ScreenGuard : Hiding Sensitive information for Screen Sharing Applications

By
Mohammed Shuhad
301462898

Project report Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Engineering

In the School of Engineering Science
Faculty of Applied Science

SIMON FRASER UNIVERSITY
Summer 2023

Approval

Name:	**Mohammed Shuhad**
Degree:	**Master of Engineering**
Title:	**ScreenGuard : Hiding Sensitive information for Screen Sharing Applications**

Supervisory Committee:

Dr. W. Craig Scratchley, P.Eng.
Project Supervisor
Professor
Simon Fraser University

Dr. Ivan V. Bajić
Additional Examiner
Professor
Simon Fraser University

Date Defended/Approved: August 17, 2023

Abstract

In today's age, the need to protect sensitive information has become increasingly crucial. With the widespread adoption of remote work, Screen sharing has become a daily routine for many individuals, facilitated by collaboration tools such as Zoom and Teams. However, this trend has also introduced the risk of accidentally giving away sensitive information, even if it appears only briefly in a single frame. Even when sharing a screen with trusted recipients, there is still a potential risk of the recipient's system being compromised. It may house dormant malicious software capable of recording and analyzing shared information. In this project, I am addressing the need for a screen sharing application that incorporates mechanisms to hide sensitive information in these collaboration tools.

In order to achieve this objective, I propose an image processing approach that entails monitoring the entire screen of the operating system for the presence of sensitive information. When sensitive information is detected, it will be automatically blacked out, and the modified images will be rendered in real-time. The contents of this application can be used for the secure sharing of screen contents through collaboration tools.

Table of Contents

List of Figures

List of Tables

List of Acronym

OS	Operating systems
GUI	Graphical user interface
LSTM	Long short-term memory networks
API	Application program interface
XCB	X protocol C-language Binding
DWM	Desktop Window Manager
OCR	Optical Character Recognition

1 Introduction

In screen sharing, a number of sensitive information can potentially be disclosed. This includes a spectrum of personal data, including full names, addresses, financial details such as credit card numbers and bank account details, authentication credentials like passwords and access tokens, health-related information, legal documents, personal communications, proprietary business data, educational records, and even location details. This sensitive information can come up in a screen sharing session in mainly two forms. One category is textual data, such as emails or content in editors handling documents like DOCs, PDFs, PowerPoint presentations, and spreadsheets. Another category involves sensitive data embedded in images such as photos of driver's licenses or other personal documents. The primary aim of this software is to mask any sensitive information that users choose to block during screen sharing.

Since the main point of concern is textual information, if the text data can be inferred before it is rendered as an image, processing overhead can be avoided. But a major limitation of this approach is that not all textual information cannot be accessed before it is rendered, which is explored in later sections. Due to the fact that sensitive information can come embedded in an image, it is important to deploy image processing techniques. Unlike the text information access, which is restricted to specific applications and dependent on the operating system, this image processing approach will provide an end-to-end solution.

Text recognition is accomplished using tesseract, an open-source Optical Character Recognition (OCR) engine. This engine is built on LSTM neural networks and comes pre-trained for over 100 languages and 35 scripts. However, for this project, only English language data is used for recognition. The accuracy of the model has been improved by applying dynamic programming alignment algorithms to identify text even when there is a reasonable amount of difference.

The software's performance bottleneck is tied to tesseract's recognition process. To overcome this, a customized algorithm has been integrated. This algorithm tracks recognized text between frames, effectively bypassing the bottleneck and resulting in smoother software performance across frames.

2 Graphical User Interface

The idea of graphical user interfaces in computers is a fundamental requirement and it has effortlessly woven itself into our lives. For many of us, the GUI defines our interaction with an Operating system, and we often associate the OS with its GUI. Different OS exhibit slightly varying GUIs due to their different implementations. If a user is familiar with one system it can often be translated smoothly to another system, this is due to the shared fundamental concepts such as: windows, application containers, icons, and menus .

In the early 1970s and 1980s, a diverse range of user interface architectures existed, each tailored to specific hardware platforms. However, the developers found this unacceptable, as they would have to rewrite a significant part of their code for porting it to different platforms. This dilemma paved the way for the creation of the X Window System, designed to provide a hardware-independent interface. The architecture of X has left a mark on subsequent operating systems. Even though GUIs of different OSs are going undergoing rapid changes their core architecture is based on the X architecture [1].

The windowing system is a vital component of all the GUI architecture, as it manages various sections of the display screen separately. At its core is the display server, a central element of the system. When an application presents its GUI within a window, it becomes a client of this display server. Communication between the display server and its clients occurs through an API or dedicated protocols, with the server acting as a bridge between users and applications. Input from the operating system and attached devices, like keyboards or mouse, is channeled through the display server to the appropriate client. The display server handles the output of clients on the computer monitor. This architecture enables users to multitask by working with multiple applications simultaneously. That being said, windowing systems are implemented slightly differently in each operating system and the windowing system for the 3 most popular OS is explored in the following section.

2.1 X Window System (Linux)

Major Linux distributions like Debian & Ubuntu use the X window system as the windowing system for their interface which is based on the X11 protocol which defines the standard for the communication between the X Server and X Client. The communication is made possible with the help of X Protocol libraries such as Xlib or XCB. The X server is responsible for managing GUI of different applications and also inputs from different hardware together. According to the X protocol (as shown in Figure 1), the task of rendering text is designated to the application itself. Even though rendering the texts as images for file managers, browser and terminal [3] are done by the X Server it is only limited to those applications.

Intercepting textual information from within an application is limited as most of the applications are not open sourced and have proprietary code bases and dependencies. The feasible way to extract information would be to intercept data between the X Server and Client. But since these data are graphic pixel data, another layer of image processing should be deployed on top of it, to gain meaningful insight. The added overhead of decoding X11 protocol and image processing limits the performance with respect to the application context.

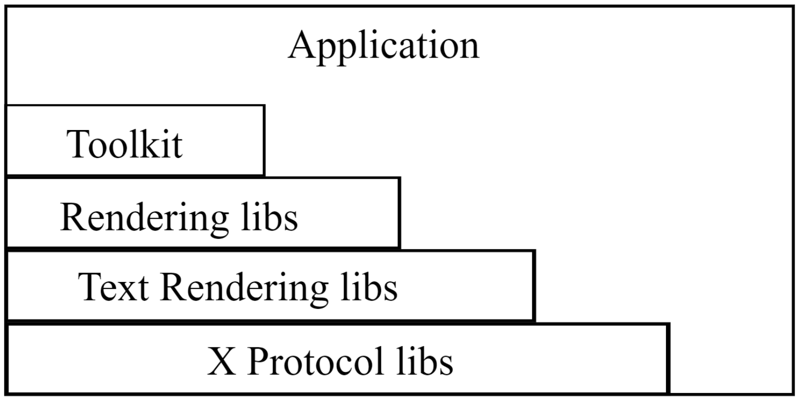


Figure 1. The Structure of an X Client Application [3]

X Server exists as a collection of libraries and components that work together and communication can be made with it using API calls with the help of Xlib or XCB libraries much like how a client would communicate with the server. However, the X Server only lets data move within a hierarchy. All the windows in an X server are arranged in strict hierarchies. At the top of each hierarchy is a root window, which covers the display screens. Each root window is partially or completely covered by child windows [2]. When an application is communicating with the X Server it will have the same hierarchy as the child window to the root window. Therefore in terms of textual data, you can only get data from the root window and any parent windows if they exist between the application and the server.

The final image that is displayed on the screen is rendered to the frame buffer by the X Server and once it's done its responsibility for them has ended [3]. This data can be captured using Toolkits such as Qt [4] and GTK+ which have high level API calls.

2.2 Quartz Compositor (macOS)

Quartz Compositor is the windowing system used by macOS. According to the developer documentation [6], Pixel output generated by other applications using libraries like Quartz 2D, Core Text is directed to a designated buffer, known as the backing store. Then the compositor extracts data from the backing store, constructs different images, and transfers these images to its memory within the frame buffer for display. The Quartz Compositor exclusively deals with raster data and possesses exclusive access to the frame buffer.

The Quartz Compositor also oversees individual windows and acquires bitmap representations of window contents and their placements from the renderer. The Compositor then integrates specified windows into the scene and presents them. Operating as a window manager, it maintains an event queue that receives input events like keystrokes and mouse clicks. By extracting events from the queue, the Quartz Compositor identifies the responsible process for the event's window and subsequently forwards the event to the respective process. In this windowing system too, gathering the complete textual information will involve developing patches for the compositor which involves image processing.

2.3 Desktop Window Manager (Microsoft Windows)

Desktop Window Manager (DWM) is the windowing manager in Microsoft Windows since Windows Vista. Each application has an offscreen buffer that it writes data to and DWM then composites each program's buffer into a final image and writes it to an onscreen buffer much like other OS [5]. DWM gets regular updates about the position of each application and uses it to composite the final image. The positions, name and other details of the applications can also be gathered by using the Win32 APIs. Under the hood DWM uses these APIs to gather information for its final composition.

An Application-defined callback function used with the “EnumWindows” function can be used to access these APIs [7]. When EnumWindows is called in the application, a separate thread managed by the operating system will execute the provided callback function for each window present on the screen. Within this callback function, various win32 APIs can be invoked to gather information about the windows. This includes details like the window's visibility status, whether it's maximized or minimized, its position and size on the screen, as well as its title and class. It is important to highlight that every window has a unique handle, which remains unique throughout its lifespan.

In Microsoft Windows, the way windows are managed is quite similar to other operating systems. Therefore, the challenges faced in directly obtaining text from the screen are similar. To fully capture the text content on the screen, image processing techniques must be employed. However, in the Microsoft environment, the 'enumwindows' callback function can serve as a tool to trace recognized text. This function could potentially help avoid some of the processing overhead in image processing.

3 Text Extraction in Video

There have been recent advancements in the field of text detection in videos. In reference to [8], researchers have conducted a comprehensive survey covering diverse studies and methodologies. This survey stands out because not only does it outline the aspects of detection, tracking, and recognition, but also analyze their interconnections within a unified video text extraction framework. The unified framework introduced here is an improved framework from past surveys and it is shown in Figure 2.

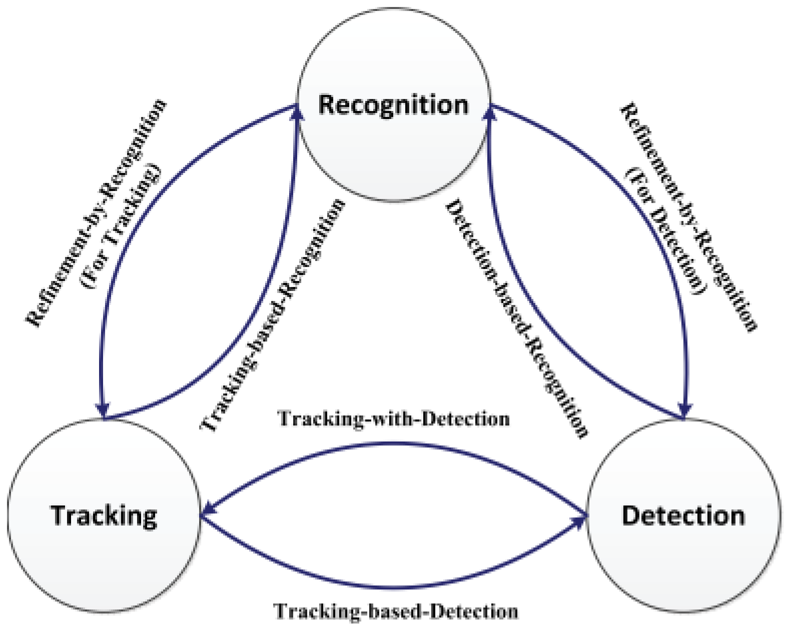

Recognition
Refinement-by-Recognition (For Tracking)
Tracking-based-Recognition
Detection-based-Recognition
Refinement-by-Recognition (For Detection)
Tracking-with-Detection
Tracking
Detection
Tracking-based-Detection

Figure 2. : A framework for text Detection, Tracking and Recognition in video [8].

In this framework, Detection is the task of localizing the text in each video frame with bounding boxes. Tracking is the task of keeping track of the text location as the frame changes and Recognition involves understanding the text by reading it, which is often done using a technique called Optical Character Recognition (OCR).

Conventionally text regions are detected first, using a trained classifier. The classifier may utilize various features including color, edges, gradients, texture to distinguish between text and background. Then the text regions are segmented and then fed into the OCR engines. The segmentation is done to reduce the background noise and false positives as most of the studies surveyed in [8] are based on scene videos, where the background and text region can be very diverse. However there is less diversity in the case of screen captures from the onscreen buffer and it may provide reasonable accuracy performing OCR without having to segment it.

3.1 Text Recognition

Optical Character Recognition(OCR) is the process of converting images of typed, handwritten or printed text into machine encoded text. It is an active field of research in pattern recognition and artificial intelligence due to its important application. The tesseract OCR engine[12] is an open source OCR engine which comes pre-trained for over 100 languages and 35 scripts. It is built on Long Short-Term Memory (LSTM) neural networks, a type of recurrent neural network architecture that is good at handling sequential data. The LSTM architecture enables it to capture contextual dependencies within text, making it less susceptible to variations in font, size, and style. This makes it well-suited for reading textual information from computer screens. While the primary focus of Tesseract has been on scanned documents, its capabilities are well suited to screen-captured text.

3.2 Text Tracking

Another important topic explored in [8] is text tracking across multiple dynamic video frames. As [9] explains, one way to match text from one frame to the next is by measuring how far the recognized word in the current frame is from a possible word in the next frame. In situations like screen captures, where text is often close together in following frames, this feature can be used to improve accuracy. As explained in the previous chapter it is possible to track the text region by tracking the position of the window where the text was first spotted. Text tracking can thus aid the overall text recognition process.

3.3 Text Matching

Text Matching is an additional step that has to be considered after text recognition in this application's context. The sensitive text can lie embedded within a sentence or with other symbols, therefore this is an important step to reach the end to end functionality of this application. In [10] the authors have explored a dynamic programming approach used to calculate the similarity of DNA sequences, DNA sequences are basically a sequence of symbols. Thus the same logic can be applied for text matching here as well. Figure 3. Shows an example of how 2 similar strings such as "ocurrance" and "occurrence" can be rearranged.

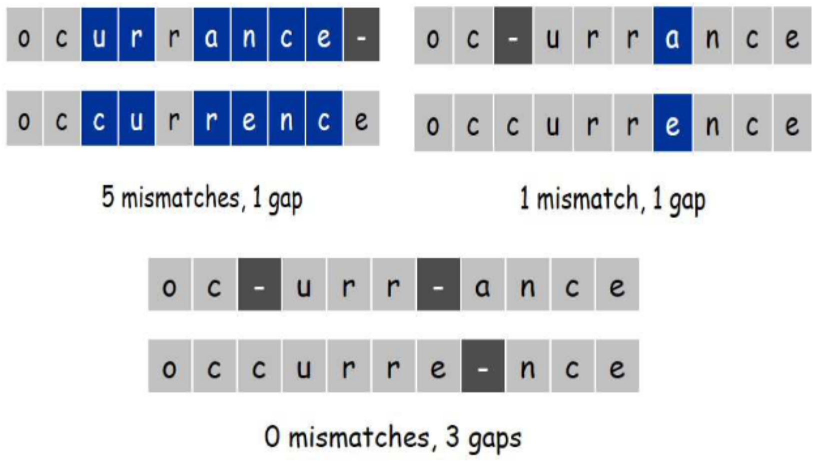


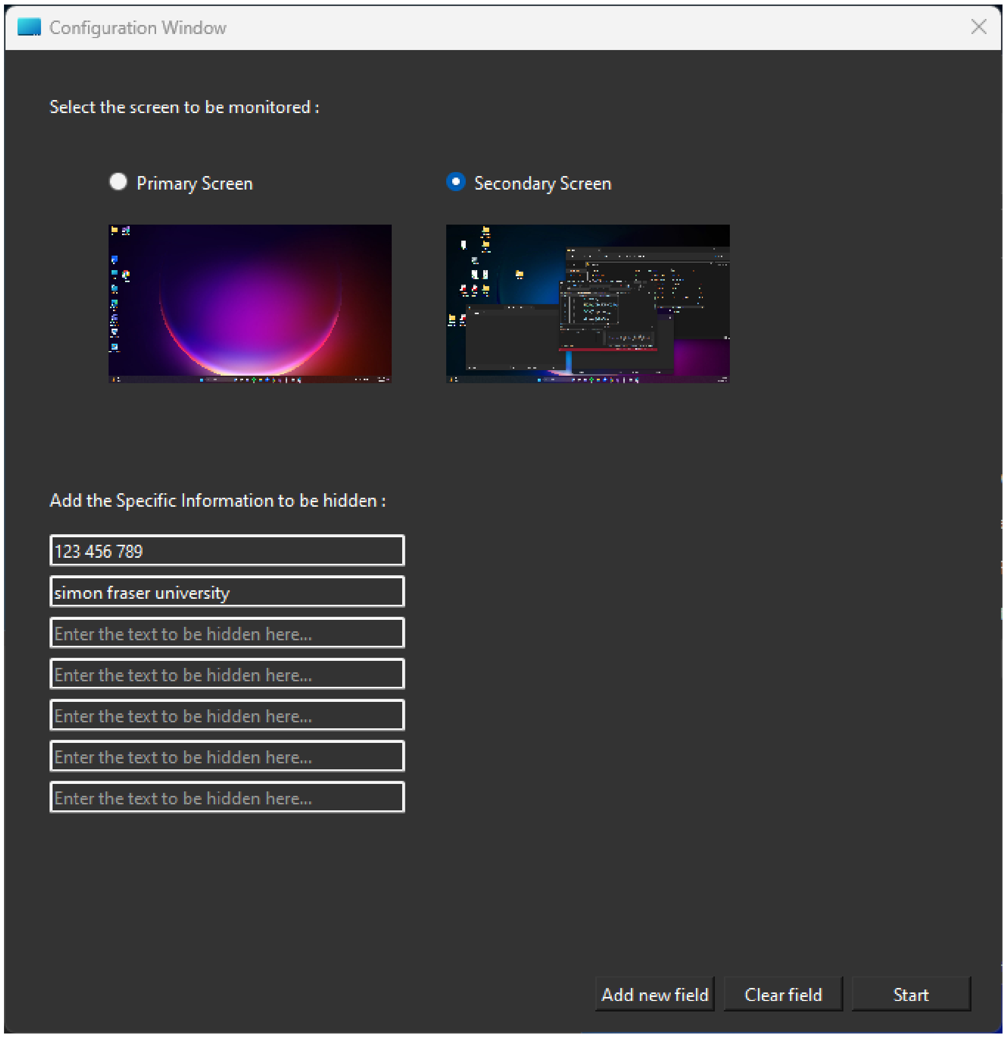
Figure 3. Different ways to arrange 2 sample strings [11]

The two sequences are considered identical if they contain no mismatches or gaps. By assigning a penalty for each mismatch and each gap, the total sum of these penalties becomes a useful measure for determining sequence similarity. The threshold penalty, a customizable parameter, needs careful selection to prevent false positives. The algorithm detailing this process is present in the appendix.

4 Methodology

This application has been developed using the Qt framework. It is a C++ framework and has cross-platform compatibility and pre-built UI elements for the creation of interactive interfaces. The core of the Qt architecture is the concept of signals and slots; it enables information exchange and coordination between different parts of the program. A signal is emitted when a particular event occurs such as a button click or timeouts, this signal can be connected to a slot, a designated function, which responds to the emitted signal.

The Landing window of the application on opening is the configuration window as shown in Figure 4.Within this window, the User inputs the sensitive information and makes a selection regarding the specific screen to monitor. Once this setup is complete, by clicking the 'start' button, the process begins and real-time filtered screen images will be continuously delivered to the User



Select the screen to be monitored : Primary Screen [✓ Secondary Screen]

Add the Specific Information to be hidden :

123 456 789

simon fraser university

Enter the text to be hidden here...

Enter the text to be hidden here...

Enter the text to be hidden here...

Enter the text to be hidden here...

Enter the text to be hidden here...

Figure 4. Configuration window

The multi-threaded architecture of the operation is depicted in Figure 5. The primary thread, denoted as 'Qt' in this context, receives the configured information and concurrently captures the screen. This captured data is then forwarded to the Tesseract OCR engine for text recognition. Tesseract processes the image and extracts information, including text regions and recognized text, which are communicated to the Window tracking module and also back to the main Qt thread.

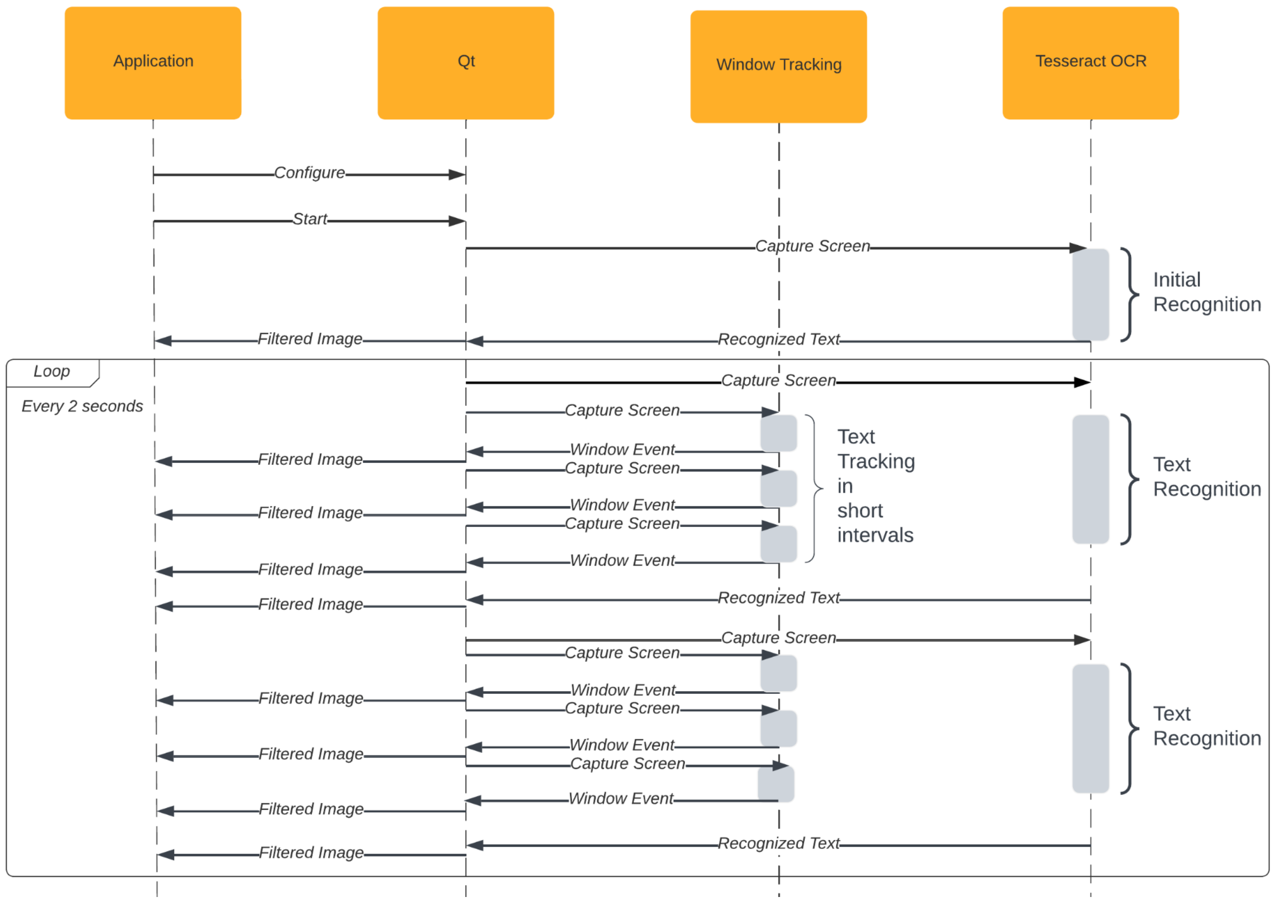


Figure 5. System Sequence diagram

At the primary thread, the obtained data is cross-referenced with the preconfigured text list using the text matching algorithm detailed in the previous chapter. This step determines the specific text region requiring filtering out. Black-filled boxes are painted on the designated areas. The resultant image is presented within the application’s UI. The application also keeps track of the list of windows containing the targeted text regions, this is done by using a unique handle supplied by the Win32 API[7], which exists for every active window.
Following the initial recognition, a timer is activated with a 2-second interval. The timer's timeout event triggers a fresh screen capture, which is fed into the Tesseract engine for another round of recognition. Meanwhile, to counteract the significant delay caused by the Tesseract engine’s processing, the ‘Window Tracking’ component tracks the position of the windows of

interest. This is much faster than tesseract’s recognition process and multiple rounds of tracking can be carried out. The detected window event is passed to the main thread. If a window has shifted since the prior frame, a corresponding adjustment is applied to the associated filtered text region and is then displayed.

This tracking mechanism is crucial in compensating for significant delays caused by the Tesseract engine's processing. Moreover, the system responds to various window-related events differently. If a new window is opened, a window is resized, or the active window changes, these events prompt a re-evaluation by the Tesseract module for text recognition. This precautionary measure acknowledges the diverse possibilities in such scenarios, as text tracking algorithms may fail in preventing the exposure of sensitive information. The timeout also triggers a new text recognition and this process is repeated until the user exits the application.

5 Results and Analysis

This application is designed to work together with collaborative tools equipped with screen sharing capabilities, such as Zoom or Microsoft Teams. To evaluate and explore its capabilities, I conducted tests using Zoom. In Zoom, when a user shares their screen, they can choose to share a specific window or their entire desktop. The recommended way to use this software is while screen sharing, select the ScreenGuard application's window to share. Essentially, by selecting the ScreenGuard window for sharing, the application enables the sharing of the entire desktop screen that was selected in the configuration as seen in Figure 4.

For an optimal experience, it is recommended to use this application with a dual-monitor setup. This configuration provides users with the advantage of interacting with the screen being shared, enhancing the overall usability and effectiveness of the ScreenGuard application. A typical setup of how the application functions is shown in Figure 6. The primary screen on the left is the workspace where the user can interact and the secondary screen has the ScreenGuard application maximized which can be shared through zoom. In having a dual-monitor setup the infinity-mirror effect can be avoided which will happen if the application is within the desktop screen that was configured to be monitored

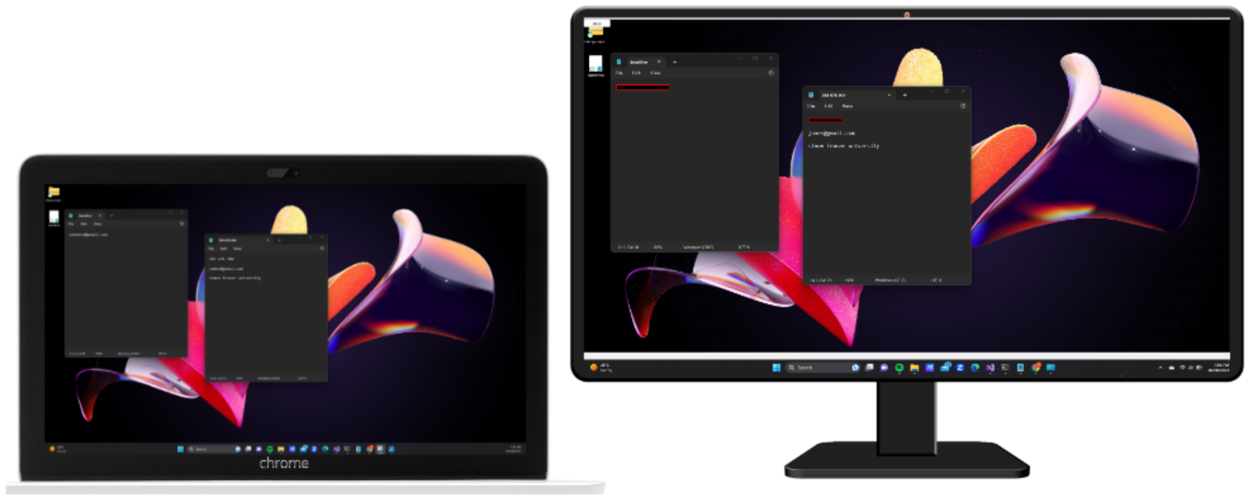


Figure 6. Recommended use of the application

5.1 Timing Benchmarks

The benchmark results were observed on the following environment :

CPU	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz (2.42 GHz)
RAM	8 GB
Operating System	Windows 11

Table 1 : Benchmarking environment details

The timing results were obtained after a computer reboot and the measurements were only taken after the processes were allowed to settle. The time taken for text recognition by the tesseract OCR engine is measured here for various screen images.

Type of Screen Image	Time taken (millisecond)
Sparse Text (Notepad with 1 line of text)	312 ms
Code editor (Full screen contains text)	627 ms
Web Browser (Email Inbox)	645 ms

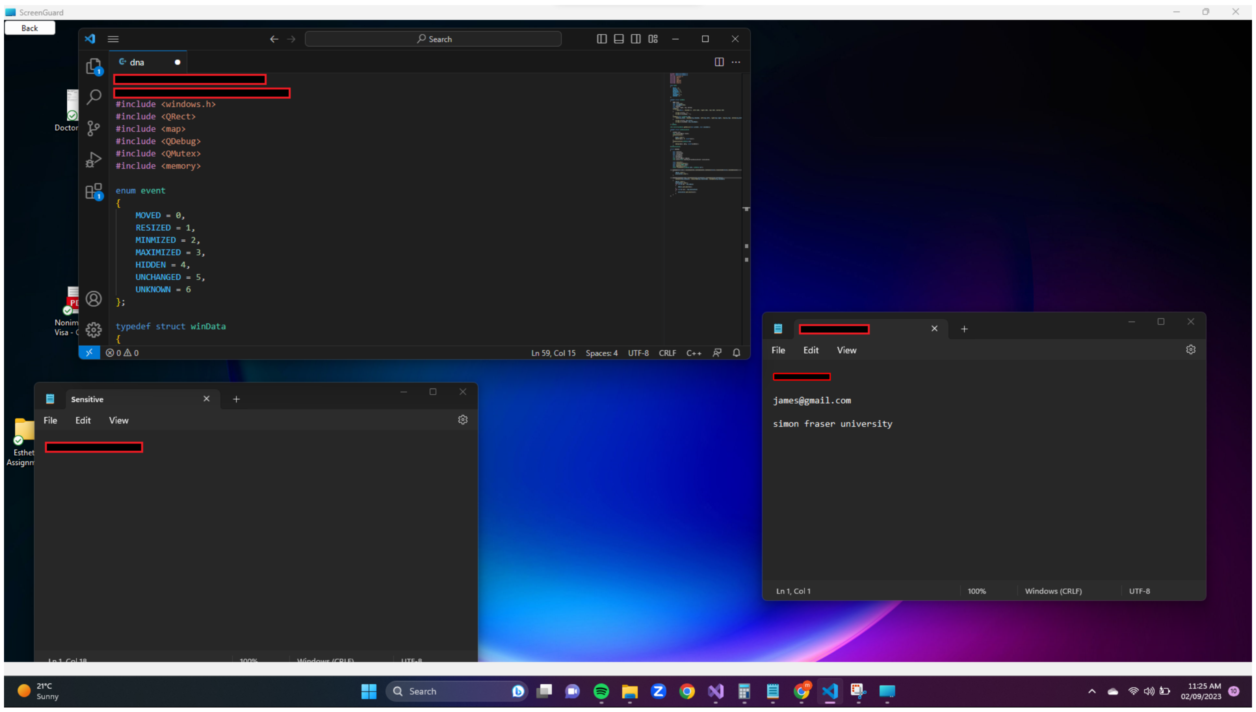
Table 2 : Benchmark Results

The time required for recognizing a single image typically falls within the range of 300 to 700 milliseconds. While a more robust processor could potentially reduce recognition time, it's important to note that this application operates alongside other processes. Thus, to avoid overloading the processor and maintain a smooth user experience, the text tracking algorithm is integrated.

5.2 Accuracy in different Scenarios

In the following sections the accuracy and reliability of the application under various conditions are analyzed.

5.2.1 Text Editors and Code Editors



```
[redacted]
[redacted]
#include <windows.h>
#include <QRect>
#include <map>
#include <QDebug>
#include <QMutex>
#include <memory>

enum event
{
    MOVED = 0,
    RESIZED = 1,
    MINMIZED = 2,
    MAXIMIZED = 3,
    HIDDEN = 4,
    UNCHANGED = 5,
    UNKNOWN = 6
};

typedef struct winData
{
```

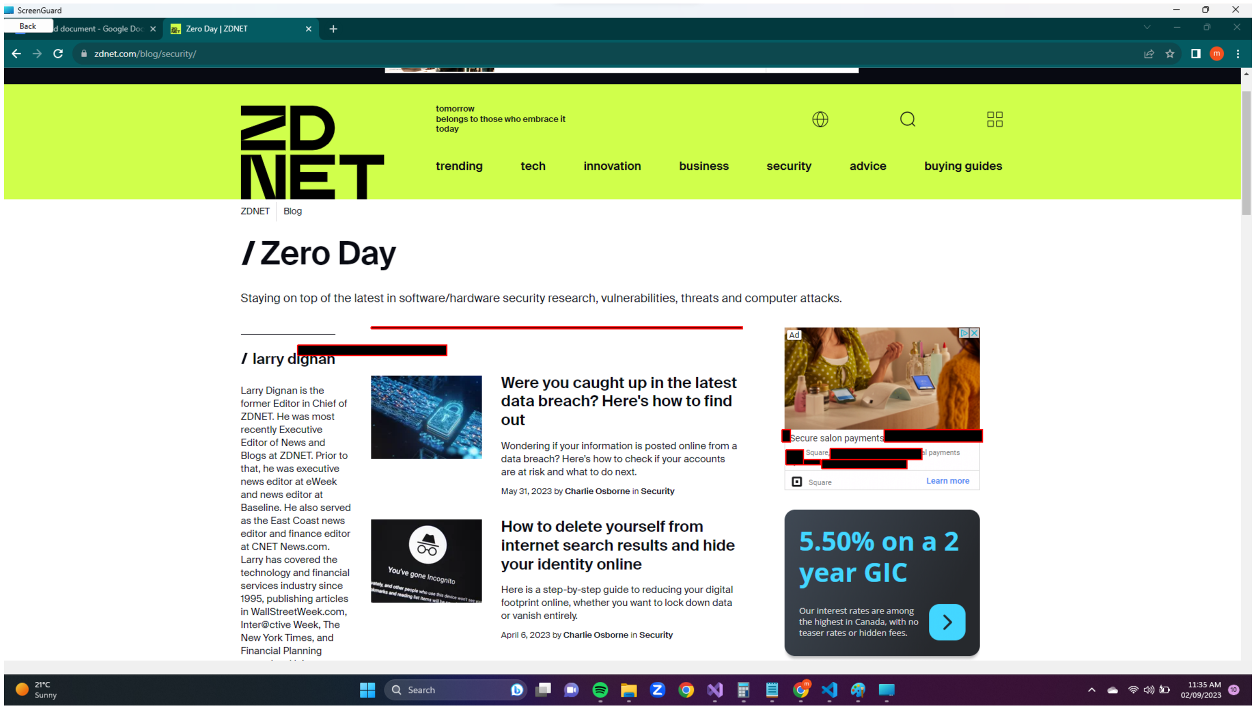
```
[redacted]

[redacted]

james@gmail.com

simon fraser university
```

Figure 7: Test for Text Editors and Code Editors

Figure 7. Shows the application in action which is monitoring text editors such as notepad and also from VS Code. In this scenario the application performs the best and is able to recognize and block all the text that is similar enough to the configured text string list.

5.2.2 Web Browsers

Figure 8 displays the output when the application is applied to a sample blog website. Here, configuration is made for the detection of specific terms like "innovation," "latest posts," and "GIC" as sensitive information. The application effectively detects and tracks standalone text like "latest post". However, when these words are embedded within sentences, the text matching process struggles to identify similarity. Additionally, some other recognized text fields are also blocked due to their similarity to the configured terms.


ZDNET | Blog

/ Zero Day

Staying on top of the latest in software/hardware security research, vulnerabilities, threats and computer attacks.

/ larry dignan [redacted]

Larry Dignan is the former Editor in Chief of ZDNET. He was most recently Executive Editor of News and Blogs at ZDNET. Prior to that, he was executive news editor at eWeek and news editor at Baseline. He also served as the East Coast news editor and finance editor at CNET News.com. Larry has covered the technology and financial services industry since 1995, publishing articles in WallStreetWeek.com, Inter@ctive Week, The New York Times, and Financial Planning

Were you caught up in the latest data breach? Here's how to find out

Wondering if your information is posted online from a data breach? Here's how to check if your accounts are at risk and what to do next.

May 31, 2023 by Charlie Osborne in Security

How to delete yourself from internet search results and hide your identity online

Here is a step-by-step guide to reducing your digital footprint online, whether you want to lock down data or vanish entirely.

April 6, 2023 by Charlie Osborne in Security

Secure salon payments [redacted]

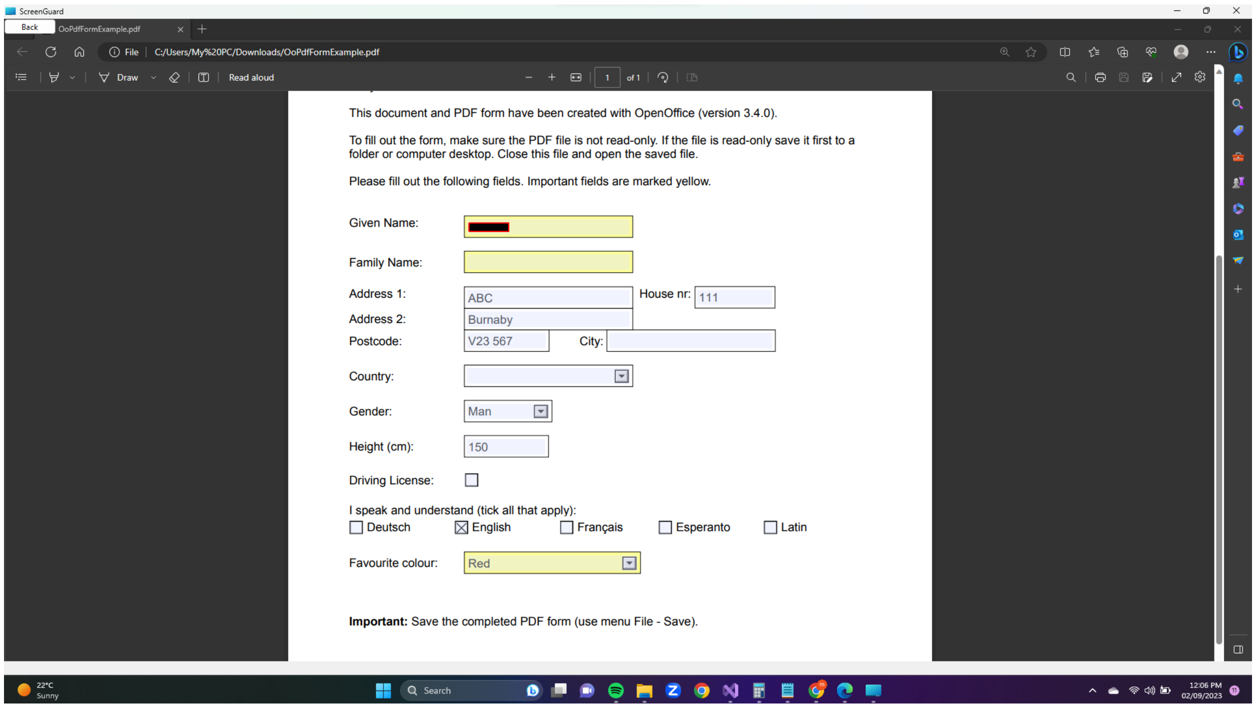
[redacted]

5.50% on a 2 year GIC

Our interest rates are among the highest in Canada, with no teaser rates or hidden fees.

Figure 8. Test for Web browser

5.3.3 PDF Documents


This document and PDF form have been created with OpenOffice (version 3.4.0).

To fill out the form, make sure the PDF file is not read-only. If the file is read-only save it first to a folder or computer desktop. Close this file and open the saved file.

Please fill out the following fields. Important fields are marked yellow.

Given Name:	[redacted]
Family Name:	
Address 1:	ABC
House nr:	111
Address 2:	Burnaby
Postcode:	V23 567
City:	
Country:	
Gender:	Man
Height (cm):	150
Driving License:	☐

I speak and understand (tick all that apply):
☐ Deutsch ☒ English ☐ Français ☐ Esperanto ☐ Latin

Favourite colour: Red

Important: Save the completed PDF form (use menu File - Save).

Figure 9 : Test for PDF Document

Figure 9 showcases the application's monitoring of a sample PDF form. In this scenario as well, words embedded within sentences pose a recognition challenge, but the application effectively identifies and filters standalone words.

6 Conclusion

In conclusion, this project provides a comprehensive approach to safeguarding sensitive information within the context of screen sharing sessions. Through the strategic utilization of Tesseract's OCR's real-time text recognition, and the dynamic tracking mechanism, the application efficiently identifies and masks sensitive data. While the application demonstrates strong accuracy in straightforward text scenarios, its performance in more diverse contexts is less reliable. Therefore, it may not be suitable for the average consumer market; however, it proves to be a valuable asset for businesses dealing with screen content rich in native text.

Looking ahead to potential enhancements, several areas could benefit from improvement. Textual information can be gathered before rendering select applications, since this only includes a fraction of all the possible applications others still would require image processing. To optimize image processing, one approach is to segment the images and subsequently apply Tesseract OCR for text recognition. This segmentation strategy might accelerate Tesseract's performance since there is an overhead of the segmenting algorithm. An investigation into this method holds potential for further refinement.

References

[1] Kent, A., & Williams, J. G. (1996). Encyclopedia of Microcomputers: Volume 19 - Truth Maintenance Systems to Visual Display Quality. CRC Press. Retrieved June 8, 2017, from ISBN 9780824727178.

[2] C. Tronche. "The Xlib Manual.". [Online]. Available: https://tronche.com/gui/x/xlib/introduction/overview.html. [Accessed: Aug 15, 2023].

[3] A. Coopersmith, M. Dev, and the X.Org team (Eds.), "The X New Developer's Guide." Edited by B. Massey, Portland, Oregon, USA, March 2012.

[4] The Qt Company. "Qt 6.5 All C++ Classes". [Online]. Available: https://doc.qt.io/qt-6/classes.html. [Accessed: Aug 15, 2023].

[5] G. Schechter, "Under the hood of Desktop Window Manager," Greg Schechter's Blog, MSDN Blogs. [Online]. Available: https://blogs.msdn.microsoft.com/greg_schechter/2006/03/22/under-the-hood-of-desktop-window-manager/. [Accessed: Aug 20, 2021].

[6] Apple. "Quartz." Available :https://developer.apple.com/documentation/quartz [Accessed: Aug 20, 2021].

[7] Microsoft. "Windows API - Win32 apps." Available : https://learn.microsoft.com/en-us/windows/win32/api/_winmsg/. [Accessed: Aug 20, 2021].

[8] X. Yin, Z. Zuo, S. Tian, and C. Liu, "Text Detection, Tracking and Recognition in Video: A Comprehensive Survey," in IEEE Transactions on Image Processing, vol. 25, no. 6, pp. 2752-2773, June 2016, doi: 10.1109/TIP.2016.2551158.

[9] P. X. Nguyen, K. Wang, and S. Belongie, “Video text detection and recognition: Dataset and benchmark,” in Proc. IEEE Winter Conf. Appl. Comput. Vis., Mar. 2014, pp. 776–783.

[10] J. Kleinberg and E. Tardos. Dynamic Programming. In Algorithm Design,(Eds 1), Cornell University, pp 278-290.

[11]. Qianping, G. (2023, August 10). "Dynamic Programming" Presented in Design and Analysis of Algorithms (CMPT 705), Simon Fraser University, August 10, 2023.

[12] Tesseract-OCR. (2023, Aug). Tessdoc: Tesseract Documentation. [Online]. Available:https://github.com/tesseract-ocr/tessdoc

Appendix

1. Dependencies

The application relies on various build dependencies to function effectively. These include Qt, specifically Qt Core, Qt GUI, and Qt Widgets, which collectively manage the application's user interface and its core operations. Additionally, Tesseract OCR is employed for text recognition, and OpenCV assists in image conversion during internal processes. The window tracking functionality is currently built on the Win32 API, rendering this application compatible exclusively with the Microsoft Windows operating system.

2. Tesseract APIs Used

int Init(const char *datapath, const char *language);

- Tesseract initialization function
- Returns zero on success and -1 on failure.
- Parameters :
  - Datapath - File path for folder containing trained data files
  - Language - Language of choice (eg; eng)

void SetPageSegMode(PageSegMode mode);

- Set the page segmentation mode
- Parameter:
  - Mode - enum value for a specific mode

void SetImage(const unsigned char *imagedata, int width, int height,
int bytes_per_pixel, int bytes_per_line);

- Provide an image for Tesseract to recognize
- Parameters:
  - Imagedata - Raw image data
  - Width - width of the image
  - Height - height of the image
  - Bytes_per_pixel - set as 4
  - Bytes_per_line - set as 4 * width

int Recognize(ETEXT_DESC *monitor);

- Recognize the image from SetImage and the output is kept internally until next SetImage
- Returns 0 on success.
- Parameters:
  - Monitor - set as 0

ResultIterator *GetIterator();

- Get a reading-order iterator to the results of LayoutAnalysis

bool BoundingBox(PageIteratorLevel level, int *left, int *top, int *right,
int *bottom) const;

- Returns the bounding rectangle of the current object at the given level
- Returns false if there is no such object at the current position.
- Parameters:
    - Level - level obtained from *GetIterator
    - Left - Left most position
    - Top - Top most position
    - Right - Right most position
    - Bottom - Button most position

3. Win 32 APIs Used

BOOL EnumWindows(WNDENUMPROC lpEnumFunc, LPARAM lParam);

- Enumerates all top-level windows on the screen by passing the handle to each window, in turn, to an application-defined callback function.
- Returns nonzero value if the function succeeds.
- Parameters:
    - lpEnumFunc - A pointer to an application-defined callback function.
    - lParam - An application-defined value to be passed to the callback function.

BOOL IsWindowVisible(HWND hWnd);

- Determines the visibility state of the specified window.
- If the specified window, its parent window, its parent's parent window, and so forth, have the WS_VISIBLE style, the return value is nonzero.
- Parameters:
    - hWnd - A handle to the window to be tested.

int GetClassNameA(HWND hWnd, LPSTR lpClassName, int nMaxCount);

- Retrieves the name of the class to which the specified window belongs.
- If the function succeeds, the return value is the number of characters copied to the buffer
- Parameters:
    - hWnd - A handle to the window and, indirectly, the class to which the window belongs.
    - lpClassName - The class name string.
    - nMaxCount - The length of the lpClassName buffer.

BOOL GetWindowRect(HWND hWnd, LPRECT lpRect);

- Retrieves the dimensions of the bounding rectangle of the specified window.
- If the function succeeds, the return value is nonzero.
- Parameters:
  - hWnd - A handle to the window
  - lpRect - A pointer to a RECT structure that receives the screen coordinates of the upper-left and lower-right corners of the window.

int GetWindowTextA(HWND hWnd, LPSTR lpString, int   nMaxCount);

- Copies the text of the specified window's title bar
- If the function succeeds, the return value is the length, else it is zero
- Parameters:
  - hWnd - A handle to the window or control containing the text.
  - lpString - The buffer that will receive the text.
  - nMaxCount - The maximum number of characters to copy to the buffer

BOOL GetWindowPlacement(HWND hWnd,WINDOWPLACEMENT *lpwndpl);

- Retrieves the show state and the restored, minimized, and maximized positions of the specified window.
- If the function succeeds, the return value is nonzero.
- Parameters :
  - hWnd - A handle to the window.
  - Lpwndpl - A pointer to the WINDOWPLACEMENT structure that receives the show state and position information.

4. Text Matching algorithm:

```
bool isSimilar(std::string x, std::string  y)
{
   bool flag = false;
   int i, j, pxy = 1, pgap = 1;

   int m = x.length();
   int n = y.length();

   int** dp = new int* [n + m + 1];
   for (int i = 0; i < n + m + 1; ++i)
      dp[i] = new int[n + m + 1];
```

```
    for (i = 0; i <= (n + m); i++)
    {
        dp[i][0] = i * pgap;
        dp[0][i] = i * pgap;
    }

    for (i = 1; i <= m; i++)
    {
        for (j = 1; j <= n; j++)
        {
            if (x[i - 1] == y[j - 1])
            {
                dp[i][j] = dp[i - 1][j - 1];
            }
            else
            {
                dp[i][j] = std::min({ dp[i - 1][j - 1] + pxy ,
                                      dp[i - 1][j] + pgap    ,
                                      dp[i][j - 1] + pgap });
            }
        }
    }

    if (dp[m][n] < 5) flag = true;

    for (int i = 0; i < n + m + 1; ++i) {
        delete[] dp[i];
    }
    delete[] dp;

    return flag;
}
```