components: (1) A Base64 encoded DLL, (2) a Base64 encoded variant of PowerSploit's Invoke-ReflectivePEInjection, and (3) a routine for decoding and executing these components. This script also contains references to "c:\windows\temp\TMP0389A.tmp" (noted in the previous post for its similarity to another DPRK file path and directory) and "c:\programdata\1.dat" as part of a "remove-item" cmdlet routine. The Base64 DLL can be copied, converted, and saved to another file for analysis.

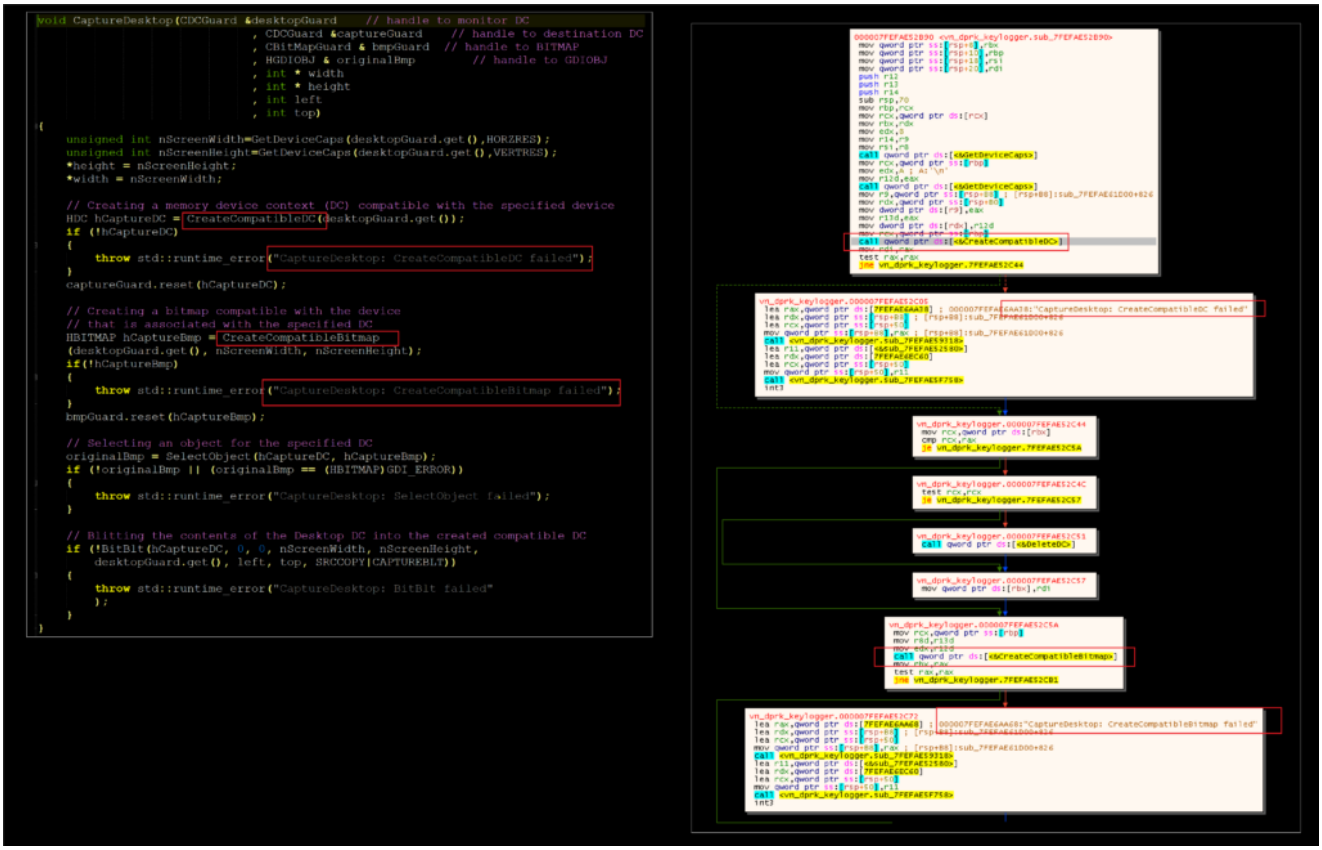
Extracted DLL
MD5: d45931632ed9e11476325189ccb6b530
SHA1: 081d5bd155916f8a7236c1ea2148513c0c2c9a33
SHA256: efd470cfa90b918e5d558e5c8c3821343af06eedfd484dfeb20c4605f9bdc30e

The extracted malware is designed for 64-bit operating systems and contains an export named "Process." The malware has two primary functions: grabbing keystroke (and clipboard) data, and grabbing screen captures of the user's desktop. At launch, the malware creates a file at "c:\windows\temp\TMP0389A.tmp" containing the directory that the malware will save files in.

Next, the file begins monitoring keystrokes. These are logged and saved to a hardcoded path (visible in plaintext) within the user's "AppData\Local\Temp" directory under a folder named "GoogleChrome" in a file named "chromeupdater_pk." The keylogging routine uses

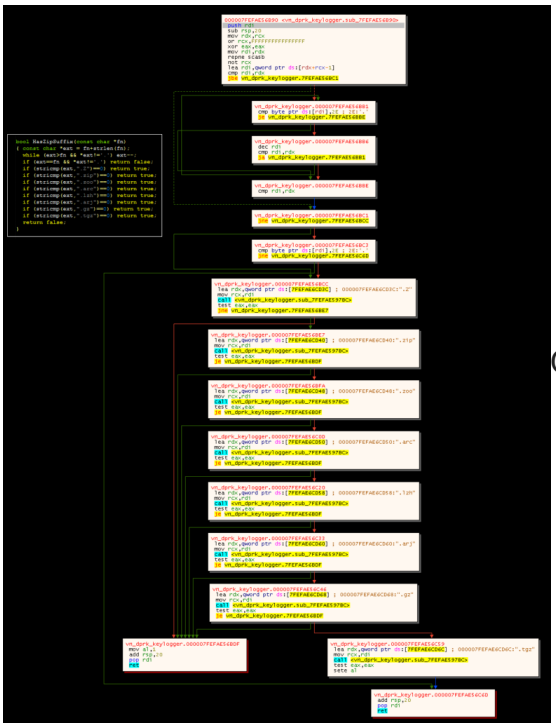
the GetKeyState and GetAsyncKeyState APIs and is not sophisticated, and logged keystroke and clipboard context is saved in plaintext.

The malware's other functionality is to capture the desktop, compressing the images and saving them in the same directory. These files are saved with the filename format chromeupdater_ps_[timestamp]. Notably, the malware uses two open-source implementations to achieve this. To capture the desktop, it uses code likely derived from this example (or code from which that example was derived). To perform compression, the malware uses the XZip library, a derivation of the Info-Zip project. The combination of these characteristics is useful for identifying an additional variant of this malware.



```
void CaptureDesktop(CDCGuard &desktopGuard    // handle to monitor DC
                    , CDCGuard &captureGuard    // handle to destination DC
                    , CBitMapGuard & bmpGuard  // handle to BITMAP
                    , HGDIOBJ & originalBmp        // handle to GDIOBJ
                    , int * width
                    , int * height
                    , int left
                    , int top)
{
    unsigned int nScreenWidth=GetDeviceCaps(desktopGuard.get(),HORZRES);
    unsigned int nScreenHeight=GetDeviceCaps(desktopGuard.get(),VERTRES);
    *height = nScreenHeight;
    *width = nScreenWidth;

    // Creating a memory device context (DC) compatible with the specified device
    HDC hCaptureDC = CreateCompatibleDC(desktopGuard.get());
    if (!hCaptureDC)
    {
        throw std::runtime_error("CaptureDesktop: CreateCompatibleDC failed");
    }
    captureGuard.reset(hCaptureDC);

    // Creating a bitmap compatible with the device
    // that is associated with the specified DC
    HBITMAP hCaptureBmp = CreateCompatibleBitmap
    (desktopGuard.get(), nScreenWidth, nScreenHeight);
    if(!hCaptureBmp)
    {
        throw std::runtime_error("CaptureDesktop: CreateCompatibleBitmap failed");
    }
    bmpGuard.reset(hCaptureBmp);

    // Selecting an object for the specified DC
    originalBmp = SelectObject(hCaptureDC, hCaptureBmp);
    if (!originalBmp || (originalBmp == (HBITMAP)GDI_ERROR))
    {
        throw std::runtime_error("CaptureDesktop: SelectObject failed");
    }

    // Blitting the contents of the Desktop DC into the created compatible DC
    if (!BitBlt(hCaptureDC, 0, 0, nScreenWidth, nScreenHeight,
        desktopGuard.get(), left, top, SRCCOPY|CAPTUREBLT))
    {
        throw std::runtime_error("CaptureDesktop: BitBlt failed"
        );
    }
}
```

Open-source screengrabbing implementation (left) and disassembled code graph (right).



Open-source XZip code implementation.

HSMBalance.exe

MD5 34404a3fb9804977c6ab86cb991fb130
SHA1 b345e6fae155bfaf79c67b38cf488bb17d5be56d
SHA256 c6930e298bba86c01d0fe2c8262c46b4fce97c6c5037a193904cfc634246fbec

The open-source screengrabbing code used in the keylogger from the Vietnam incident is relatively uncommon: while a basic VirusTotal pivot on one of the more distinct strings from this code identifies dozens of additional files that use it, most belong to a benign screen-sharing package. On the other hand, the malicious files include the Vietnam keylogger and a second keylogger submitted by a user in Pakistan with strikingly similar static and dynamic properties (the hash of this file is listed above). A brief static analysis identifies the following strings:

CDisplayHandlesPool: GetDC failed
CDisplayHandlesPool: EnumDisplayMonitors failed
CreateBitmapFinal: GetDIBits failed
CaptureDesktop: CreateCompatibleDC failed
CaptureDesktop: CreateCompatibleBitmap failed
CaptureDesktop: SelectObject failed
CaptureDesktop: BitBlt failed
SpliceImages: CreateCompatibleDC failed
SpliceImages: CreateCompatibleBitmap failed
SpliceImages: SelectObject failed
SpliceImages: BitBlt failed
wild scan
more < 2

.zip
.zoo
.arc
.lzh
.arj
.tgz

As a triaging step, this strongly suggests the use of the same compression and screengrabbing libraries. In addition, there are several other string similarities:

Pakistan file:
%s%s
%s\tmp_%s
[%02d%02d-%02d:%02d:%02d]
[Num %d]
[ENTER]
[EX]
keycode = %ls keystatus = %d \n
%s\tmp_%s_%02d%02d_%02d%02d%02d
PSLogger.exe

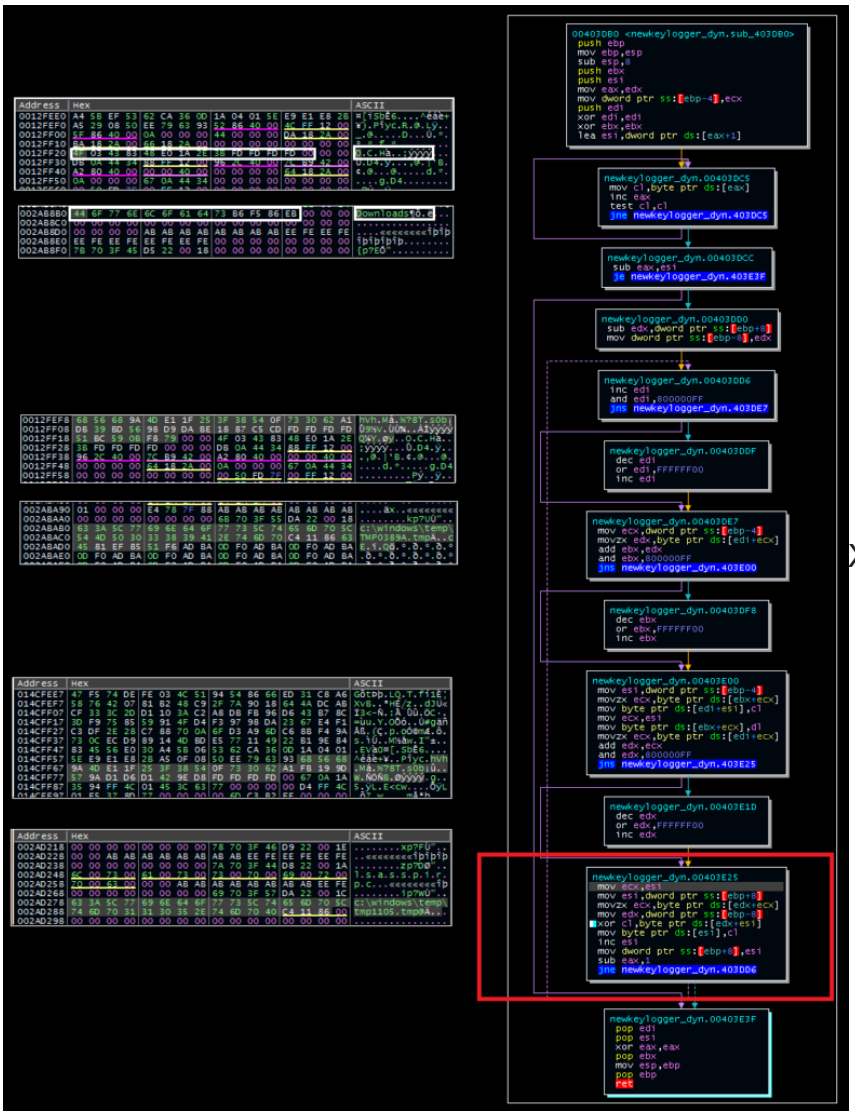
Vietnam File:
%s\chromeupdater_pk
%s\chromeupdater_ps_%04d%02d%02d_%02d%02d%02d_%03d_%d
[%02d:%02d:%02d:%03d]
%s%s
[ENTER]
[EX]
[CTL]
PSLogger.dll

While some of these are not a 1:1 match, there are some clear similarities regarding the likely functionality of the file and the naming conventions. Notably, this “new” file also contains the same exported function name (“Process”) as the Vietnam DLL despite being an executable, suggesting that it may have been built using the same codebase. In addition, the file contains a reference to the same “TMP0389A.tmp” file and path as the Vietnam keylogger (though this is decoded at runtime). The combination of the shared strings, export, libraries (including XZip), and (as will be explored shortly) functionality strongly suggests that this file is likely attributable to the same threat actor.

Functionality

As mentioned, this file contains a decoding routine responsible for decrypting several strings, including:

“Downloads” – Appended to the user’s Appdata\Local\Temp path.
“c:\windows\temp\TMP0389A.tmp” – Intended to contain directory storing keylogs and screenshots.
“c:\windows\temp\tmp1105.tmp” – Unknown purpose


XOR decoding routines for several file paths within the malware

As with the Vietnam file, this file’s two core functions are screengrabbing (using the same library) and keylogging. Both files are stored at “Appdata\Local\Temp\Downloads.” Screenshots are generated as Bitmaps via CreateCompatibleBitmap, compressed, and saved in this directory as “tmp_[username][mmdd{time}]” (e.g. tmp_userA_0121_142748″). As additional screenshots are created, they are appended to the same file (although re-running the malware will create a new file). Kesystrokes (along with process data) are recorded in the same directory, under a file named “tmp_[user]” – unlike the Vietnam file, these keylogs are encrypted prior to storage.

While the malware author did take anti-analysis steps (including encoding several filepaths and keystroke logs), the malware as a whole remains generally unsophisticated.

Closing Thoughts

Neither variant of the malware is particularly sophisticated; in fact, key components of each rely on clunky implementations of open source tools and code (including screengrabbing, compression, and memory injection). This deficiency is most evident in the screenshot compression segment, in which new data is simply appended to an older file. A tool such as 7Zip cannot properly unpack every screenshot appended this way; instead, the adversary would need to manually carve these out (or write an additional tool to do so) given the fact that additional zip data is simply appended to the end of the file.

It is also worth noting that neither file contains a C2 mechanism, meaning that log files and compressed images would have to be extracted from the target device manually. This suggests that these tools are designed for post-compromise use, possibly on machines intended to be monitored for an extended period of time.