electronics

MDPI

Article

A Deep-Learning-Based Approach to Keystroke-Injection Payload Generation

Vitalijus Gurčinas, Juozas Dautartas, Justinas Janulevičius, Nikolaj Goranin and Antanas Čenys *

Department of Information Systems, Faculty of Fundamental Sciences, Vilnius Gediminas Technical University, LT-10223 Vilnius, Lithuania

* Correspondence: antanas.cenys@vilniustech.lt

Abstract: Investigation and detection of cybercrimes has been in the spotlight of cybersecurity research for as long as the topic has existed. Modern methods are required to keep up with the pace of the technology and toolset used to facilitate these crimes. Keystroke-injection attacks have been an issue due to the limitations of hardware and software up until recently. This paper presents comprehensive research on keystroke-injection payload generation that proposes the use of deep learning to bypass the security of keystroke-based authentication systems focusing on both fixed-text and free-text scenarios. In addition, it specifies the potential risks associated with keystroke-injection attacks. To ensure the legitimacy of the investigation, a model is proposed and implemented within this context. The results of the implemented implant model inside the keyboard indicate that deep learning can significantly improve the accuracy of keystroke dynamics recognition as well as help to generate effective payload from a locally collected dataset. The results demonstrate favorable accuracy rates, with reported performance of 93–96% for fixed-text scenarios and 75–92% for free-text. Accuracy across different text scenarios was achieved using a small dataset collected with the proposed implant model. This dataset enabled the generation of synthetic keystrokes directly within a low-computation-power device. This approach offers efficient and almost real-time keystroke replication. The results obtained show that the proposed model is sufficient not only to bypass the fixed-text keystroke dynamics system, but also to remotely control the victim's device at the appropriate time. However, such a method poses high security risks when deploying adaptive keystroke injection with impersonated payload in real-world scenarios.

Keywords: keystroke dynamics; machine learning; deep learning; behavioral biometrics; keystroke recognition

Citation: Gurčinas, V.; Dautartas, J.; Janulevičius, J.; Goranin, N.; Čenys, A. A Deep-Learning-Based Approach to Keystroke-Injection Payload Generation. *Electronics* **2023**, *12*, 2894. https://doi.org/10.3390/electronics12132894

Academic Editors: Younho Lee and Huy Kang Kim

Received: 7 June 2023
Revised: 26 June 2023
Accepted: 28 June 2023
Published: 30 June 2023

1. Introduction

Cybercrime is a challenging topic to deal with, requiring a technically advanced toolset and high-competence staff operating it to be able to detect, mitigate, and perform forensic analysis on a plethora of available technologies and scenarios to achieve it. However, in some cases, performing a forensic investigation becomes very challenging. Among these cases are keystroke-injection attacks. Because machines typically trust the input devices plugged into them, such as keyboards and mice, it is hard to distinguish between legitimate user input and input emulated by a rogue device. If an attacker gains physical access to the computer using social engineering and installs such malicious hardware without leaving any physical trace, detecting such change can be challenging, if not next to impossible.

Upon connecting a malicious USB device to a machine, the system detects and recognizes the device by relying on the information presented by the malicious USB device. Subsequently, the system determines the claimed identity of the device and assigns it a high initial trust level as the default setting. In such scenarios, malicious devices are often detected and categorized as input devices (e.g., keyboard, mouse, ethernet adapter, etc.). If no protective tools are used, the machine will trust these devices by default. Even with the

current USB Type-C specification and its insecurity, it is possible to pretend that the USB device is an ethernet adapter or a device with even more functionality, as well as obtaining the status of the user interface [1,2]. Malicious payloads of these devices will be injected as keystrokes or mouse clicks at high speed, gaining the objective of the attack.

Such attacks have resulted in the development of defensive methods to protect against them. Some of them are based on hardware solutions, such as adapters, that block HID (human interface device) input from USB devices. Software solutions vary from basic solutions that follow a rule-based routine to more advanced ones that provide reasoning based on neural networks. Basic systems usually block any USB devices with an unknown signature since they can be identified by their vendor_id and product_id fields in the device descriptor. More complex and advanced systems use reasoning mechanisms that can identify a person behind a keyboard by their typing patterns. When these systems are installed, the user must define the action a system should take in the case of triggering of an intrusion alert. USB ports may be blocked temporarily or until a legitimate user authenticates itself with a password, etc.

Some forensic tools have been developed to address this issue. A good example of such a tool is the DuckHunt tool, which uses post-mortem memory dumps on Windows 10 machines. After Hak5's rubber ducky attack, traces of PowerShell scripts can be located in the memory as part of the DOS command [3].

This paper aims to implement deep-learning methods to generate payloads for keystroke injection from small datasets by utilizing a low-computational-power device implanted inside the keyboard. Furthermore, this paper points out how to enhance the security of keystroke-based authentication systems, while addressing the potential risks of keystroke-injection attacks. To ensure the legitimacy of the investigation, a working hardware-based model of dataset collection and payload generation is proposed and realized within this context. The working model includes a physical keyboard with an implant to collect a user's biometric typing data with functionality to inject later-generated payload according to the requirements of the threat actor and a tool for generating payload to impersonate a user's biometric typing pattern. The research shows good performance with 93–96% accuracy in fixed-text scenarios and 75–92% accuracy in free-text scenarios. In addition, the payload was successfully generated and validated. These results demonstrate the precision of the proposed approach to predicting and producing payloads in fixed- and free-text contexts. On the other hand, the proposed model could be used for second- or multi-factor authentication with slight modifications. The results of this research raise a discussion about the positive effects of machine-learning techniques for keystroke dynamics recognition and the negative side of malicious devices which use similar machine-learning techniques to bypass keystroke dynamics security.

The second chapter of this research includes a review of related work concerning keystroke-injection attacks and the methods employed. This chapter explores various approaches, techniques, and strategies utilized by researchers in the realm of keystroke-injection attacks and detection methods. The third chapter presents our proposed model along with its distinctive characteristics, and understanding of its architecture and functionality. The fourth chapter focuses on the results and discussions derived from the research conducted, providing analysis and interpretations of the findings.

2. Related Work

USB-based keystroke-injection attacks involve manipulating USB devices to inject malicious keystrokes into the target system. These attacks exploit the trust of USB devices and can bypass traditional security measures if they are taken into account and adapted to the systems designed to prevent such attacks. By impersonating a keyboard or using programmable USB devices, attackers can execute unauthorized commands or gain unauthorized access to sensitive information mimicking legitimate user keystrokes. Different attack vectors, such as BadUSB [2,4] and rogue device [5] attacks, have drawn attention to the potential risks and ramifications involved in these types of attacks. However, the

emergence of advanced attack methods necessitates the development of more sophisticated countermeasures. These attacks pose a significant security risk and highlight the importance of implementing strong defenses to mitigate the potential impact of such exploits, especially those that can bypass keystroke dynamics systems using rogue USB devices with implants.

The related work paragraph adopts a four-part structured approach that addresses each specific aspect of keystroke dynamics and provides key elements of this work. The first part focuses on hardware solutions, especially USB imperfections. The second part refers to possible attacks that exploit USB imperfections. The third part discusses keystroke dynamics solutions and the techniques employed to circumvent them. The fourth part is aimed at data requirements for keystroke-related research collection and analysis.

2.1. USB Imperfections

Infrastructure security is a natural starting point for a data security plan [6]. Most USB hardware attacks are closely associated with social-engineering tactics, as these types of attacks often necessitate a compromise of physical security measures. In 1998, the first widely supported USB protocol USB 1.0 was released with a data-transfer rate of 1.5 Mbit/s. There was an updated version—USB 1.1 supported two data-transfer rates, low-speed 1.5 Mbit/s and full-speed 12 Mbit/s. Due to the limitations imposed on transfer speeds, the standard in question only supported a restricted range of devices, such as keyboards and mice. Then, in 2000, the USB 2.0 specification was introduced. High-speed (480 Mbit/s) mode meant that devices such as cameras, external storage devices, printers, and network cards were also supported. The convenience of the high data-transfer rate provided the momentum for the popularity of USB flash drives. Although various peripherals are supported in USB 2.0, there is no reliable way to identify the type of device by 'vendor_id' or 'product_id' [7,8]. The absence of robust identification mechanisms creates a vulnerability that can be exploited by keystroke-injection attacks. USB 3.0 and its 2013 update USB 3.1 introduced the USB Type-C connector. This provided a unified connector type for power, HDMI, display port, and Thunderbolt. For example, USB type-C can transfer video stream data, and USB 3.1 cables can deliver 4K (UltraHD) video and audio. This can potentially enhance keystroke-injection attacks as it provides the attacker with visual access to the victim's machine, enabling them to observe the ongoing activities, choose the best time for the attack or even extract a greater amount of data from the victim within a limited timeframe [2]. However, no improvements in the field of security were introduced in these revisions. This allows even more possibilities for successful attacks via the USB interface [1,2]. Utilization of USB for malicious intents should not be solely understood in terms of USB versions and direct physical connections. USB attacks should be comprehended in a broader context. If there are means for directly detecting harmful USB devices, even with moderate difficulty, the attack surface significantly expands when malicious USB devices are connected through intermediate circuits such as USB hubs or other commonly used expansion devices. To demonstrate this weakness, research has been conducted on the creation of a malicious USB device to bypass USB blocking mechanisms by manipulating USB protocol and spoofing data to trusted USB hubs [9]. This shows a diverse range of offensive attack capabilities, as well as the corresponding countermeasures in both direct and side-channel scenarios.

More than 400 vulnerabilities related to USB peripherals are listed on the CVE (common vulnerabilities and exposures) list. As a result, it has become a standard practice for an attacker to use these vulnerabilities and exploit the trust-by-default characteristics of USB to conduct attacks. And this is a security risk for the private, government, and personal sectors [2]. New and more advanced USB technologies offer more features which are used by mobile devices and tablets, as well as computers.

A taxonomy of USB-based attacks has been developed by categorizing them into three main categories: programmable microcontrollers, USB peripherals, and electrical [10]. This

classification is used to analyze USB hardware attacks and gain a deeper understanding of each type of attack in this paper.

1. Electrical attacks are related by nature to the denial-of-service (DoS) attack. As an example, there is a hardware device called the 'USB killer'. It is an electrical discharger disguised as a simple USB device. Once this device is plugged into the USB port, the capacitors will charge up and discharge a critical amount of current back to the USB port in intervals, making computer hardware components unusable;
2. USB peripheral attacks utilize flash-drive firmware or driver to deliver malicious payloads. These types of attacks can perform buffer overflows, DNS overrides, and even keystroke injection and, in some cases, launch an executable;
3. Microcontroller attacks use microcontrollers that emulate keyboards and can inject keyboard input at high. They belong to a class of HID attacks that has evolved over the years and became much more sophisticated [11].

Other researchers proposed a USB-based attacks taxonomy with a more granular approach by creating different bases that cover adversary intentions, object of impact, attack mechanism, level of secrecy, level of complexity, assets, and so forth [12]. A recent study discusses attacks on keyboard-firmware attacks in the banking sector, where vulnerable software inside keyboard controllers is used to sniff sensitive data [13]. As follows, this highlighted the need for secure USB protocols and firmware-verification mechanisms. The keyboard has become a more security concern over the years due to the sensitive nature of its use.

2.2. USB Attacks

USB attacks encompass a variety of techniques that exploit vulnerabilities in the USB interface to gain unauthorized access or compromise system security. Keystroke injection, as a specific type of USB attack, involves injecting malicious commands or keystrokes into a target system to perform unauthorized actions or gain elevated privileges. The USB keystroke-injection attack is also known as the keypress-injection attack and the keyboard-injection attack [14–16]. It is an attack method where by connecting a malicious USB device it is possible to enter predetermined keystrokes in the terminal, enter and execute scripts, use keyboard shortcuts to control the device, and, thus, maliciously affect the computer [17–19].

The significance and feasibility of keystroke-injection attack are derived from several factors. One factor is USB device detection, or what happens before a malicious USB device begins to perform malicious activity; it is essential as further options of keystroke injection will depend on the level of trust gained. When a USB device is plugged into the USB port, the host detects that a new device was connected and waits for 100 milliseconds to ensure, that the new device has the time to be powered properly. The host then issues a reset command to place the device in its default state and allow it to respond. The host will ask the device for the first 64 bytes of its device descriptor. This step is very important to conduct a successful keystroke-injection attack. The device descriptor contains information about the product, its vendor, required power levels, the number of interfaces, endpoint information, etc. Once these are established, the host will communicate with the USB device using the appropriate drivers [20]. The utilization of replacement descriptors and static device information in USB penetration testing has been used for some time in automated platforms to prepare such devices [21]. Merely pretending that the device is highly trustworthy will not be sufficient to bypass protected systems without implementing additional measures. A USB storage device is completely unsuitable for this type of attack, as a simple device-protection tool can identify it as a threat or authenticate every USB flash drive [22,23].

The vast majority of researchers who researched keystroke-injection-hardware-based solutions used microcontrollers [15,24–27]. In addition to microcontrollers, some research used more powerful small-factor computers that offer additional capabilities [14,28]. Accordingly, first and foremost, in order to inject keystrokes, a microcontroller must identify

itself as a HID (human interface device). Researchers recently engaged in developing a model that establishes a correlation between HIDs and vulnerability categories, thereby aligning them with specific types of attacks [29].

2.3. Keystroke Dynamics and Its Circumvention

When such HID- and BadUSB-type attacks emerged, serious concerns were raised regarding the security of USB devices. Consequently, over the past four years, the research on protective solutions has increased, but also the number of researchers investigating how to circumvent these measures. One of the commonly discussed keystroke-injection-attack-detection solutions outlined in scientific papers is rule-based. Rule-based keystroke-injection protection is a straightforward and widely adopted method that involves logging and monitoring keyboard input for detection. On the contrary, an alternative approach entails analyzing the USB packet traffic to identify and mitigate potential malicious activities [30]. Typically, it is beyond human capacity to type at an extremely high speed of thousands of words per minute. Therefore, when the system detects abnormal keypress speeds, it can trigger various protective measures, such as disabling the keyboard, requiring password input, or logging the activity for further analysis [17].

Certain rule-based systems rely on white-listing known vendors by USB ID and not allowing other HID devices to function unless the user explicitly confirms and authorizes them [4]. Another approach involves employing contextual analysis tools that utilize a combination of contextual events. These tools take advantage of a heuristic approach by executing a USB drive within a sandboxed or isolated environment for a brief period, typically, a few seconds. During this time, the system monitors the processes and actions initiated by the USB drive. Subsequently, an evaluation phase is performed to determine whether the observed actions exhibit malicious behavior [31].

Another advanced method employed to detect and mitigate keystroke-injection attacks involves the utilization of behavioral biometrics, particularly keystroke dynamics. The concept of keystroke dynamics originated in the 1970s, primarily focusing on analyzing fixed-text data.

Keystroke-dynamics-based systems identify users based on their interaction with a computer via input devices. Therefore, in some literature, keystroke dynamics can also be referred to as keystroke biometrics. These systems typically use neural networks that are trained using user interactions with computers. Systems that use this for identification may be prompted to enter their password multiple times or type a paragraph of text, allowing the system to train and learn from these inputs. Alternatively, the system can train itself in the background while the user performs its daily tasks, continuously refining its understanding of the user's keystroke dynamics. Some of these systems may be vulnerable to what is called a frog-boiling attack [32]. This implies that the dataset used for identification purposes can be poisoned with small packets containing false data and these would lead to false identification. Such poisoning attacks can have two different objectives: to reduce the performance of the model and to manipulate the model by injecting false data [33]. Within the context of keystroke injection, our research and test results obtained show that malicious devices can accurately mimic the input of legitimate users.

Keystroke dynamics does not require any additional hardware as required by other biometric authentication methods, such as fingerprints or facial recognition. Personal keystroke biometrics is difficult to forge and could be used for authentication [34,35]. For example, the online learning platform Coursera applies this method to authenticate online students [36]. It is worth mentioning that legitimate users may be blocked by the system in cases where training patterns will not match the input patterns in cases like nervous emotional state, hand injury, or, simply, a different keyboard or software update. Other scientists use certain characteristics and behaviors that impede accurate recognition as specific indicators when determining Parkinson's disease using fixed- and free-text writing habits [37]. This shows us that for accurate recognition behavioral patterns need to be

updated regularly. However, these problems can be solved, and it is very likely that keystroke biometrics will gain more popularity in the near future [38].

There exist a limited number of approaches to implement keystroke analysis which vary depending on the intended purpose. In general, keystroke dynamics can be divided into two main steps: training neural networks from collected user data and performing authentication. Moreover, keystroke dynamics can be classified into two distinct types: fixed-text and free-text. The key difference between these two methods is the dataset that is used to authenticate a user [39,40]. Fixed-text authentication is mostly used as a second-factor or multi-factor authentication. One of the leaders in commercial keystroke dynamics authentication systems is a company called TypingDNA. And, according to this company, the average fixed-text login credentials (email and password) contain about 30 characters [41].

Compared to fixed-text, free-text is a continuous type of authentication rather than second- or multi-factor. The user's computer interaction is continuously monitored and compared to existing user data for analysis and comparison. If a predetermined threshold of deviance from typical user behavior is passed, then a user might be blocked, logged out, or asked to authenticate himself based on system settings [42]. Other researchers consider analyzing both keystroke and mouse usage behavior patterns to prevent a situation where an attacker avoids detection by restricting to one input device because the system only checks the other input device [43]. Conversely, alternative proposals state that mouse dynamics requires simpler hardware to capture the biometric data without using sensitive user data from the users and propose a method based on mouse dynamics based on deep-learning for continuous and silent user authentication [44]. In our case, the collection of sensitive data is a limitation of our proposed method while we collect actual data inputs.

Free-text is more dynamic and uses a self-adaptive dataset. Typically, users are asked to type a few paragraphs of text or the required data could be collected simply by monitoring how the user interacts with a keyboard on a daily basis [45]. These systems are used for continuous authentication and are often used to defend against keystroke-injection attacks. The evaluation of accuracy and performance in free-text keystroke dynamics is an ongoing area of investigation; researchers are actively developing methods to minimize error rates in this domain [46], while other researchers have explored the application of cGAN networks to generate fake keystroke dynamics patterns with the intention of deceiving keystroke-authentication systems [47]. Given the growing need for remote learning, continuous authentication with keystroke dynamics was implemented and performed very effectively [48], although keystroke dynamics has significant inaccuracies when applied to RDP (remote desktop protocol) and VNC (virtual network computing) systems, resulting in poor or non-functioning operation.

In modern approaches for fixed- or free-text keystroke dynamics, different types of neural networks are being used in conjunction. As an example, Siamese neural networks are used [49]. In this model, two neural networks are trained using the same parameters and weights and work in conjunction with one another. One neural network receives original legitimate user data and another receives data that should be verified. Later, these results are compared. In contrast, generative adversarial neural networks are also gaining momentum for attacks against user identification systems [47,50]. In this architecture, two neural networks work against each other. One is trained as a discriminator and tries to identify the user while the generator receives random noise input at the beginning and tries to generate output that would fool the discriminator. Despite the aforementioned, in certain scenarios statistical algorithms have demonstrated superior performance compared to deep-learning approaches, particularly when dealing with large volumes of unlabeled data [51].

In subsequent years, Bayesian classifiers based on the mean and variance of time intervals between two or three consecutive keypresses were applied to the problem. The results claim a classification accuracy of 92% on a dataset with 63 users [38]. In situations where we only want to implement this as a 2FA (password—what we know and keystroke

biometrics—what we are) dataset for training needs to contain data about the parameters of how a user enters his username and password. This means that analyzed text is usually short (in average 8–20 characters long) and that the user will be asked to enter the same text (in this case, his password) for a set number of times. There are several drawbacks associated with the use of keystroke dynamics for continuous authentication. One limitation is that the data collected from user input often consists of a limited set of characters, such as letters, numbers, or symbols. Therefore, if a user changes his password (which is recommended for security purposes), the model would need to be retrained to adapt to the new input patterns. This retraining process can be time-consuming and may introduce delays in the authentication process. Additionally, reliance on fixed-text input may not capture the full range of user behavior and typing patterns, limiting the overall accuracy and effectiveness of the system.

Recently, research in keystroke dynamics has been heavily focused on machine-learning techniques, including random forests, fuzzy logic, RNN (recursive neural network), CNN (convolutional neural network), Gaussian mixture models, *k*-nearest neighbours (*k*-NN), *K*-means clustering, and many other approaches [51,52].

The two main ideas used to make a convolutional neural network particularly successful are sparse connections and weight sharing. According to the study, activation functions (ReLu, Maxout), loss functions (SoftMax, hinge), regularization technique (dropout), optimization method (data augmentation, batch normalization), and fast processing (sparse convolution) were used in conjunction with CNN [53]. As an alternative there are long short-term memory (LSTM) networks and a variation of the LSTM, called a gated recurrent unit (GRU). GRUs have a simpler design with fewer parameters, which allows for quicker training due to the reduced number of operations. On the other hand, convolutional neural networks (CNNs) are predominantly used for image-related tasks like processing, classification, segmentation, and pattern identification. However, they have also demonstrated impressive results in various other classification tasks.

Depending on the machine-learning model and algorithms used, the data from the user has to be modified to fit the algorithm accordingly. A good example could be RNN (recurrent neural network). This automatically learns time series and has shown good performance in applications such as speech recognition, document abstraction, and NLP (natural language processing). However, if RNN is used, the keystroke data must be vectorized [36].

In a typical implementation, keystrokes or keystroke pairs are converted into vectors, and neural network weights are adjusted by backpropagation during the model-training phase. The trained model is then used during the authentication phase to assign a probability to the observed typing pattern for a specific user. The typing pattern is periodically compared to the stored user data, and, based on a predefined threshold, the model determines whether the observed pattern belongs to the legitimate user or not.

2.4. User Keypress Data and Their Minimum

To identify the user hiding behind the keyboard, a dataset with a collection of user keypresses is required. When collecting data solely based on the dynamics of key-presses without additional information, only a limited number of parameters are necessary and collected. Depending on the algorithm and system, some parameters may differ, but Dwell time and Down-to-down time are being used frequently. Dwell time (Dt) is a duration in which a key is pressed down (H.time). And Down-to-down time (DD.time) measures the duration between one keypress to another, Up-to-down time (UD.time) is the time from the release of one key and the press of another (in some literature, called flight time) [34]. The data-collection and training phase of keystroke dynamics can be categorized based on the length and type of the text [54]. Fixed-length models use a limited dataset that consist of username and password. In other words, the analyzed text is replicated in a concise manner and iterated until a sufficient amount of data are collected for training purposes. However, some datasets are available for testing purposes, including the Carnegie Mellon

University (CMU) fixed-text dataset [55]. It is often used as a reference to test techniques in keystroke dynamics research. This dataset consists of 51 participants' keystroke dynamics information, where each participant typed password '.tie5Roanl' a total of about 400 times. Participants had to wait at least one day after certain typing sessions, so that variations of each subject's typing would be captured daily. Furthermore, this password was chosen as an example of a strong 10-character password, which meets common requirements for password security. The dataset compiled by Gonzalez comprises a combination of publicly accessible keystroke datasets [56–58]. It encompasses both authentic human-generated keystrokes and synthesized forgeries [59]. Researchers can utilize this dataset to evaluate the efficacy of liveness-detection techniques for keystroke dynamics, specifically compared to a diverse range of state-of-the-art methods for synthesizing samples. The first dataset included in the dataset collection was from CMU [56]. The second dataset was collected from individuals performing daily tasks in an enterprise setup and was used for evaluation of free-text keystroke dynamics for authentication [57]. The third dataset, obtained from anonymous subjects through a crowd-sourcing platform, was aimed at identifying indicators of fraudulent intent by analyzing variations in typing patterns [58]. It is important to note that the datasets referenced in the literature sources do not contain recorded click values.

The minimum number of data needed for fixed-text and free-text keystroke dynamics varies depending on factors such as the complexity of the analysis model, desired accuracy, and specific characteristics of users and their typing behavior as there could be many users with similar typing biometrics. Considering the focus of this research on small datasets and drawing insights from the existing literature, we can conclude that the minimum requirement for fixed-text keystroke data ranges from 10 to 30 samples, while for free-text keystroke data it ranges from 50 to 150 samples. It is important to highlight that the datasets referenced in the literature sources do not contain recorded keypress values, except those used to collect specific fixed-text data, such as the password mention in the CMU dataset.

A trend has started to emerge recently wherein facial recognition is combined with keystroke dynamics (fixed-text) as a MFA (multi-factor authentication) system [60]. This solution, although seeming promising, needs to take into consideration that not all devices have cameras and there are various types of keyboards and their layouts. An alternative approach suggested by the researchers was to propose a defense in depth strategy by implementing a three-factor authentication system. This multi-layered approach involves utilizing a passcode as the first factor of authentication, a password as the second factor, and keystroke dynamics as the third factor [61]. Such an approach requires the use of even more diverse datasets to increase the reliability of results.

In summary, comparative studies, including machine-learning-based methods and rule-based methods to evaluate the effectiveness of different keystroke-injection-detection and -prevention mechanisms, demonstrate that injection methods can be accurate enough to bypass both fixed-text and continuous authentication systems where the level of biometric accuracy does not need to be as precise as physical biometric data. Keystroke dynamics system armoring, which uses fake keypress data in order to distinguish between artificial keystrokes and real ones, and methods which follow the patterns of keypresses along with the timings have shown good results in recent years, but require a big collection of user data and come with a high false-rejection rate. Furthermore, the evolving landscape of keyboard-injection attacks requires ongoing research to eliminate emerging threats and to develop effective countermeasures. Keystroke-injection attacks are capable of jumping the air gap of secured infrastructures that have a high security level with the help of social engineering. Therefore, the demand for tools that will help defend against these types of attacks is growing over the years. The reliability of these tools should be tested regularly. And these systems may not always work because new attack methods emerge and adapt to increased security standards.

With this in mind, this study focuses on a more complex situation, i.e., using small datasets when generating payload and injecting them with a low-power device to bypass

both fixed- and free-text keystroke dynamics systems. A small dataset is essential for integrating an accurate deep-learning solution into a compact device, such as a keyboard implant, as exemplified in this research.

3. Method for Keystroke-Injection Detection and Payload Generation

To achieve the objectives of the study, an approach has been devised to detect keystroke injection and to generate keystroke payloads. The evaluation of the results obtained will ascertain the accuracy of keystroke dynamics authentication systems and the usability of the generated payloads.

3.1. Keystroke-Injection Detection

To elucidate and prepare for keystroke-payload generation, an analysis of the fixed-text injection solution was conducted. For empirical experiments concerning the authentication of fixed-text keystroke biometrics, TypingDNA API for web applications is used [62].

This integration uses JavaScript in the backend to collect user keystroke dynamics data during the sign-up and login phases. During sign-up (see Figure 1), the user is required to enter his username or email and password a few times. This procedure yields sufficient data to establish a user's keystroke biometric profile, which can then be employed as a reference for comparison with the keystroke data gathered during the login process.

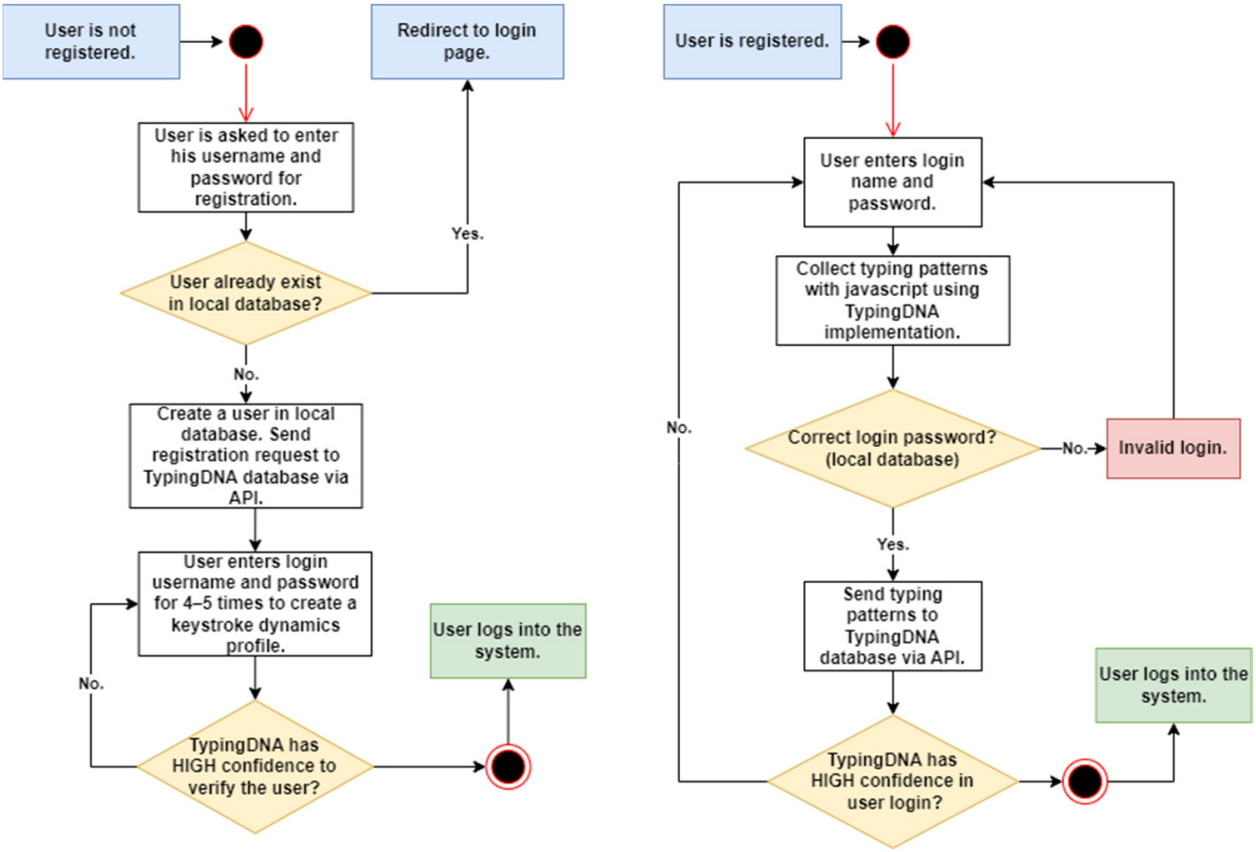


Figure 1. TypingDNA registration and login processes.

After acquiring the keystroke data and constructing the user profile, a login attempt can be initiated. During this phase, the user must enter his login credentials (fixed-text) and if they are correct then a second factor authentication will commence (see Figure 1). Vectorized keystroke data are sent in JSON format to be verified by TypingDNA. Following this process, a response will be received indicating the degree of confidence that the TypingDNA API assigns to the legitimacy of the user, based on the response generated by the neural network [62,63].

To evaluate the accuracy of keystroke data as a form of two-factor authentication (2FA), a test was conducted involving the creation of 10 user accounts. Each user was assigned a unique username but shared the same password. During the sign-up process, distinct keystroke data were recorded for each user. The TypingDNA test application successfully recognized and authenticated all 10 users based on their individual keystroke patterns.

The choice to employ LSTM in this research is supported by a range of significant factors identified in previous researches and some are provided below. Behavioral authentication utilizing HID devices, such as a keyboard and/or a mouse in conjunction with artificial neural networks, is becoming increasingly popular. There are various ways and architectures of neural networks that are being used for specific tasks [64]. Convolutional and recurrent networks are often used for the purpose of authenticating users by the way in which they interact with a computer. Manny research indicates that the best error-rate values achieved for the preceding experiments regarding CNN and LSTM (a type of RNN) are as follows: CNN, 2.3% and 6.5% (with and without data augmentation); LSTM, 13.6%; and CNN + LSTM, 2.36% or 5.97%, depending on the dataset utilized [65]. Furthermore, the benefits of combining various neural networks can be seen in other fields as well. The combination of CNN and long short-term memory (LSTM) can be used for medical purposes as well [66].

Since scientific research shows that efficient and accurate results regarding keystroke dynamics can be achieved by combining CNN and RNN (in particular, LSTM), it was chosen to implement these types of neural networks in our proposed keystroke-injection-attack model. Figure 2 illustrates a standard LSTM architecture [67]. Information is stored in the cells and memory manipulation is performed through gates. LSTM is highly effective in tasks that involve complex sequential patterns and long-term dependencies or trends in data. Given that LSTM is classified as a recurrent neural network (RNN) and known for its ability to capture and analyzing sequential data, it is particularly suitable for modeling and predicting keystroke dynamics.

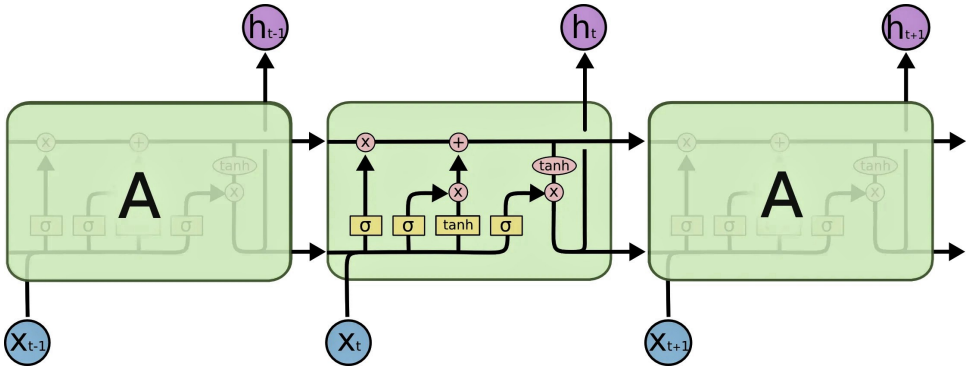


Figure 2. Architecture of LSTM [67].

To test the LSTM model with a fixed-text dataset a "Comparing Anomaly-Detection Algorithms for Keystroke Dynamics" (DSN-2009) was used along the Python deep-learning library TensorFlow. A preliminary evaluation revealed that the error rate of the test set exceeded the anticipated level, indicating a potential case of overfitting. To mitigate this issue, the implementation of the Gaussian dropout technique proved to be effective in preventing overfitting of our model.

SoftMax activation function is used in output layers to achieve a final prediction of which user is behind the keyboard and, for example, if blocking should occur. This function calculates the probability of output neurons to each defined class. However, the neurons in the output layer should be equal to the number of total classes.

$$f(y_i) = \frac{e^{y_i}}{\sum_k e^{y_k}} \quad (1)$$

The model used for the DNS-2009 tests had five layers, each containing 136 neurons. Tanh activation functions were used for all layers except the output layer.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x - e^{-x}} \tag{2}$$

The output layer of the neural network employed the SoftMax activation function for user classification. A batch size of 16, representing the number of samples fed to the neural network simultaneously, was utilized in the experiment. Adam (adaptive moment estimation) optimizer was used. Optimizers update the model in response to the output of the loss function, and they assist in minimizing the loss function. And Adam optimizer is well suited for large datasets. It minimizes the training epochs needed to train our model. In this model, 50 epochs were used (see Figure 3), however, similar results could be achieved with about 40 epochs, and having more than 50 could lead to overfitting.

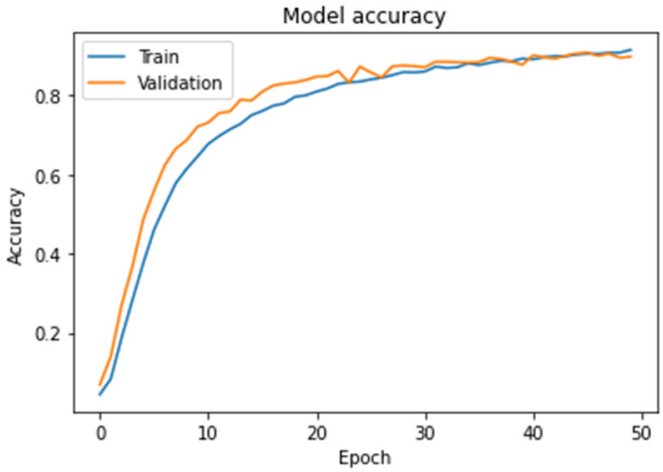


Figure 3. Optimal epoch number for the analyzed dataset using the tanh function.

The trained model with the DSN-2009 dataset managed to reach about 89% accuracy for validating users with the provided data. These findings demonstrate results comparable to those observed in enterprise products. Further experimentation was conducted by substituting the tanh activation function with ReLU (rectified linear unit). However, the 86% accuracy achieved after 50 epochs was consistent with the previous results.

The ReLu activation function can be mathematically defined as follows:

$$f(x) = \max(0, x) \tag{3}$$

This function transforms the output of the neuron to be either 0 or the given value. In essence, it activates the neuron more strongly when the input value is positive.

We can see from Figure 4 that maximum accuracy of ReLu was not reached in 50 epochs. Therefore, we can assume that tanh activation function is more optimal for our model. A similar accuracy (88.9%) was reached after 80 epochs.

To implement the proposed keystroke-dataset collection and keystroke-injection payload generation method, a combination of Raspberry PI Zero 2W and Teensy 3.5 is used. The purpose of using multiple hardware devices is to collect precise keystroke data from the target and establish communication with the command-and-control (C2) centre, where collected data are subsequently transmitted for the purpose of training the neural network, if training, and payload generation requires more computational power.

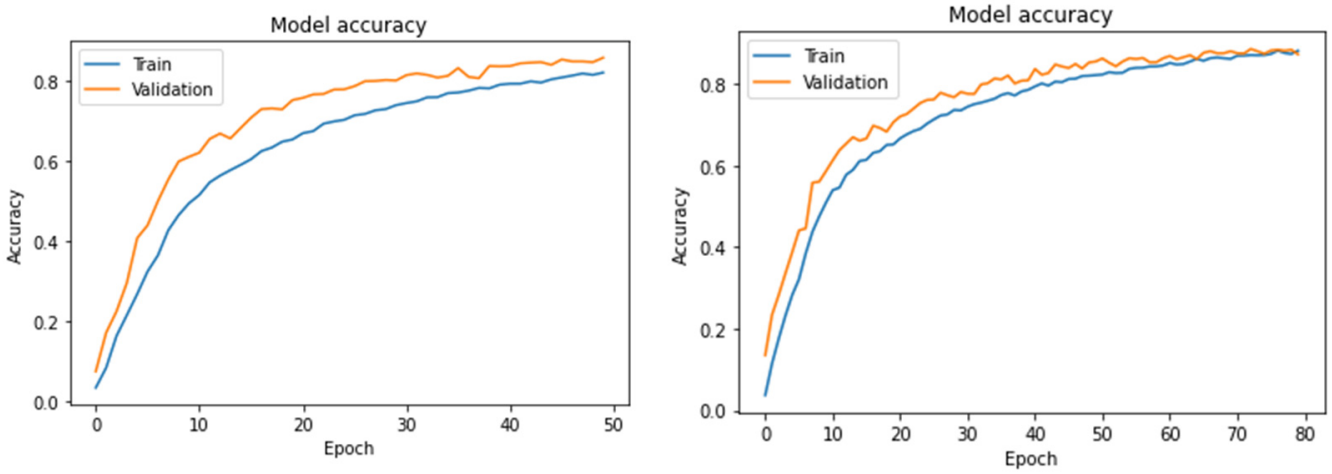


Figure 4. Optimal epoch number for the analyzed dataset using the ReLu function.

If the scenario is consistent with the circumstances, when sufficient data are collected, a payload can be generated that corresponds to the biometrics of the victims' keyboard keystrokes. The payload is sent to the device as a keypress-injection attack using a keyboard. One of the rationales behind adopting such an implant architecture is the ability to receive all input data, irrespective of intended actions, including exceptional stealth capabilities, and present significant challenges in terms of detection. This design decision also enables the testing of data-poisoning scenarios by incorporating predetermined noise into the user's typed inputs.

The Raspberry PI unit receives the input directly from the keyboard. It collects and stores the keystrokes alongside the keystroke biometric data. A dedicated microcontroller is also in place to mimic the HID device and inject payloads. Teensy communicates with the Raspberry Pi unit via a serial communication link and directly relays all keystrokes to the computer. When the attacker wants to launch a keystroke-injection attack, the Raspberry PI sends a payload to Teensy and injects it into the victim's computer. Furthermore, while the attack is being conducted, the physical keyboard is disabled so that if a victim sees this attack taking place, he cannot interfere with it.

An important characteristic of our proposed implant is its ability to emulate the USB vendor and product identifiers. In our research, this capability was employed to target a specific Dell keyboard model for the implant's integration with VENDOR_ID—0 $\times$ 413c and PRODUCT_ID—0 $\times$ 2106, respectively.

An additional objective of the implant was to integrate remote-control functionality for the keystroke-injection device through a 4G or Wi-Fi network. This would grant enhanced control to the attacker, allowing real-time adjustments via an SSH connection. This capability represents an advance over using microcontrollers alone, as it allows functionality to be changed without physical access to the device in any stage of data collection or attack. In our experimental setup, communication between the threat actor and the implant was established through a virtual private network node. However, the listed functionality may change depending on the size of the camouflage and power requirements.

The primary objective of the proposed model is to bypass fixed-text systems using limited datasets, with secondary emphasis placed on free-text scenarios, that are highly dependent on the system's actual accuracy. The limitations of this proposed model were intentionally created by the researchers. These limitations include the implementation of a stealth mode for the implant to minimize detection, ensuring that the device does not appear conspicuous and separate, avoiding the need for additional software and minimizing the storage of data inside the model. In addition, the device would incorporate a secure data-erasure feature to minimize the number of data exposed in the event of detection and forensics. Furthermore, it would possess local capabilities to bypass fixed-text keystroke dynamics solutions when disconnected from the control server by disabling the

external communication module. Moreover, the proposed device would be designed to be universally compatible, allowing for implantation on various keyboards. It would operate with minimal power consumption and offer remote interchangeability of functionalities. Additionally, the device would effectively operate with a small dataset, closely resembling real attack scenarios.

The flexibility of our proposed model is exceptionally high, enabling dynamic modifications of its functionality on the fly. In contrast, in microcontrollers utilized by other researchers, planning and pre-defining the functions and their potential alterations are essential. Under unforeseen circumstances, the proposed model would incorporate measures to prevent the successful execution of the attack or would be intentionally programmed to fail. Additionally, when alternative manipulators or data-input devices are employed, and obtaining all necessary data for the attack is not feasible, our proposed model would activate and remotely control a camera, or respond to pre-defined triggers, ensuring adaptability and maintaining functionality. The control distance of the solution is virtually unlimited where the device will have a connection. The proposed model is designed to be cross-platform-compatible, ensuring seamless functionality across different platforms.

3.2. Implementation of Keystroke-Payload Generation

The proposed keystroke-injection method encompasses four stages (as detailed in Figure 5) and its realization (see Figure 6):

- Data collection—achieved by facilitating the 'spy_k3y' keystroke and key logging tool along with the single-board computer and a microcontroller. The tool hooks up all keyboard events. The setup collects keystroke dynamics data and pressed-key values, logs them, and relays them to the target machine. Additional checks such as if this is a first keypress or a keypress after a long typing break (time longer than 5 s is, in our case, considered to be a break, while other researches use 1–1.5 s) were also implemented. Once the data collection is completed, the exfiltration stage can begin using a secure file-transfer protocol for transmission;
- Extraction of keystroke dynamics data and generating the keystroke-injection payload using collected data;
- Verification of the generated payload using the LSTM neural network. The generated payload is reversed to keystroke dynamics data and compared with the collected data. If this payload is at least 85% similar to the source, the payload is generated successfully and under the command or automatically could be injected impersonating the user's typing pattern. Alternatively, if necessary, the payload is regenerated and the verification process is repeated;
- Committing the verified payload to the remote device (in cases where extensive computational power is unnecessary, the task can be performed directly on the device) and injecting it into the remote machine.

When the collected data file is extracted, the payload-generation process starts with the "snipe_inject" tool. The data are split into separate characters and the tool will also search for special values such as "<enter>", etc. Then, it will query the dataset for values regarding needed keypresses (the tool will check for H.H, D.D.h.i, U.D.h.i, etc.). If searched values are found in the columns of the dataset, 'snipe_inject' will find three values: average, min value, and max value of that keypress. Then, a random value will be generated in between a range of min and max values for H.time and U.D.time (this simulates the time the user would search for the next key). An alternative and more advanced approach to generate the payload with a refined deviation time was examined and tested. However, deploying this method on low-computation devices introduces additional computational load.

In cases where searched value is not present in the dataset file, 'snipe_inject' will guess this value by using previously calculated averages from other keys or key pairs. Based on this, a value tool will generate keypress values for H.time and U.D.time. In the scenario where the first letter of the payload is missing, indicating an empty average array, random values for H.time and UD.time will be generated from predefined hardcoded values within

the dataset. Once this is done, the generated payload in a form of script can be loaded and executed in the keyboard implant.

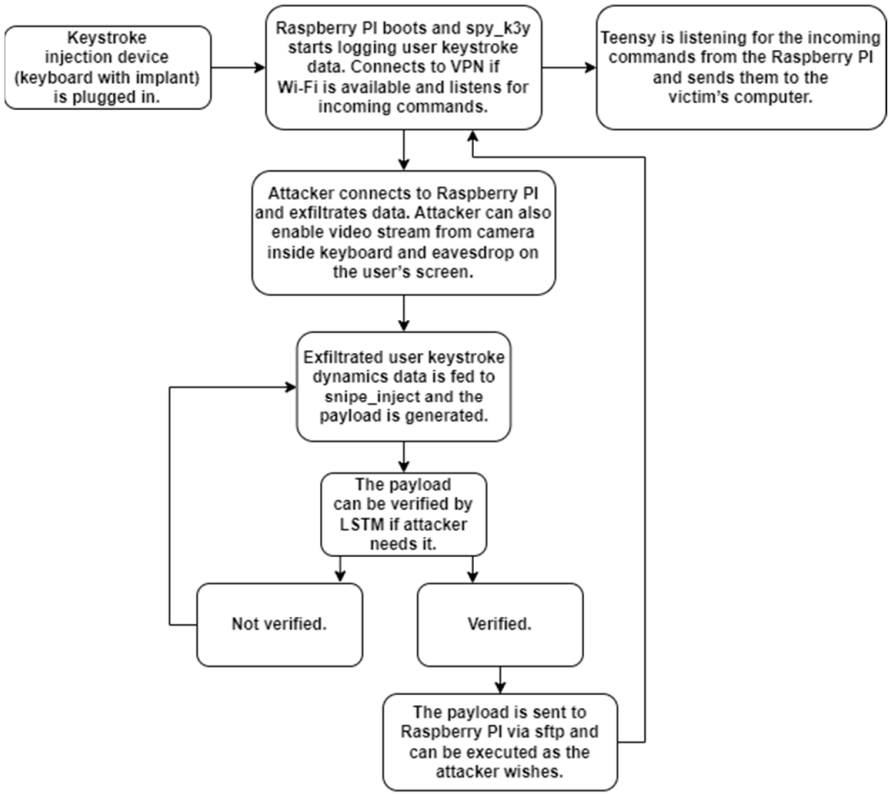


Figure 5. Proposed keystroke-injection attack flow.

Figure 6. Keyboard with implant setup.

There are two parameters required to generate payload based on users' keystroke dynamics: H.time—this indicates how long a single key was pressed; and U.D.time—this is the time between release of the previous key and pressing of another. A key release is initialized by sending a zero byte to the microcontroller. So, first, a command is sent to press a specified key to the microcontroller. The H.time value is generated for that specific key. Then, a command to release that key is sent by sending a zero byte to the microcontroller on a key-up event. Finally, we obtain U.D.time for a specified key sequence. This process is repeated until the end of the payload is reached.

For payload verification, two parameters are used: H.time and U.D.time. Using these values, we can also calculate the D.D.time using this equation:

$$\text{D.D.time}\left(\text{prev}_{\text{key}}\text{next}_{\text{key}}\right) = \text{H.time}\left(\text{prev}_{\text{key}}\right) + \text{U.D.time}\left(\text{prev}_{\text{key}}\text{next}_{\text{key}}\right) \quad (4)$$

This allows us to recreate the data that the security system uses to verify a user. Then, we can see if the keystroke-injection payload is verified and assigned to a user we are trying to impersonate. To reach more accurate conclusions about the effectiveness of the proposed method, it was tested on fixed- and free-text keystroke dynamics.

As seen in Figure 6, a Raspberry PI zero 2W computer and a Teensy 3.5 microcontroller are embedded in a keyboard as an implant for this research. The keyboard controller is directly wired to the USB ports of the computer. Using the aforementioned implant configuration, we gathered the dataset and generated payloads to conduct experiments that involve both fixed- and free-text scenarios. These payloads encompassed login credentials, as well as typed blocks of paragraphs. The generated payloads were thoroughly tested, successfully bypassing the 2FA mechanism by using the generated payload and enabling user impersonation by providing a specific block of paragraph text for payload generation. Both types of payload generation for keystroke-injection attacks were, at least, 90% similar to the victims' typing patterns and are detailed in the next section.

4. Results and Discussion

The results and findings cover the two fields that were analyzed: fixed-text keystroke dynamics and free-text keystroke dynamics. Table 1 presents the datasets utilized for the experiments, along with the key parameters associated with the datasets gathered by the researchers. The table consists of three columns, which, respectively, indicate the quantity of keystrokes in each dataset, the number of users from whom the data were collected, and the count of user errors encountered in the collected data. The reported errors in this context refer to additional keystrokes performed by users who were not part of the original typed text, as well as corrections made to the typed text. To address these inconsistencies, the collected fixed-text dataset underwent a process of normalization and error removal. This was necessary due to the diverse nature of the data collected from multiple individuals.

Table 1. Summary of collected datasets.

	Total Keystrokes	Users	User Error Percentage
Fixed-text dataset	4443	15	±9%
Fixed-text filtered dataset	4070	15	-
Free-text dataset	21,119	20	-

The resulting dataset, referred to as the fixed-text filtered dataset, is presented in the second row of the table. The free-text dataset collected by the researchers is appropriately denoted in the third row of the table. In addition to the datasets showcased in Table 1, two additional datasets were created specifically for experiments to verify the dissimilarity between our generated free-text payload and datasets obtained from other researchers. To facilitate this, a dataset was generated by mixing a portion of the user data collected specifically for this research with a segment of the 136 million keystrokes dataset collected by V.

Dhakal [68], with a total of about 150 thousand keystrokes. Additionally, another dataset was created by combining mixed data sourced from the 136 million keystrokes dataset and A. Mishra's keystroke dataset [69], with a total of about 120 thousand keystrokes.

4.1. Fixed-Text Keystroke Dynamics

For fixed-text keystroke dynamics, we collected a dataset of 15 users while each user entered the same passphrase in several times until they started to display signs of rejection. Keypress timings, showing the first passphrase typed by users x1–x11, are provided in Figure 7. We observed that the first typing is the one that most closely matches the user's pattern.

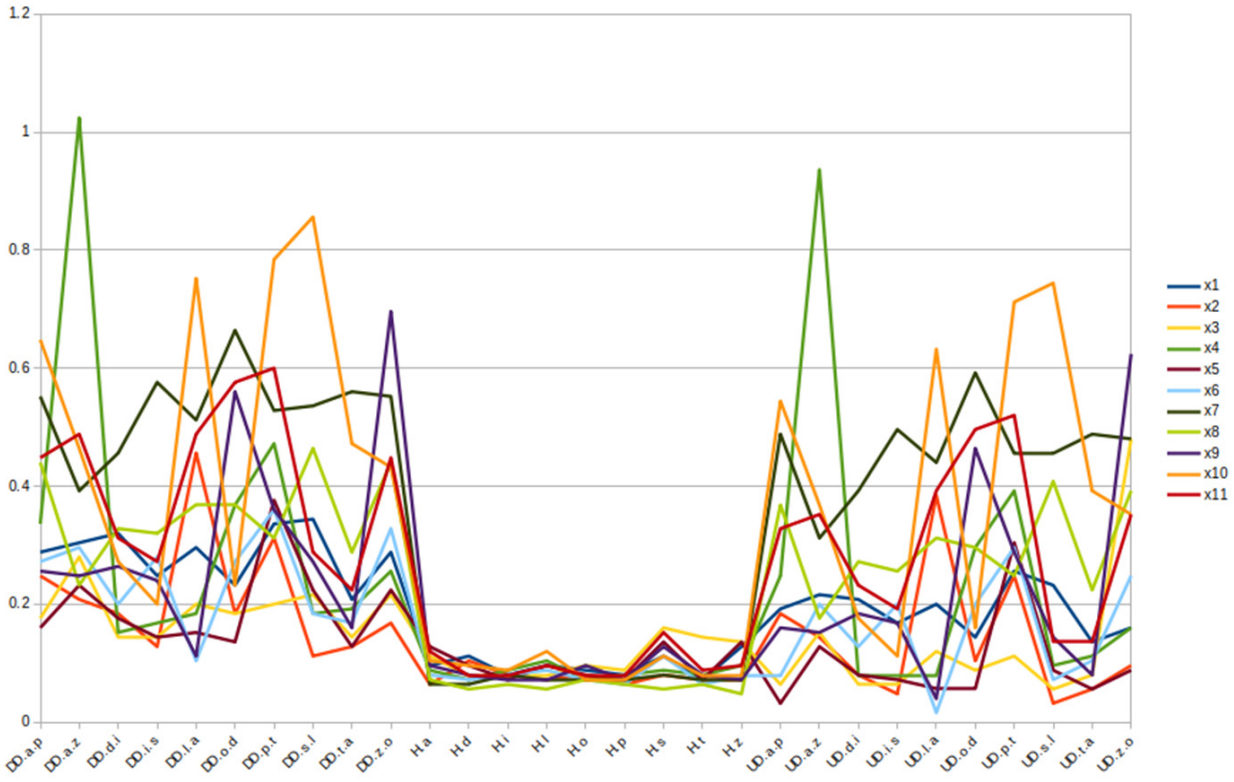


Figure 7. Keypress dynamics with timings, typing the same passphrase by users x1–x11.

As seen in Figure 7, the H.time values are more similar among all users of this dataset. However, the DD.time and UD.time features give us more indications on what the user was typing.

These data were used to train the LSTM network which had the same number of layers as the DSN-2009 tests. To extract more features, each LSTM layer had fewer neurons. The output layer will have the same number of neurons as our dataset has unique users.

After the training phase for the TypingDNA test, we proceeded to provide our model with unlabeled data that included the generated payload, which serves as the test set. The results demonstrated that the LSTM successfully identified the generated payloads as belonging to the user in 9 out of 10 instances. The left section of Figure 8 exhibits a heatmap chart that illustrates a comparative analysis of the five payloads generated for user x2. On the contrary, the right side of Figure 8 displays the timings of 10 user keypresses along with the payload generated for user x2. Notably, the generated payloads adhere to the acceptable accuracy range typically observed in fixed-text systems such as TypingDNA.

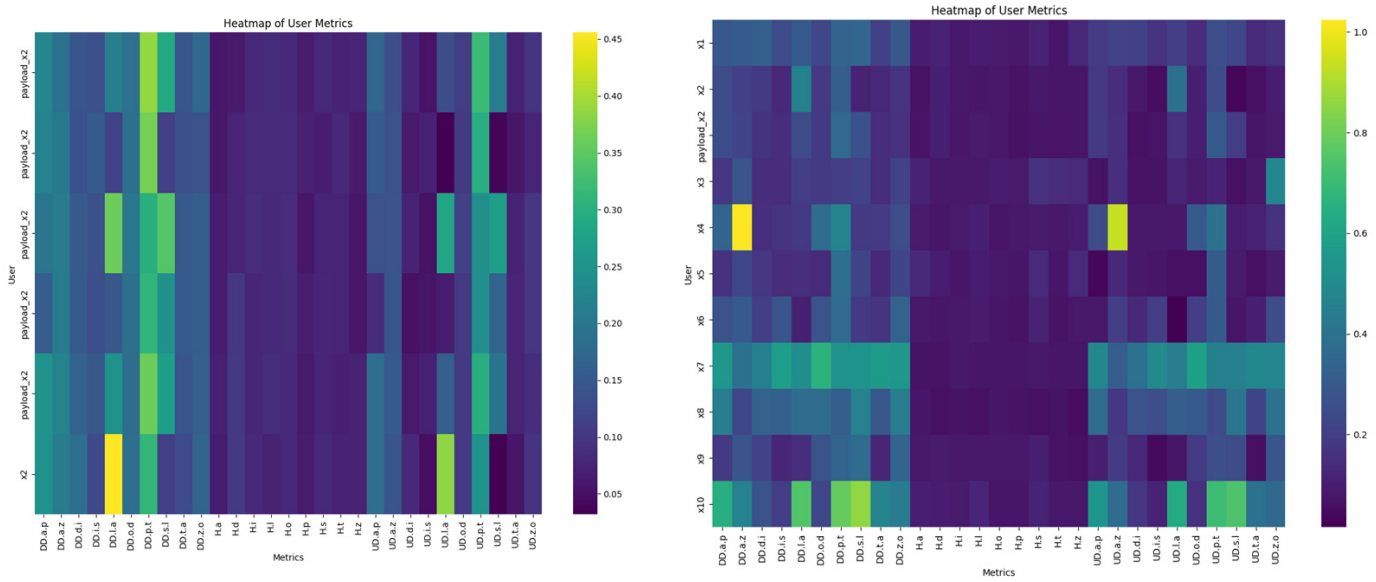


Figure 8. Payloads generated for user x2 comparison heatmaps.

During the training of the neural network with our collected dataset, it reached a maximum accuracy of 90–96%. The train–test split used for fixed-text experiments was 70/30, respectively. Precision, recall, and f1-scores were not equal to or close to one, respectively, except in the case of a few classes. The precision value for class x9 was 0.71. Recall values for the x10 class were 0.94 and the x6 class 0.56, respectively. F1-score notable values obtained for x6 class were 0.71 and x9 class 0.83, respectively. After completion of the training phase, the model was given unlabeled data encompassing the generated payload for a particular user, using a "snipe_inject" tool. The results demonstrated that LSTM reached a recognition rate of ~90% in identifying the generated payloads as coming from the specified user, based on a series of 10 tests. For most tests 150–200 epochs were used. Checkpoints were used to save the best epoch based on the accuracy of the validation and 30% of the dataset was used for validation, as seen in Figure 9. The confusion matrix can be seen in Figure 10. Figure 11 presents the k-fold cross-validation results which show the model's ability to learn and provide consistent reliable outputs. The average accuracy of all folds was 96% and the validation accuracy of all folds was 93%. The standard deviation of all folds was 1.7% and 3%, respectively.

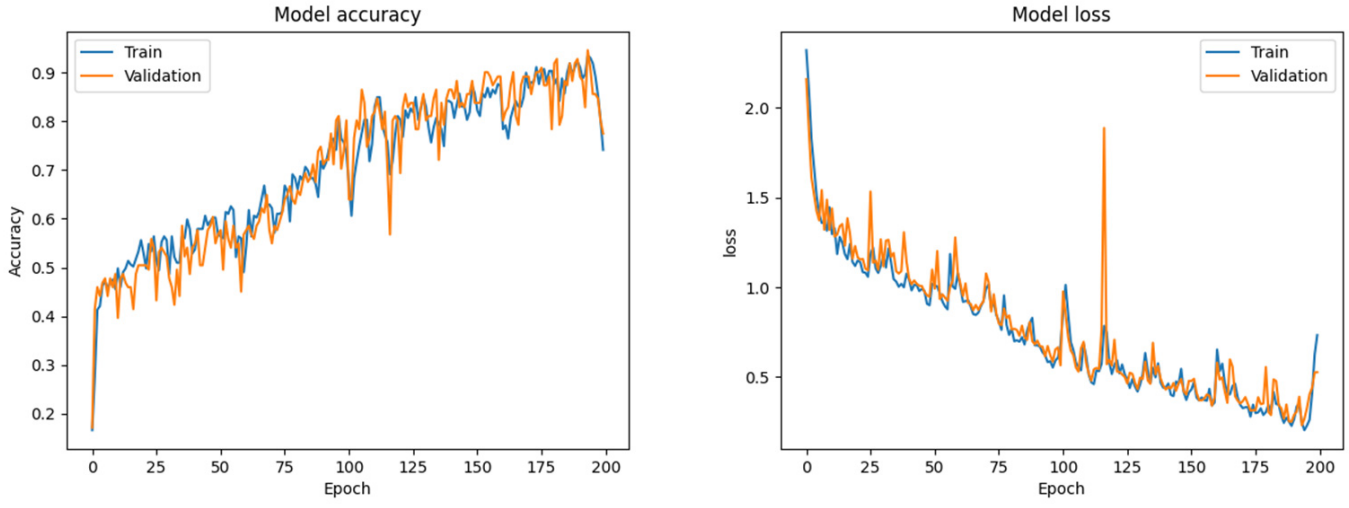


Figure 9. LSTM accuracy and loss after 200 epochs.

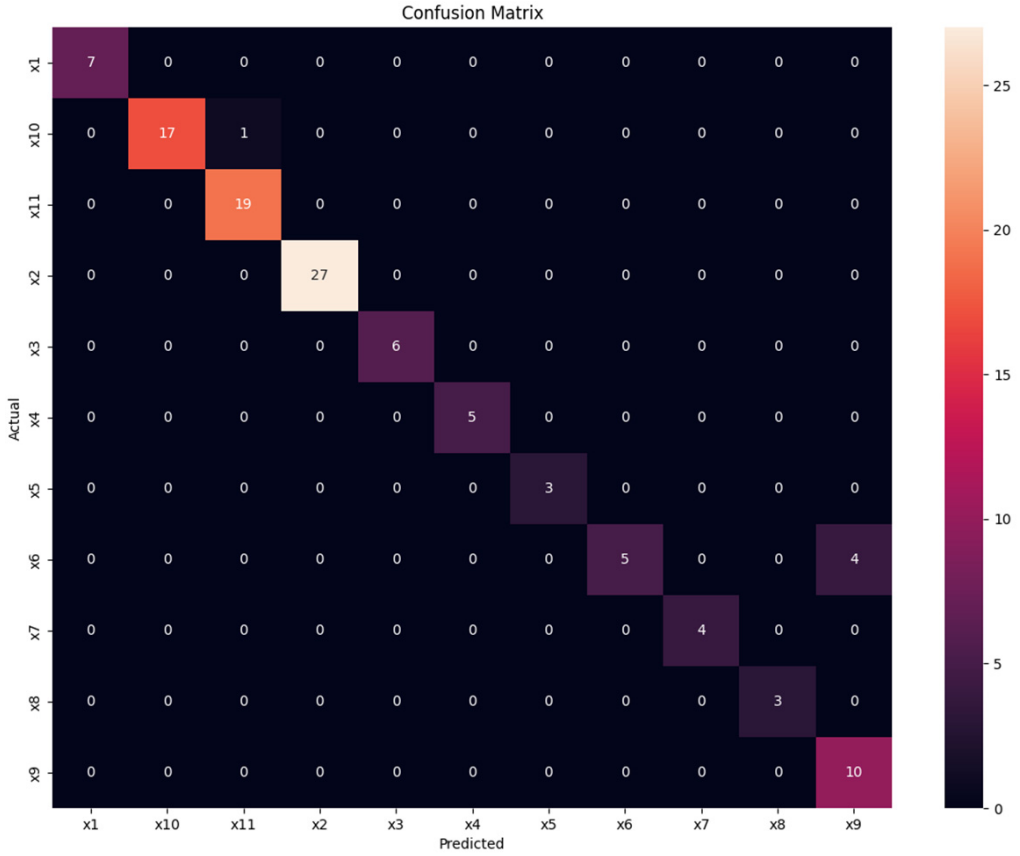


Figure 10. Fixed-text confusion matrix with 11 user classes.

```
------------------------------------------------------------------------
Training for fold 5 ...
------------------------------------------------------------------------
Score per fold
------------------------------------------------------------------------
> Fold 1 - Train loss: 0.9459459185600281 - Validation loss: 0.9054054021835327
------------------------------------------------------------------------
> Fold 2 - Train loss: 0.9425675868988037 - Validation loss: 0.8783783912658691
------------------------------------------------------------------------
> Fold 3 - Train loss: 0.9898648858070374 - Validation loss: 0.9864864945411682
------------------------------------------------------------------------
> Fold 4 - Train loss: 0.9527027010917664 - Validation loss: 0.9459459185600281
------------------------------------------------------------------------
> Fold 5 - Train loss: 0.9695945978164673 - Validation loss: 0.9594594836235046
------------------------------------------------------------------------
Average scores for all folds:
> Accuracy: 0.9601351380348205 (+- 0.01754156893264166)
> Validation Accuracy: 0.9351351380348205 (+- 0.038602316190887255)
------------------------------------------------------------------------
```

Figure 11. K-fold cross-validation of fixed-text dataset.

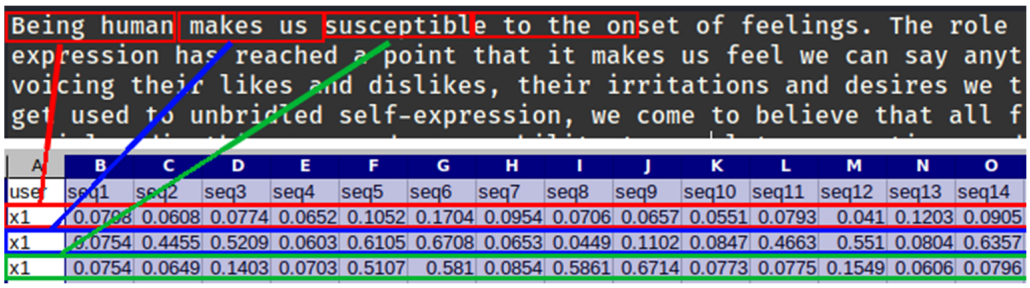
4.2. Free-Text Keystroke Dynamics

The free-text keystroke dynamics systems require significantly more data to train, varying from specified text entries to typing different paragraphs. That means that H, UD, and DD times had to be calculated for each key value.

One approach involves having users type the same text or paragraphs, while another approach involves having each participant type different paragraphs, which closely resembles free-text keystroke dynamics. A good example of this is a dataset collected by two researchers that comprises a comprehensive collection of approximately 136 million keystrokes derived from a diverse range of users and devices [68]. In our research, this dataset was utilized for experiments; however, the data collected by these researchers

specifically consisted of the operating system's time values for keypress and release events. Consequently, the calculation of the hold (H), up–down (UD), and down–down (DD) times was necessary for each key value. The test results obtained using our proposed method were inaccurate and unsuitable for the generation due to the inherent limitations of the data-collection process, including variations in devices, keyboard layouts, and limited collection of typing features. Therefore, the subsequent sections of this research present the findings obtained from a dataset that was specifically collected by us, utilizing the "spy_k3y" and keyboard implant, solely for the purpose of this research.

For the purposes of this research on free-text analysis, a unique dataset was collected by soliciting individuals to type a minimum of two paragraphs. Table 1 denotes the size of the dataset and the number of individuals from whom it was gathered. One paragraph was identical among all participants, while the remaining paragraphs varied individually. The texts utilized for the collection of datasets were sourced from a typing-practice platform (https://thepracticetest.com (accessed on 3 April 2023)) to ensure maximal diversity and an authentic representation of the characteristics of the user's text typing. All samples used in the free-text research were no longer than 700 characters and the average word length was about 5.6 words. All the paragraphs contained most letters from the English alphabet. The average word length was important for us since we decided to use 30 features (see Figure 12) as an input for our LSTM model because we noticed that participants typing given paragraphs tend to type about 10 symbols before looking at the text again. Data collected in free-text dataset are split into chunks of 30 data entries each (stream of mix of H, UD, and DD values). Each element in that chunk is represented in seq from 1 to 30 and a new line in the dataset (which was used for LSTM) is a new chunk of seq from 1 to 30. Figure 12 illustrates one of the keystroke resolutions employed in the research, specifically utilizing four decimal places. Additionally, another resolution of eight decimal places was employed to accommodate a larger number of users. The choice of different resolutions allowed flexibility in capturing and analyzing keystroke dynamics.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	user	seq1	seq2	seq3	seq4	seq5	seq6	seq7	seq8	seq9	seq10	seq11	seq12	seq13	seq14
x1		0.0708	0.0608	0.0774	0.0652	0.1052	0.1704	0.0954	0.0706	0.0657	0.0551	0.0793	0.041	0.1203	0.0905
x1		0.0754	0.4455	0.5209	0.0603	0.6105	0.6708	0.0653	0.0449	0.1102	0.0847	0.4663	0.551	0.0804	0.6357
x1		0.0754	0.0649	0.1403	0.0703	0.5107	0.581	0.0854	0.5861	0.6714	0.0773	0.0775	0.1549	0.0606	0.0796

Figure 12. Features and data chunking in free-text analysis.

The LSTM employed in the free-text research was configured with three layers and a single dense layer. The initial layer consisted of 512 neurons, while the second and third layers contained 128 neurons each. Activation was performed using the Softmax function. The Adam optimizer was used for the training of the model. Experimental trials were conducted, and, when employing five layers, the number of neurons doubled. However, incorporating such a solution into a computationally limited device presents considerable challenges, particularly when prioritizing appropriate payload generation. Consequently, the specified configuration was chosen to seek a balance between speed and results.

The model's performance, in terms of accuracy and loss, exhibited a comparatively inferior performance of approximately 10–20% when compared to free-text experiments. Validation stood out in particular, since the number of data for free-text was very small. The train–test split employed for free-text experiments followed a distribution of 70% for training and 30% for testing. Most tests incorporated 200 epochs. The precision and recall values varied widely across experiments, particularly when dealing with short texts or a limited number of blocks, ranging between 0.95 and 0.71. Notably, as the number of blocks increased, the results decreased even more. The confusion matrix for the free-text

analysis can be observed in Figure 13. The average accuracy on five folds varied from 74% to 88%. To achieve sufficient results aligned with the accuracy required for free-text analysis, approximately 10 users were required. The outcomes obtained from the tests conducted on the mixed dataset specified in Table 1 were only acceptable when dealing with a limited number of blocks, typically limited to the size of several passwords. For longer texts, the approaches used in the research were ineffective or could not be used to bypass keystroke dynamics systems.

Figures 14 and 15 illustrate the outcomes of payload generation pertaining to free-text experiments, specifically in relation to prediction and assigning payload blocks to user x1. The generated payloads were produced subsequent to the submission of user x1's captured keystrokes to the inject tool, with the specification of the desired payload type based on the logged key values or the entered text. The generation of the depicted payload involves utilizing one of the texts from the dataset collection phase, which includes 26 blocks, as well as a randomly selected text. Subsequently, the objective of deep-learning is to identify the imitated ownership of the blocks within the generated payload. In the case depicted in Figure 14, eight blocks were used, which could be represented as several relatively lengthy passwords. The prediction results in this scenario were good. Further experiments involving predictions with up to 10 blocks yielded results ranging from 75% to 92%.

Figure 15 shows the prediction results for 26 blocks or features. In particular, when the LSTM model attempted to predict with this particular number of blocks, the accuracy decreased, averaging at 74%. The comparison of all generated payloads was conducted among 16 users. Following thorough training of the network, the predictions consistently exhibited no more than two falsely identified users, who were included among the actual users whose data were utilized to generate the payload. During the payload-generation process, the model demonstrated success in payload prediction even when not all required data were available. In such cases, the model leveraged the data from the closest user when generating the payload, provided that it corresponded to a specific block or generated it according to the predefined values. This approach can be particularly effective when the number of users is limited and the accuracy of the keystroke dynamics system is average or lower.

The payload generation tool performed effectively without requiring any modifications both in the context of free-text experiments and its original design for fixed-text payload generation and verification tasks. The experiments on payload prediction demonstrated that the "snipe_inject" tool not only generated payloads based on the keylogged data but also successfully generated payloads based on specified text provided prior to the payload generation process.

Analysis of collected data also revealed that users are more focused at the beginning of the paragraph and tend to be more distracted over time and regain focus during the end. This observation becomes particularly evident when conducting analysis of data derived from the same paragraph that was typed by multiple users.

To evaluate the efficiency and precision of the payload generated from the collected free-text data we developed POCs for two distinct attacks, unlocking a screen-locked computer and executing a PowerShell script.

By leveraging keystroke dynamics to impersonate a user, a threat actor can execute a PowerShell script or bypass the login screen by utilizing a malicious USB device. The first technique involved mimicking the typing dynamics of the target user, allowing the threat actor to input specific commands into the PowerShell prompt using hidden window mode without alerting the user or raising suspicions, especially if user keystroke dynamics solutions were invoked. Figure 16 shows a snippet of the payload script with keystroke timings. The provided example demonstrates a static payload configuration that triggers a predefined keystroke. However, the same principle can be applied to create a user-specific payload capable of injecting malicious data while the user is typing. This approach is regarded as on-demand payload generation, where the keypress timings are dynamically

altered to introduce variations. Figure 8 showcases the generation of multiple payloads for the same user, highlighting the continuously changing keypress timings.

The lock-screen bypass attack involves connecting the keyboard with an implant or malicious USB device to the target system, which tricks the computer into recognizing it as a trusted input device. By emulating the user's keystroke patterns, the threat actor can bypass the login-screen lock, gaining unauthorized access. In our case, the keyboard with implant had the full functionality for such an attack, including managing the attack remotely. If the user would use on-screen keyboard, the camera planted in our keyboard could help to exploit this as well.

To see whether our collected data resemble the typing characteristics of distinct users, we compared two users to see how different they appear by graphing their typing values during the typing period. Their results are provided in Figure 17.

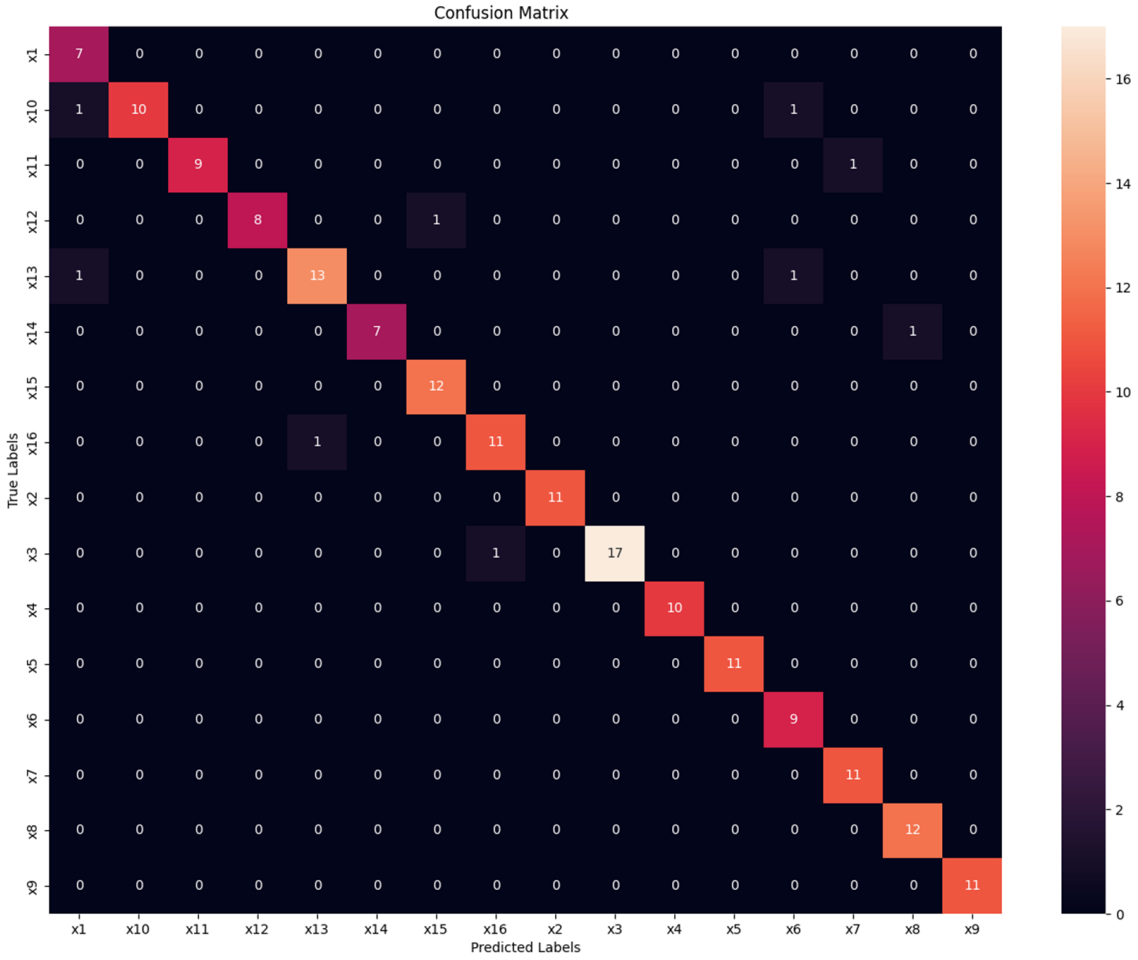


Figure 13. Free-text confusion matrix with X user classes.

Predicted Usernames:

	Predicted Usernames	Prediction Percentage
0	x1	0.608519
1	x1	0.926162
2	x1	0.921622
3	x1	0.915620
4	x1	0.639046
5	x1	0.723356
6	x1	0.550013
7	x1	0.746314

Statistical Information:

	Prediction Percentage
count	8.000000
mean	0.753832
std	0.151585
min	0.550013
25%	0.631415
50%	0.734835
75%	0.917121
max	0.926162

Figure 14. Small generated payload for x1 user prediction.

	Predicted Usernames	Prediction
0	x8	0.612084
1	x1	0.855823
2	x8	0.540779
3	x1	0.925977
4	x1	0.797307
5	x1	0.857802
6	x1	0.894726
7	x8	0.564262
8	x8	0.851242
9	x1	0.744720
10	x1	0.852996
11	x1	0.754527
12	x8	0.671141
13	x8	0.828406
14	x1	0.795405
15	x1	0.745525
16	x1	0.787003
17	x1	0.710669
18	x1	0.595794
19	x1	0.922042
20	x1	0.598022
21	x1	0.825578
22	x1	0.649179
23	x8	0.512084
24	x1	0.751080
25	x1	0.726593

Statistical Information:

	Prediction
count	26.000000
mean	0.745030
std	0.120233
min	0.512084
25%	0.654669
50%	0.752804
75%	0.845533
max	0.925977

Figure 15. Generated average payload for x1 user prediction.

```
port = serial.Serial("/dev/serial0", 115200)
port.write(b'\16')
time.sleep(0.00439709)
port.write(b'\0')
time.sleep(0.22020047)
port.write(("p").encode("utf-8"))
time.sleep(0.19973716)
port.write(b'\0')
time.sleep(0.17199515)
port.write(("o").encode("utf-8"))
time.sleep(0.08080615)
port.write(b'\0')
time.sleep(0.21599415)
port.write(("w").encode("utf-8"))
```

Figure 16. Payload snippet of payload impersonating user keystrokes.

From Figure 17, we can see that User 2 was typing faster and was less distracted during the test. To achieve deeper conclusions on typing characteristics of these two participants, we differentiated and compared three features (see Figure 18)—H.time (blue), U.D.time (red), and D.D.time (yellow). These features are also used for our LSTM model and by many others. The UD and DD times show us how fast a person can type words, as they reveal how long someone searches for a letter. And H time shows us how strongly an individual presses a single key (longer time—more force for a press is used).

We can see that most U.D. and D.D. values of User 2 are in the 2 s range. And the H. times are very low. Most of User 2 U.D. and D.D. values are in the range of 1.5 s. This means that User 2 probably types faster. However, he uses more force to press a single key and we can see that in higher H. time values.

Since most participants tend to be less distracted during the beginning, we compared the first 10 characters typed (30 timings of keystroke data) of all users from our dataset and see if we can observe a noticeable difference (see Figure 19). Each line of a different color in Figure 19 represents an individual user and his keystroke dynamics, expressed in seconds.

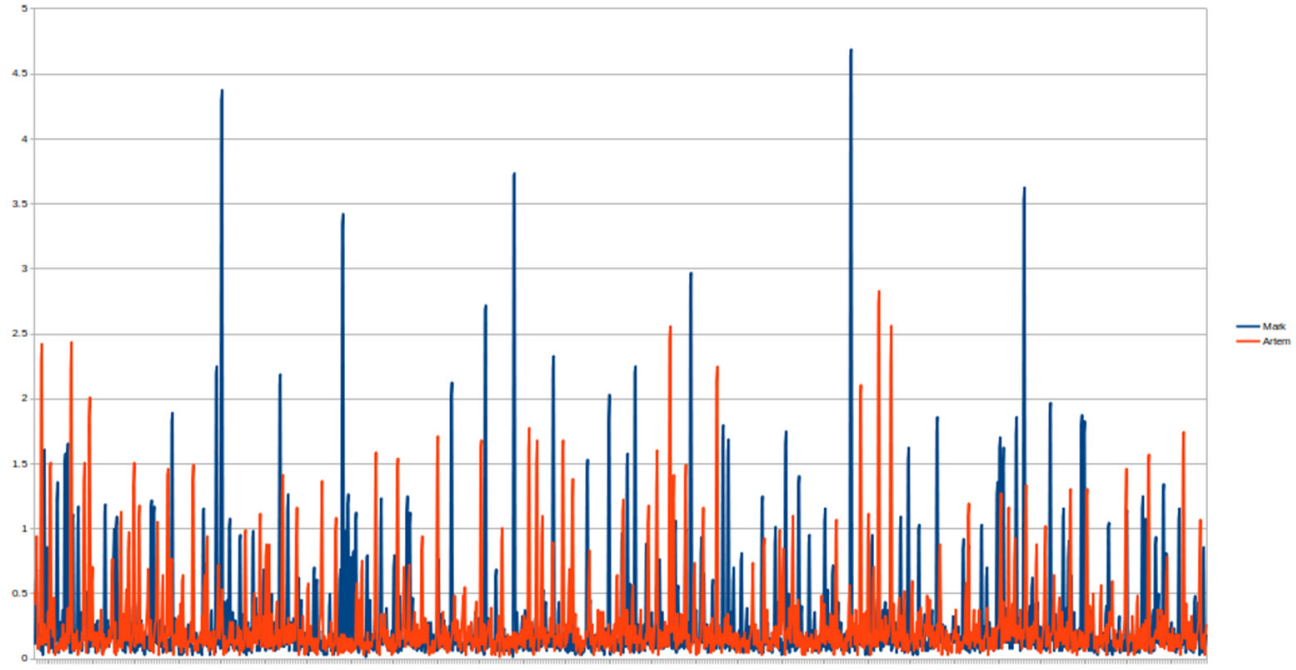


Figure 17. Comparison of the typing data of User 1 (blue) and User 2 (red) (the same paragraph).

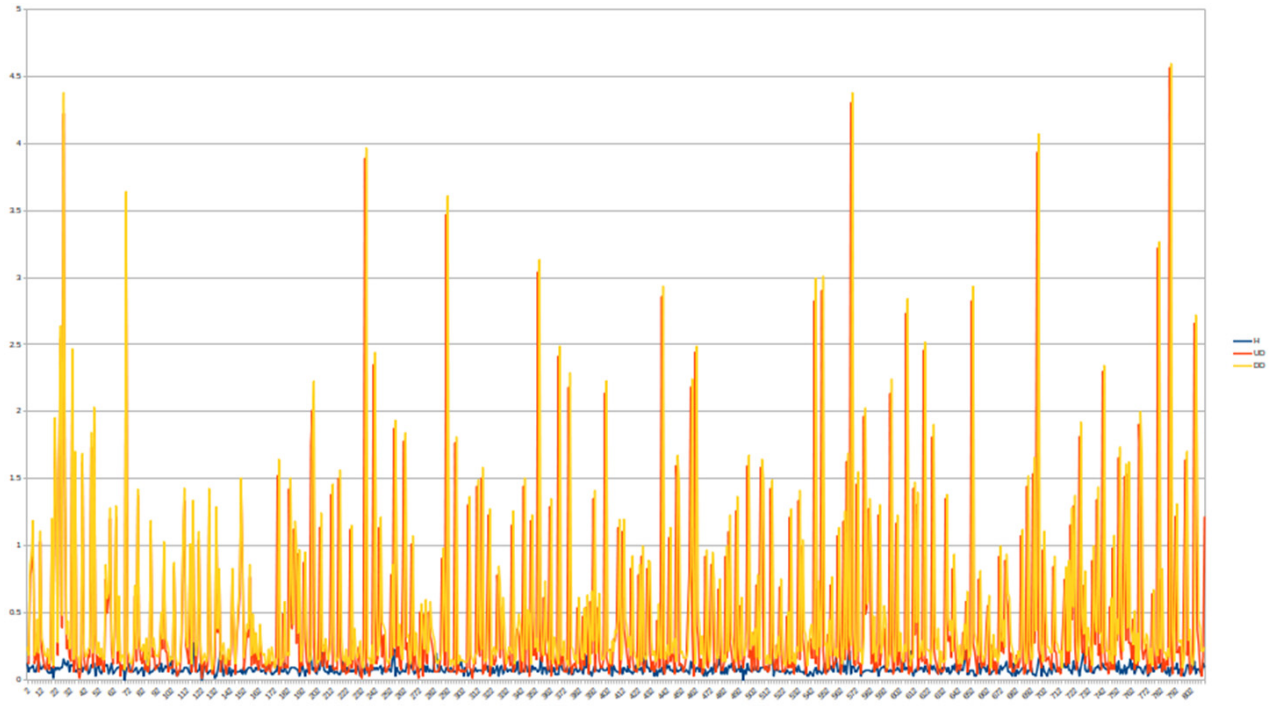


Figure 18. H. time, U.D.time, and D.D.time characteristics of the research.

Observations from the experiment revealed a higher occurrence of anomalies in the generated payload compared to the original input, which shows that in high-precision systems it can be detected if the anomaly in the generated payload occurs more often. We analyzed the reason behind these results and noticed that users that have fewer anomalies in their datasets tend to "poison" the payload generation with rear accidental anomalies that appear unpredicted and do not repeat. This would increase the interval when random values are generated because during the payload generation process we search for specific values that are needed inside that dataset and we generate a random value inside the

interval of min and max. To improve this method, anomaly data can be collected and used to develop a mechanism to control how many anomalies are present in user data and take that into consideration when the payload is generated.

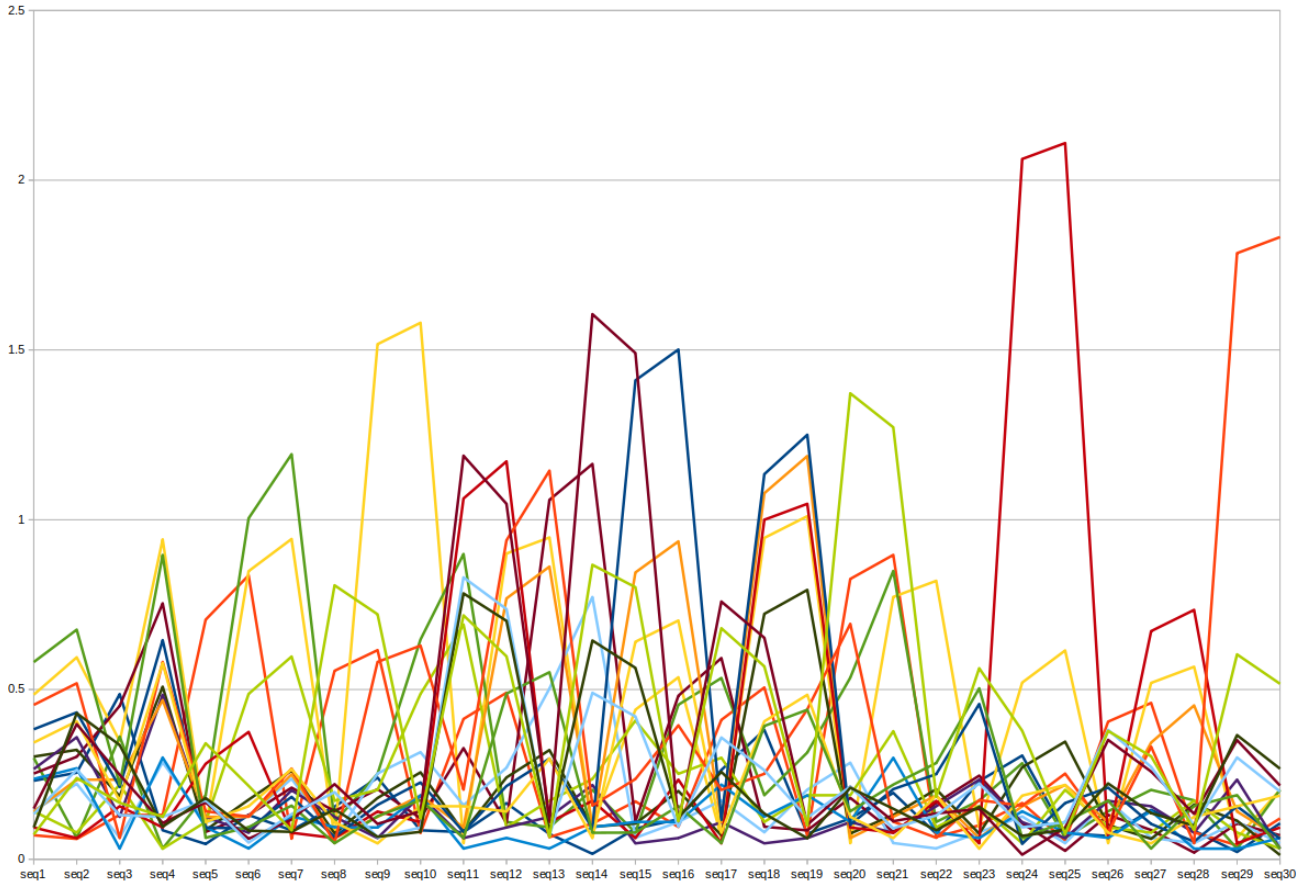


Figure 19. Difference in typing speed between users.

4.3. Comparison with the Previous Results Obtained

Direct comparison of our results with studies conducted by other researchers poses challenges due to divergent research focuses. Some researchers have prioritized hardware implementation for keystroke-injection attacks [24], while others have focused on software implementation [47]. Notably, in terms of user-classification accuracy with free-text, their results ranged from 94% to 97%, closely aligning with the outcomes of our own research. The findings of this investigation indicate that greater computational power facilitates the attainment of more precise results, particularly when the quantity of data from the impersonated user is known. However, in our proposed approach, we do not presuppose prior knowledge of the data provided by the targeted user. This assumption stems from the notion that this attack method can be employed in scenarios where fundamental information about the user of the compromised device is not accessible.

When comparing our proposed model with the research findings of the "USB keyboard attack case study" [27], we observe a very simplified version compared to our proposed model. The primary advantage of utilizing a microcontroller, as opposed to our proposed solution, lies in its low-power consumption. However, it is important to note that the microcontroller alone lacks sufficient power to collect data and execute attacks autonomously without the aid of external devices. Furthermore, the attack scenario presented in the aforementioned case study necessitates physical access to the target device, specifically during the data-collection phase after a certain period of time. Scheduling an attack at a specific time entails several drawbacks, including the potential for the system to be powered off during the scheduled attack, the user's active utilization of the device at the

designated time, and the possibility of detecting the attack. On the contrary, our proposed model enables the launch of attacks based on real-time circumstances. The success of an attack is dependent upon the prevailing conditions rather than upon adhering to a predetermined time.

Among the existing solutions, the solution that closely resembles our proposed model is "Malboard: A novel user keystroke impersonation attack" [14]. This study presents three adaptations of side-channel attacks for detecting keystroke-attack devices, which aligns well with the functionality of our proposed model. Compared to our proposed model, there are several key distinctions. First, our model focuses on launching attacks on demand, allowing for targeted and specific actions. Additionally, we incorporate payload verification mechanisms to ensure compatibility and prevent mismatches. Moreover, our model employs sophisticated averaging techniques to handle unknown values during payload generation when the dataset size is very limited. Furthermore, we utilize camera-based methods to bypass keyboard checks and gain a user's perspective on the controlled device, enhancing the overall attack capabilities.

5. Conclusions

The proposed keystroke-injection method implements a covert hardware keystroke-injection platform that collects keystroke dynamics data (H.time, UD.time, and DD.time + keystrokes), generates payload, and transmits that data via serial communication to be injected with the same value as a user (live keypress and up events).

The fixed-text dataset contained 11 users who each typed a fixed pass phrase multiple times. Graphed data showed noticeable differences in typing speeds between these users, especially in UD.time and DD.time.

The free-text dataset contained data from 20 users typing the same paragraph and some participants also typed a different paragraph. All paragraphs were taken from a typing-practice website and contained various symbols and letters. The difference in typing speeds was also noticeable by each user, however, during the duration of the test all users showed anomalies in their typing habits (caused by lack of focus, emotions, or overall tiredness, etc.). Some users had more anomalies than others. These findings offer valuable insight into distinguishing between human users and robots, enabling identification of impostors.

Experiments were performed with collected user data, as well as TypingDNA web 2FA. The proposed keystroke-injection method was able to bypass the fixed-text keystroke dynamics solution by TypingDNA and login to the user account. LSTM was trained on the collected user dataset with a passphrase (91% validation accuracy was achieved), and 10 payloads were given as unlabeled data to be verified. Nine times out of 10, the payload was identified as the victim user. Victim data were compared graphically to a payload and two random users and a noticeable similarity between payload and user could be seen, which indicates that our payload is effective.

The free-text keystroke dynamics dataset was also used to train the LSTM model, although 74% validation accuracy was achieved. A dataset of 136 million keystrokes [68] was also tested, however, experiment results revealed that data did not meet the requirements to generate payload with the required level of accuracy. Generated free-text payload prediction accuracy results ranged from 75% to 92%. However, similar graphical comparison was drawn between a payload and victim, and the payload showed similar similarities to a user as in fixed-text, but an important difference was noticed. Users who had fewer anomalies in their data (made fewer mistakes while typing and were more focused overall) tend to display more noticeable difference from the generated payload.

During the research, it was observed that in order to achieve the desired accuracy for bypassing security systems, it is important to record the timings of keystrokes along with the corresponding key values. This is because the analysis of the collected data revealed that users who natively use various keyboard layouts have distinct ways of entering special characters or capital letters. For CLICKA researchers, attempts to recognize the native language of users based on keystroke dynamics allowed them to achieve only slightly

higher accuracy than guessing [70]. With keylogging and mapping keys to appropriate behavior, the accuracy of such research would increase significantly. In keystroke dynamics systems, these anomalies can serve as significant features with substantial importance. The absence of specific keypress data can notably diminish the accuracy during payload generation. Overall research showed that the "snipe_inject" payload is effective in free-text keystroke dynamics as well.

6. Limitations and Future Research

At its current stage, the proposed model does have some limitations and shortcomings. One such limitation is that dataset collection and payload generation can be achieved with a relatively small number of users as the device operates locally. However, this limitation can also be viewed as a significant advantage of the proposed model, particularly when the targeting of specific user populations becomes crucial.

The research did not specifically examine the impact of hardware and software solutions on the accuracy of collected keypress times and the results obtained by using fixed-time values. However, it is worth noting that the precision employed in capturing keypresses during certain experiments exceeded that of other researches, utilizing a time resolution of eight decimal places.

The plug-and-play applicability of the proposed implant to all keyboards without the need for additional reprogramming has not been thoroughly studied.

The results obtained using free-text user data from other researchers' datasets did not yield the desired level of accuracy. To better understand the reasons behind these moderate results, more research is needed. Additionally, it is important to test the proposed model against advanced keystroke-injection detection methods, including those that involve faked keypress data for deception purposes. The integration of a microphone would also enhance data-capture capabilities, complementing the existing camera functionality.

Author Contributions: Conceptualization, V.G. and J.D.; methodology, N.G.; software, J.D.; validation, J.J. and A.Č.; formal analysis, N.G.; investigation, J.D. and V.G.; resources, A.Č.; data curation, J.D. and V.G.; writing—original draft preparation, J.D. and J.J.; writing—review and editing, N.G.; visualization, J.D.; supervision, N.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Publicly available datasets were analyzed in this study. These data can be found here: [https://github.com/itsecprof/keystroke-payload (accessed on 5 June 2023)].

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Tian, J.; Scaife, N.; Kumar, D.; Bailey, M.; Bates, A.; Butler, K. SoK: 'Plug & Pray' Today-Understanding USB Insecurity in Versions 1 Through C. In Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 20–24 May 2018; pp. 1032–1047. [CrossRef]
2. Lu, H.; Wu, Y.; Li, S.; Lin, Y.; Zhang, C.; Zhang, F. BADUSB-C: Revisiting BadUSB with Type-C. In Proceedings of the 2021 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 27 May 2021; pp. 327–338. [CrossRef]
3. Thomas, T.; Piscitelli, M.; Nahar, B.A.; Baggili, I. Duck Hunt: Memory forensics of USB attack platforms. *Forensic Sci. Int. Digit. Investig.* **2021**, *37*, 301190. [CrossRef]
4. Mohammadmoradi, H.; Gnawali, O. Making whitelisting-based defense work against bad USB. In Proceedings of the 2nd International Conference on Smart Digital Environment, ICSDE'18, Rabat, Morocco, 18–20 October 2018; ACM International Conference Proceeding Series; ACM: New York, NY, USA, 2018; pp. 127–134. [CrossRef]
5. Liu, H.; Spolaor, R.; Turrin, F.; Bonafede, R.; Conti, M. USB powered devices: A survey of side-channel threats and countermeasures. *High Confid. Comput.* **2021**, *1*, 100007. [CrossRef]
6. Dieter, G. *Computer Security*, 3rd ed.; John Wiley & Sons: Hoboken, NJ, USA, 2011.
7. Karantzas, G. Forensic Log Based Detection for Keystroke Injection 'BadUsb' Attacks. *arXiv* **2023**, arXiv:2302.04541.
8. Lawal, D.; Gresty, D.; Gan, D.; Hewitt, L. Have You Been Framed and Can You Prove It? In Proceedings of the 2021 44th International Convention on Information, Communication and Electronic Technology, MIPRO, Opatija, Croatia, 27 September–1 October 2021; pp. 1236–1241. [CrossRef]

9. Dumitru, R.; Wabnitz, A.; Genkin, D.; Yarom, Y. The Impostor Among US(B): Off-Path Injection Attacks on USB Communications. *arXiv* **2022**, arXiv:2211.01109.
10. Nissim, N.; Yahalom, R.; Elovici, Y. USB-based attacks. *Comput. Secur.* **2017**, *70*, 675–688. [CrossRef]
11. Arora, L.; Thakur, N.; Yadav, S.K. USB rubber ducky detection by using heuristic rules. In Proceedings of the IEEE 2021 International Conference on Computing, Communication, and Intelligent Systems, ICCCIS, Greater Noida, India, 19–20 February 2021; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2021; pp. 156–160. [CrossRef]
12. Mamchenko, M.; Sabanov, A. Exploring the taxonomy of USB-based attacks. In Proceedings of the 2019 12th International Conference "Management of Large-Scale System Development" (MLSD), Moscow, Russia, 1–3 October 2019; pp. 1–4. [CrossRef]
13. Lee, K.; Yim, K. Vulnerability Analysis and Security Assessment of Secure Keyboard Software to Prevent PS/2 Interface Keyboard Sniffing. *Sensors* **2023**, *23*, 3501. [CrossRef]
14. Farhi, N.; Nissim, N.; Elovici, Y. Malboard: A novel user keystroke impersonation attack and trusted detection framework based on side-channel analysis. *Comput. Secur.* **2019**, *85*, 240–269. [CrossRef]
15. Ramadhanty, A.D.; Budiono, A.; Almaarif, A. Implementation and Analysis of Keyboard Injection Attack using USB Devices in Windows Operating System. In Proceedings of the 2020 3rd International Conference on Computer and Informatics Engineering, IC2IE, Yogyakarta, Indonesia, 15–16 September 2020; pp. 449–454. [CrossRef]
16. Negi, A.; Rathore, S.S.; Sadhya, D. USB Keypress Injection Attack Detection via Free-Text Keystroke Dynamics. In Proceedings of the 2021 8th International Conference on Signal Processing and Integrated Networks (SPIN), Noida, India, 26–27 August 2021; Institute of Electrical and Electronics Engineers (IEEE): Piscataway, NJ, USA, 2021; pp. 681–685. [CrossRef]
17. Borges, C.D.B.; de Araujo, J.R.B.; de Couto, R.L.; Almeida, A.M.A. Keyblock: A software architecture to prevent keystroke injection attacks. In Proceedings of the XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, Brasilia, Brazil, 6–9 November 2017; pp. 518–524. [CrossRef]
18. Tian, D.J.; Bates, A.; Butler, K. Defending against malicious USB firmware with GoodUSB. In Proceedings of the 31st Annual Computer Security Applications Conference, ACSAC '15, Los Angeles, CA, USA, 7–11 December 2015; ACM: New York, NY, USA, 2015; pp. 261–270. [CrossRef]
19. Wahanani, H.; Idhom, M.; Kurniawan, D.R. Exploit remote attack test in operating system using arduino micro. *J. Phys. Conf. Ser.* **2020**, *1569*, 022038. [CrossRef]
20. Clements, A. *Principles of Computer Hardware*, 4th ed.; Oxford University Press: Oxford, UK, 2006.
21. Faircloth, J. Client-side attacks and social engineering. In *Penetration Tester's Open Source Toolkit*; Elsevier: Amsterdam, The Netherlands, 2017. [CrossRef]
22. Sun, C.; Lu, J.; Liu, Y. Analysis and Prevention of Information Security of USB. In Proceedings of the 2021 International Conference on Electronic Information Engineering and Computer Science, EIECS, Changchun, China, 23–26 September 2021; pp. 25–32. [CrossRef]
23. Cronin, P.; Gao, X.; Wang, H.; Cotton, C. Time-Print: Authenticating USB Flash Drives with Novel Timing Fingerprints. In Proceedings of the 2022 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 22–26 May 2022; pp. 1002–1017. [CrossRef]
24. Eswar, P.V.D.S. Microcontroller Manipulated As Human Interface Device Performing Keystroke Injection Attack. *Int. Res. J. Mod. Eng. Technol. Sci.* **2021**, *3*, 1230–1233.
25. Muslim, A.A.; Budiono, A.; Almaarif, A. Implementation and Analysis of USB based Password Stealer using PowerShell in Google Chrome and Mozilla Firefox. In Proceedings of the 2020 3rd International Conference on Computer and Informatics Engineering, IC2IE, Yogyakarta, Indonesia, 15–16 September 2020; pp. 421–426. [CrossRef]
26. Ferreira, J.L.S.; Amorim, M.F.; Altafim, R.A.P. Biometric patterns recognition using keystroke dynamics. In Proceedings of the XVIII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais, Natal, Brazil, 22–25 October 2018.
27. Bojović, P.D.; Bojović, P.D.; Bašičević, I.; Pilipović, M.; Bojović, Ž.; Bojović, M. The Rising Threat of Hardware Attacks: USB Keyboard Attack Case Study. *IEEE Secur. Priv.* **2020**, *preprint*. Available online: https://www.researchgate.net/publication/359509222 (accessed on 19 June 2023).
28. Ahire, J.; Shembekar, A.; Makadia, K.; Bhokare, D. Exploring Attack Vectors Using Single Board Computers. *Int. Res. J. Mod. Eng. Technol. Sci.* **2022**, *4*, 2911–2914.
29. Nicho, M.; Sabry, I. Threat and Vulnerability Modelling of Malicious Human Interface Devices. *Technol. Eng. Math. (EPSTEM)* **2022**, *21*, 241–247. Available online: www.isres.org (accessed on 19 May 2023). [CrossRef]
30. Neuner, S.; Voyiatzis, A.G.; Fotopoulos, S.; Mulliner, C.; Weippl, E.R. Usblock: Blocking USB-Based keypress injection attacks. In *Data and Applications Security and Privacy XXXII*; LNCS; Springer International Publishing: Cham, Switzerland, 2018; Volume 10980. [CrossRef]
31. Kang, M.; Saiedian, H. USBWall: A novel security mechanism to protect against maliciously reprogrammed USB devices. *Inf. Secur. J.* **2017**, *26*, 166–185. [CrossRef]
32. Wang, Z. Poisoning Attacks on Learning-Based Keystroke Authentication Poisoning Attacks on Learning-Based Keystroke Authentication and a Residue Feature Based Defense and a Residue Feature Based Defense. Available online: https://digitalcommons.latech.edu/dissertations (accessed on 19 May 2023).
33. Szoke, D. Model Poisoning in Federated Learning: Collusive and Individual Attacks. Ph.D. Thesis, The Ohio State University, Columbus, OH, USA, 2023.

34. Porwik, P.; Doroz, R.; Wesolowski, T.E. Dynamic keystroke pattern analysis and classifiers with competence for user recognition. *Appl. Soft Comput.* **2021**, *99*, 106902. [CrossRef]
35. Hazan, I.; Margalit, O.; Rokach, L. Supporting unknown number of users in keystroke dynamics models. *Knowl. Based Syst.* **2021**, *221*, 106982. [CrossRef]
36. Lu, X.; Zhang, S.; Hui, P.; Lio, P. Continuous authentication by free-text keystroke based on CNN and RNN. *Comput. Secur.* **2020**, *96*, 101861. [CrossRef]
37. Roy, S.; Roy, U.; Sinha, D.; Pal, R.K. Imbalanced ensemble learning in determining Parkinson's disease using Keystroke dynamics. *Expert Syst. Appl.* **2022**, *217*, 119522. [CrossRef]
38. Chang, H.-C.; Li, J.; Wu, C.-S.; Stamp, M. Machine Learning and Deep Learning for Fixed-Text Keystroke Dynamics. *arXiv* **2021**, arXiv:2107.00507.
39. Ibrahim, M.; Abdelraouf, H.; Amin, K.M.; Semary, N. Keystroke dynamics based user authentication using Histogram Gradient Boosting. *Int. J. Comput. Inf. IJCI* **2023**, *10*, 36–53. [CrossRef]
40. Nnamoko, N.; Barrowclough, J.; Liptrott, M.; Korkontzelos, I. A behaviour biometrics dataset for user identification and authentication. *Data Brief* **2022**, *45*, 108728. [CrossRef]
41. Parkinson, S.; Khan, S.; Crampton, A.; Xu, Q.; Xie, W.; Liu, N.; Dakin, K. Password policy characteristics and keystroke biometric authentication. *IET Biom.* **2021**, *10*, 163–178. [CrossRef]
42. Zeid, E.S.S.; ElKamar, R.A.; Hassan, S.I. Fixed-Text vs. Free-Text Keystroke Dynamics for User Authentication. *Eng. Res. J. Fac. Eng.* **2022**, *51*, 95–104.
43. Mondal, S.; Bours, P. A study on continuous authentication using a combination of keystroke and mouse biometrics. *Neurocomputing* **2016**, *230*, 1–22. [CrossRef]
44. Ciaramella, G.; Iadarola, G.; Martinelli, F.; Mercaldo, F.; Santone, A. Continuous and Silent User Authentication Through Mouse Dynamics and Explainable Deep Learning: A Proposal. In Proceedings of the 2022 IEEE International Conference on Big Data, (Big Data 2022), Osaka, Japan, 17–20 December 2022; pp. 6628–6630. [CrossRef]
45. Shadman, R.; Wahab, A.A.; Manno, M.; Lukaszewski, M.; Hou, D.; Hussain, F. Keystroke Dynamics: Concepts, Techniques, and Applications. *arXiv* **2023**, arXiv:2303.04605.
46. Iapa, A.C.; Cretu, V.I. Modified Distance Metric That Generates Better Performance for the Authentication Algorithm Based on Free-Text Keystroke Dynamics. In Proceedings of the SACI 2021—IEEE 15th International Symposium on Applied Computational Intelligence and Informatics, Timisoara, Romania, 19–21 May 2021; pp. 455–460. [CrossRef]
47. Eizaguirre-Peral, I.; Segurola-Gil, L.; Zola, F. Conditional Generative Adversarial Network for keystroke presentation attack. *arXiv* **2022**, arXiv:2212.08445.
48. Kochegurova, E.A.; Zateev, R.P. Hidden Monitoring Based on Keystroke Dynamics in Online Examination System. *Program. Comput. Softw.* **2022**, *48*, 385–398. [CrossRef]
49. Bernatavičienė, J. Proceedings of the 13th Conference on "Data analysis methods for software systems". *Vilnius Univ. Proc.* **2022**, *31*, 1–110. [CrossRef]
50. Eizagirre, I.; Segurola, L.; Zola, F.; Orduna, R. Keystroke Presentation Attack: Generative Adversarial Networks for Replacing User Behaviour. In Proceedings of the 2022 European Symposium on Software Engineering, ESSE '22, Rome, Italy, 27–29 October 2022; Association for Computing Machinery: New York, NY, USA, 2023; pp. 119–126. [CrossRef]
51. Wahab, A.; Hou, D. When Simple Statistical Algorithms Outperform Deep Learning: A Case of Keystroke Dynamics. In Proceedings of the 12th International Conference on Pattern Recognition Applications and Methods ICPRAM, Lisbon, Portugal, 22–24 February 2023; pp. 363–370. [CrossRef]
52. Kar, S.; Bamotra, A.; Duvvuri, B.; Mohanan, R. KeyDetect—Detection of anomalies and user based on Keystroke Dynamics. *arXiv* **2023**, arXiv:2304.03958.
53. Tewani, A. Keystroke Dynamics based Recognition Systems using Deep Keystroke Dynamics based Recognition Systems using Deep Learning: A Survey Learning: A Survey. *techRxiv* **2022**, *preprint*. [CrossRef]
54. Toosi, R.; Akhaee, M.A. Time–frequency analysis of keystroke dynamics for user authentication. *Future Gener. Comput. Syst.* **2021**, *115*, 438–447. [CrossRef]
55. Killourhy, K.S.; Maxion, R.A. Comparing anomaly-detection algorithms for keystroke dynamics. In Proceedings of the International Conference on Dependable Systems and Networks, Lisbon, Portugal, 29 June–2 July 2009; pp. 125–134. [CrossRef]
56. Killourhy, K.S.; Maxion, R.A. Free vs. transcribed text for keystroke-dynamics evaluations. In LASER '12: Proceedings of the 2012 Workshop on Learning from Authoritative Security Experiment Results, Arlington, VA, USA, 18–19 July 2012; ACM International Conference Proceeding Series; Association for Computing Machinery: New York, NY, USA, 2012; pp. 1–8. [CrossRef]
57. González, N.; Calot, E.P. Finite context modeling of keystroke dynamics in free text. In Proceedings of the 2015 International Conference of the Biometrics Special Interest Group (BIOSIG), Darmstadt, Germany, 20–22 September 2015; Lecture Notes in Informatics (LNI), Proceedings-Series of the Gesellschaft fur Informatik (GI); Gesellschaft fur Informatik: Bonn, Germany, 2015; Volume P-245. [CrossRef]
58. Banerjee, R.; Feng, S.; Kang, J.S.; Choi, Y. Keystroke Patterns as prosody in digital writings: A case study with deceptive reviews and essays. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 26–28 October 2014; pp. 1469–1473. [CrossRef]

59. González, N.; Calot, E.P. Dataset of human-written and synthesized samples of keystroke dynamics features for free-text inputs. *Data Brief* **2023**, *48*, 109125. [CrossRef]
60. Tewari, A.; Verma, P. An Improved User Identification based on Keystroke-Dynamics and Transfer Learning. *Webology* **2022**, *19*, 5369–5387. [CrossRef]
61. Nirmal, J.R.; Kiran, R.B.; Hemamalini, V. Improvised multi-factor user authentication mechanism using defense in depth strategy with integration of passphrase and keystroke dynamics. *Mater. Today Proc.* **2022**, *62*, 4837–4843. [CrossRef]
62. TypingDNA. Available online: www.typingdna.com/ (accessed on 19 May 2023).
63. Fernando, K.J.L.; Jayalath, W.J.D.L.D.D.; Ranasinghe, A.D.R.N.; Bandara, P.K.B.P.S.; De Silva, H. Innovative, Integrated and Interactive (3I) LMS for Learners and Trainers. In Proceedings of the ICAC 2020—2nd International Conference on Advancements in Computing, Malabe, Sri Lanka, 1–11 December 2020; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2020; pp. 37–42. [CrossRef]
64. Chen, C.H. *Fuzzy Logic and Neural Network Handbook*; McGraw-Hill, Inc.: New York, NY, USA, 1990.
65. Kasprowski, P.; Borowska, Z.; Harezlak, K. Biometric Identification Based on Keystroke Dynamics. *Sensors* **2022**, *22*, 3158. [CrossRef] [PubMed]
66. Shan, X.; Ma, T.; Gu, A.; Cai, H.; Wen, Y. TCRNet: Make Transformer, CNN and RNN Complement Each Other. In Proceedings of the ICASSP 2022—2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Singapore, 23–27 May 2022; Institute of Electrical and Electronics Engineers (IEEE): Piscataway, NJ, USA, 2022; pp. 1441–1445. [CrossRef]
67. Olah, C. LSTMs. Available online: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed on 29 May 2023).
68. Dhakal, V.; Feit, A.M.; Kristensson, P.O.; Oulasvirta, A. Observations on typing from 136 million keystrokes. In Proceedings of the CHI '18: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, Montreal, QC, Canada, 21–26 April 2018. [CrossRef]
69. Mishra, A. IIITBh-Keystrokes Database. Available online: https://github.com/aroonav/IIITBh-keystroke (accessed on 19 May 2023).
70. Buckley, O.; Hodges, D.; Windle, J.; Earl, S. CLICKA: Collecting and leveraging identity cues with keystroke dynamics. *Comput. Secur.* **2022**, *120*, 102780. [CrossRef]