“Is It Really You Who Are Taking the Screenshot?”

/// A System Level Solution to Detect Unauthorized Screenshot on Your Laptop 💻 \\\

Junzhou Fang

Supervised by Prof. Qian

INTRODUCTION

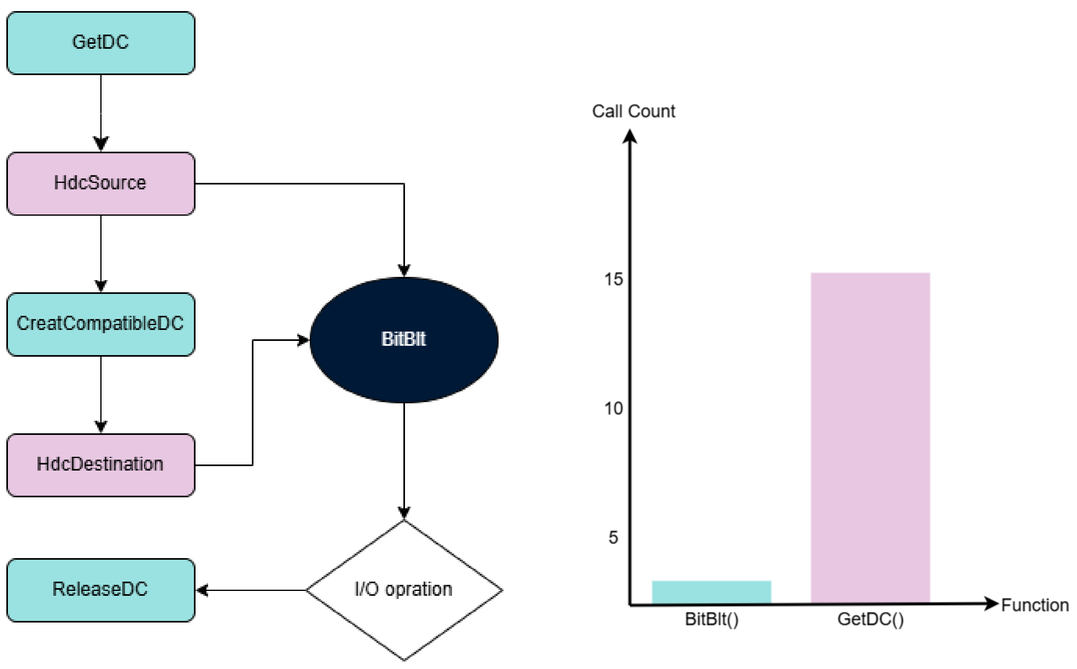
Screenshot malwares may take screenshot to steal various information, from contact information to password. These malwares are hard to detect due to their furtive behavior and tiny size [1]. So far, several works using code analysis or ML to against screenshot malware have been carried out. However, they faced the issues like occupying too much resources and mistakenly attacking legitimate screenshots.

Based on these information, we believe that **the key of against screenshot malware is detection.** One can delete a malware easily after detection. Therefore, in this work we want to design a **light-weight, system level program** that can detect and record all screenshots taken. Then user may refer to the record log to check if there is any application taking unauthorized screenshots. To achieve this, we first find a symbolic event for taking screenshot. Then we use function hooking to monitor the event, which should be equivalent to taking screenshots. The program should be ran with nearly no extra resources needed. Currently our method has gain a preliminary result, and we are working on a comprehensive evaluation. Our work will be open-source at https://github.com/FFFFFFFANG1/Check_it_Twice.

METHOD

Screenshot Design in Windows OS

Our design is based on Windows 11 operating system. Windows11 does not have a unique screenshot function, instead, screenshot is taken by a sequence of functions [1]. Yet, we can find the workflow of screenshot method according to the windows API documentation[2], published by Microsoft. And we summarize the screenshot workflow as follows:



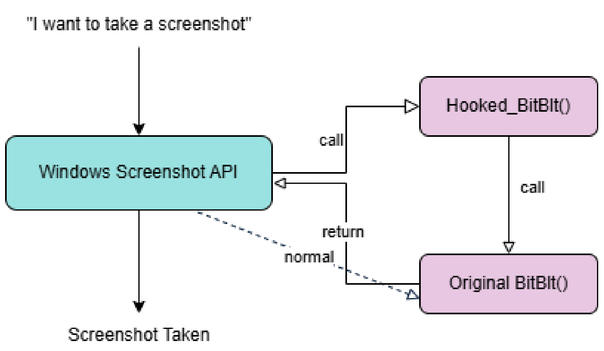
Function Hooking

As one can see, GetDC() and BitBlt() should be two inevitable function called in screenshot method. However, both of them are also called in other methods. We first hook both of them and count how many time they are called beyond taking screenshot. After 2 hours of regular use of laptop at night (playing games, watching videos, chatting), we find GetDC() is called more often, and thus BitBlt() is more suitable for hooking. In the hook function, we record the process ID that asks for the screenshot, and also the image that it takes. The procedure of hook function is concluded in the following pseudo-code and flowchart. We use detours [5] to implement the hook.

Algorithm 1 pseudo-code of Windows screenshot API

```
procedure TAKE_SCREENSHOT
    //find the display
    handle = get_device_context() buffer of the screen
    ...Get all the needed information, like screen size, etc.
    //store the screenshot to memory
    BitBlt(screen_bitmap, memory)   Substitute to Hooked_BitBlt()
    ...Cleanup
end procedure
```

```
procedure HOOKED_BITBLT()
    ...Record the PID of current process
    ...Use the parameters passed in to create a copy of the screenshot and save it
    //call original BitBlt()
    return BitBlt(screen_bitmap, memory)
end procedure
```



METHOD (Cont.)

Injector

In order to force all process to follow our design, the hook is compiled to a dynamic load library (DLL). Then, we write an injector program, which can insert our DLL to either some specific processes or global environment. The injector is compiled as an executable, serving as the interface of our program.

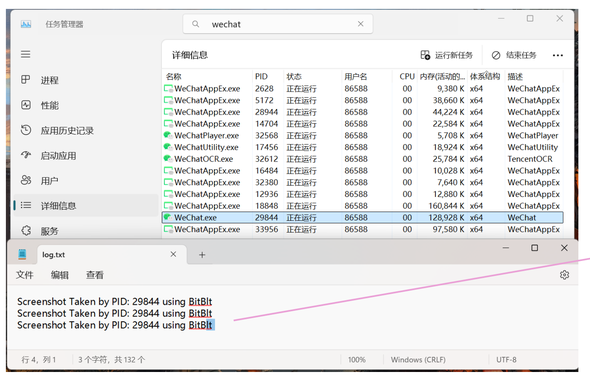
Algorithm 1 pseudo-code of injector

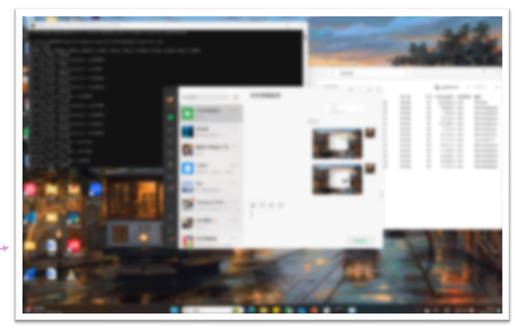
```
procedure INJECT HOOK DDL TO OTHER PROCESS
    Open target process
    Allocate virtual memory for DDL
    Write DDL to allocated memory
    Create a new thread to load DDL to the process
    ...Cleanup
end procedure
```

RESULTS

Functionality

We first use legal, authorized screenshot to prove the functionality of our hooking. We use WeChat as example here. Our program can record WeChat PID after one presses alt+a for screenshot.

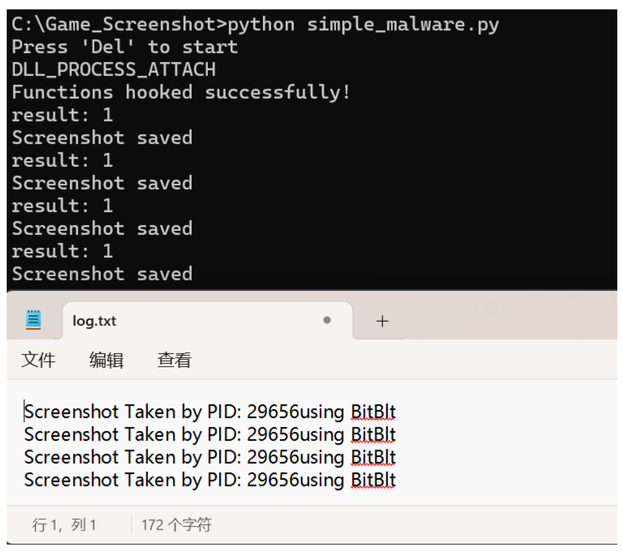




*Sample screenshot, blurred for privacy

Upon Screenshot Malware

MITRE [3] is a cybersecurity website which contains a list of over 60 screenshot malware behaviors. We start with some simple, open-source applications to evaluate the performance of our solution. Mr. Peter [4] is a simple payload to take the screenshot of victims desktop and upload it to C&C. The malware simulates pressing “printscreen” key to take screenshot when it detects the victim is using some targeted applications, then loads the screenshot to customized server. To allow the malware only runs in local environment and facilitate testing, we modify the malware to take 4 screenshots randomly in every hour, and just send it to localhost address. The result shows that our program successfully identify all of them.



*Sample screenshots, blurred for privacy

CONCLUSION & FUTURE WORK

We have proposed a simple and efficient method for screenshot detection. To the best of our knowledge, this is the first system level solution in the field. In the future, we plan to:

- Utilize the dataset [3] to fully evaluate our model, compare it to existed methods.
- Possible improvement on hooking. As mentioned in METHODS, BitBlt() is also not a unique function for screenshot, so we want to further find a symbolic functions combo.
- Extend the hooking to various cases, like Android

REFERENCES

1. H. Sbaï, *et, al*, "Dataset Construction and Analysis of Screenshot Malware," 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Guangzhou, China, 2020
2. https://learn.microsoft.com/zh-cn/windows/apps/
3. Mitre, Screen Capture, https://attack.mitre.org/techniques/T1113/ (Retrieved 01/08/2024)
4. Mr.Peter, https://github.com/mrfr05t/Mr.Peter (Retrieved 28/07/2024)
5. Detours, https://github.com/microsoft/Detours (Retrieved 01/07/2024)