DOI: https://doi.org/10.52651/sam.a.2022.1.29-36

THE HUMAN INTERFACE DEVICE (HID) ATTACK ON ANDROID LOCK SCREEN NON-BIOMETRIC PROTECTIONS AND ITS COMPUTATIONAL COMPLEXITY

Sebastián POTOCKÝ, Jozef ŠTULRAJTER

Abstract: Nowadays, information obtained from mobile phones is often the subject of evidence in front of a court. Forensic analysts often come across smartphones about which they have no prior information. However, they need to extract data from them. The main prerequisite to extract the data is to bypass Android lock screen protection. The HID attack is a promising method to break Android lock screen protection. In many cases, this is the only way how to break the smartphone´s non-biometric lock screen protections on newer Android OS versions. The article contains examples of three non-biometric types of Android smartphone lock screen protections and their computational complexity. The paper describes hardware and software requirements for implementation of HID attack.

Keywords: Android; HID; Attack; Bypass; Protection; PIN; Pattern; Password.

1 INTRODUCTION

It is not an impossible task to break into a locked device and access the data. There are various ways how their lock screens can be breached or bypassed. Some of them are applicable for all Android devices, or all possible situations. Generally, there are three common techniques how to access data from Android devices. They are manual, logical and physical data acquisitions [1].

Manual acquisition utilizes the user interface to investigate the contents of the phone´s memory and it only acquires the data that appears on the mobile phone. Manual extraction introduces a greater degree of risk in the form of human error, and there is a chance of deleting evidence [1].

Physical acquisitions like imaging an Android phone, JTAG (join test action group), the chip-off technique is a bit-by-bit copy of the physical storage. From Android version 6.0 is encryption turned on by default with full disk encryption or file based encryption and not like a user option and therefore the JTAG and chip-off techniques are almost useless. Thus, this work will not deal with physical acquisition of data due to the encryption mentioned [1].

Logical acquisition extracts logical storage objects, such as files and directories that reside on a filesystem. It contains ADB pull data extraction, ADB dumpsys extraction, ADB backup extraction, browsing SQL data. Data obtained by logical extraction such as call history, SMS/MMS, photos, videos, documents, calendars, GPS locations, browser history information, social networking chats, backup extraction, dumpsys extraction are usually sufficient to clarify the case [1].

The main prerequisite for a successful logical acquisition is to break the locked screen protection of device. Device´s lock screen protection types are divided into biometric and non-biometric ones and can be bypassed with root privileges, but, in some cases, also without them [1].

Rooting usually wipes the phone, so all your files (non-system files) are unlinked, deleted [1].

A human interface device (HID) attack is a simulation of human activity programmatically. A HID attack vector is a combination of customized hardware and restriction bypass via mouse or keyboard emulation [2].

It is a scenario in which an attacker takes a programmable embedded development platform such as the Teensy, Arduino or in our case smartphone and an associated software package to create a USB device which, when plugged into a smartphone, will execute a pre-configured set of keystrokes to break the lock screen protection and allow the logical acquisition [2].

There are two main advantages of the HID attack. The locked phone does not need to be rooted. There is no need to enable USB debugging. USB debugging allows an Android device to communicate with a computer that's running the Android SDK tools in order to use advanced operations [1] [2].

Based on this, HID attack is a promising method for obtaining data from Android devices.

This attack can be executed on the non-biometric lock screen protections, such as pin, pattern or password.

The goal of this paper is to explain the HID attacks and describe their input structure and computational complexity.

2 TYPES OF NON-BIOMETRIC LOCK SCREEN PROTECTIONS AND THEIR COMPUTATIONAL COMPLEXITY

Complexity is a measurement of how fast and efficient an algorithm performs based on an input size and situation in which the algorithm has to run [3].

Computational complexity is divided into two types, memory complexity and time complexity [3].

Memory complexity measures how much computer memory the algorithm would take. Memory

requirement is irrelevant in HID attack, given the huge resources modern computers possess [3].

Time complexity measures approximate number of operations an algorithm takes when processing an input of a certain size [3].

The vital pointer is time requirement which is mostly affected with timeouts during the HID attack. Timeout is a certain period of time that the user has to wait before entering a new combination of pin, pattern or password after too many incorrect attempts have been entered [4].

Each manufacturer specifies different types of timeouts for different types of their phones. There is no way to bypass timeout without previous exploiting. Even though, timeouts are a means of protection against brute force attacks, but, in some cases, it is still possible to execute a HID attack [4].

For instance, Samsung smartphones use timeouts which are mentioned below [4]:

- After 1-5 wrong attempts – 1x 30 seconds timeout.
- After 6-10 wrong attempts – 1x 30 seconds timeout.
- Between 11-41 wrong attempts – 30 seconds timeout after each wrong attempt.
- After 41 wrong attempts – 60 seconds timeout after each wrong attempt.

2.1 Invalid attempt

Each attempt must meet some minimum requirements. If the entered attempt to break the PIN, pattern and password does not meet even the minimum requirements, this attempt is not affected by the timeout.

When a PIN, passcode or pattern consisting of fewer than four characters are entered ("entered" meaning followed by the "Enter" key), Android does not consider that an actual unlock attempt. It will show no message from the Android Lock Screen saying "Failed Attempt", "Try Again," etc. Incorrect Passcode event wouldn't trigger due to an incomplete attempt. If the input is four digits or greater, Android will display a message along the lines of "PIN incorrect, please try again" or "Wrong PIN" [5].

Therefore, inputs smaller than 4 digits or characters, will not be taken into account during measures.

2.2 PIN

Rules:

- Software requirement of a PIN input in android devices prerequisite minimum 4 digit input, but no more than 16 digit input, which consist of Arabic numerals.

A formula for computational complexity is variation with repetition.

$$V_n^k = n^k \tag{1}$$

The table below shows the time complexity of breaking an Android screen protected by a PIN of different lengths. Total time was calculated from the number of attempts executed in the worst-case and the timeouts of Samsung smartphones between incorrect attempts which were mentioned in section 2.

Tab. 1 Time complexity (total attempts) of check all PIN inputs calculated with the formula of variation with repetition

n	k	Total attempts	Total time
10	4	10 000	6,927 days
10	5	100 000	69,427 days
10	6	1 000 000	1,902 years
10	7	10 000 000	19,025 years
10	8	100 000 000	190,258 years
10	9	1 000 000 000	1902,587 years
10	10	10 000 000 000	19025,874 years

Source: author.

2.3 Pattern

A valid unlock pattern should follow the following rules:

- A pattern should connect at least 4 dots.
- A dot can be connected only once, meaning that
  a pattern connects no more than 9 dots.
- A pattern will always connect the first unconnected dot along its path. Then it may go further to connect other unconnected dots.
- A pattern can go through a previously connected dot along its path in order to connect an unconnected dot [6].

At first glance, it might seem that the total number of attempts is calculated using variation without repetition formula. However, because of the limitations mentioned above it is much less.

$$V_n^k = \frac{n!}{(n-k)!} \tag{2}$$

For comparison, the table below shows the computational complexity of the PATTERN, if it were calculated according to the variation without repetition formula.

Tab. 2 Time complexity (total attempts) of check all PATTERN inputs calculated with the formula of variation without repetition

n	k	Total attempts	Total time
9	4	3024	2,083 days
9	5	15120	10,483 days
9	6	60480	41,983 days
9	7	181440	125,994 days
9	8	362880	251,983 days
9	9	362880	251,983 days

Source: author.

Problem of all valid PATTERNs was solved by generating all patterns that follow the rules described above. [6], [7].

The table below shows the time complexity of breaking an Android screen protected by the PATTERN of all possible combinations in a 3 x 3 grid, which is default option in the most smartphones. Total time was calculated from the number of attempts executed in the worst-case and the timeouts of Samsung smartphones between incorrect attempts which were mentioned in section 2.

Tab. 3 Size statistics of all valid patterns

n	k	Total attempts	Total time
9	4	1624	1,079 days
9	5	7152	4,949 days
9	6	26016	18,049 days
9	7	72912	50,616 days
9	8	140704	97,694 days
9	9	140704	97,694 days

Source: [6].

The table below proves, that real computation complexity of PATTERN is on average 2,3x less than expected.

Tab. 4 Comparing of real computation complexity with variation without repetition

n	k	Variation without repetition	Real computation	Total
9	4	3024	1624	1,862x less
9	5	15120	7152	2,114x less
9	6	60480	26016	2,325x less
9	7	181440	72912	2,488x less
9	8	362880	140704	2,579x less
9	9	362880	140704	2,579x less

Source: author.

2.4 Password

Rules:

- Software requirement of a password input in Android devices prerequisite minimum 4 character input, but no more than 16 character input, which consist of small letters, capital letters with or without diacritics (it depends on language), Arabic numerals and special characters.

A formula for computational complexity is variation with repetition (1).

The tables below show the time complexity of breaking an Android screen protected by a PASSWORD of four and five digits lengths. PASSWORDs containing lowercase, uppercase, and numbers are included in the comparison. Total time was calculated from the number of attempts executed in the worst-case and the timeouts of Samsung smartphones between incorrect attempts which were mentioned in section 2 [8].

Tab. 5 Four digits password

Small letter	Capital letter	Numbers	n	k	Total attempts	Total time
YES	NO	NO	26	4	456 976	317 days
YES	YES	NO	52	4	7 311 616	13,910 years
YES	YES	YES	62	4	14 776 336	28,113 years

Source: author.

Longer PASSWORDs are irrelevant for comparison due to their high computational complexity already with a 5-character password, a significant increase can be seen.

Tab. 6 Five digits password

Small letter	Capital letter	Numbers	n	k	Total attempts	Total time
YES	NO	NO	26	5	11 881 376	22,605 years
YES	YES	NO	52	5	380 204 032	723,371 years
YES	YES	YES	62	5	916 132 832	1743,022 years

Source: author.

Special characters that would only increase the computational complexity were not taken into account in the comparison.

2.5 Overview of time complexity

Up to now, HID attack has been described as a brute-force attack with timeouts. Looking at the time complexity, it is clear that HID attack is not effective in every case. It does not make sense to run attack that can last several years in the worst case. Every HID attack should finish within a reasonable time.

Therefore, it is desirable to know the time complexity of each case in the worst case.

The Table 6 below shows the time complexity of non-biometric Android lock screen protections from the weakest to the strongest.

Tab. 7 Overview of type protections and their time complexity

Type of protection	Time complexity in worst case
4 dots pattern	1,079 days
5 dots pattern	4,949 days
4 digits pin	6,927 days
6 dots pattern	18,049 days
7 dots pattern	50,616 days
5 digits pin	69,427 days
8 dots pattern	97,694 days
9 dots pattern	97,694 days
4 digits password from small letters	317,327 days
6 digits pin	1,902 years
4 digits password from small and capital letters	13,910 years
7 digits pin	19,025 years
5 digits pin from small letters	22,605 years
4 digits password from small, capital letters and numbers	28,113 years
8 digits pin	190,258 years
5 digits password from small and capital letter	723,371 years
5 digits password from small, capital letters and numbers	1743,022 years
9 digits pin	1902,587 years
10 digits pin	19025,874 years

Source: author.

It is obvious that the 5 dot long pattern is similar in strength to a 4 digit PIN combination and a 7 dot long pattern is similar in strength to a 5 digit PIN combination. However, a 6 digit PIN is already more secure than all the patterns combined together.

Breaking a 6 or more digit PIN with brute-force is irrelevant, because it will take a lot of time.

As it can see on table above, HID attack of passwords is computationally intensive.

Otherwise, the time complexity can be significantly decreased by entering password´s inputs successive from the most to the least probable, like in the dictionary attack. There is no other effective option because random guessing of letters is ineffective.

The difference with brute force attack is that, in brute force, a large number of possible key permutations are checked whereas, in the dictionary attack, only the words with the most possibilities of success are checked and therefore it is less time consuming than the brute force one.

There are many articles that deal with the probability of selected pins, passwords or patterns according to human psychology [9], [10], [11].

Locked and attack devices have the ports for charging their batteries occupied during the whole HID attack. It is necessary to interrupt the attack while recharging the batteries. This situation can occur several times during an attack [4].

Therefore, the total time of breaking a lock screen protection will be affected by charging time of the locked or attack devices.

3 HARDWARE AND SOFTWARE REQUIREMENTS

There are many tools that can perform a HID attack like Rubber Ducky [12], Teensy [13], Cellebrite [14], XPIN Clip [15], etc. However, these solutions require special hardware and no documentation is published, as they are commercial paid tools.

First of all, we have to create a cracking device. It means we need a rooted Android device with HID kernel support. The most famous software is Kali NetHunter. Kali NetHunter is a free and open-source mobile penetration testing platform for Android devices, based on Kali Linux. However, Kali NetHunter is not necessity. The most important thing is to prepare enabled HID endpoints and these are then mirrored to our victim device.

The basic HID handling is done in the kernel, and HID reports can be sent/received through I/O on the /dev/hidgX devices (keyboard, mouse, joystick). For our purposes it is /dev/hidg0 for keyboard and /dev/hidg1 for mouse [16].

To use these devices properly, formatted input has to be sent to them.

3.1 USB keyboard keypress mechanism

Report format must be created according to certain rules and must be of a certain length.

The USB keyboard report may be up to 8 bytes in size, although not all these bytes are used, it's possible to implement a proper implementation using only the first three or four bytes [17].

Tab. 8 B.1 Protocol 1 (Keyboard)

Byte	Description
0	Modifier keys
1	Reserved field (unused/reserved for OEM)
2	Keypress 1
3	Keypress 2
4	Keypress 3
5	Keypress 4
6	Keypress 5
7	Keypress 6

Source: [18].

Not every character on the keyboard corresponds to a single keystroke. To write uppercase letters, special characters and diacritics is necessary to use modifier keys.

Modifier keys in HID report is a bitfield, where each bit corresponds to a specific modifier key. When a bit is set to 1, the corresponding modifier key is being pressed [18].

Tab. 9 The bit structure of modifier keys byte

Bit	Bit Length	Description
0	1	Left Ctrl
1	1	Left Shift
2	1	Left Alt
3	1	Left GUI (Win/Super key)
4	1	Right Ctrl
5	1	Right Shift
6	1	Right Alt
7	1	Right GUI(Win/Super key)

Source: [17].

Reserved field in report format is unused or reserved for OEM (original equipment manufacturer). This byte is reserved by the USB HID specification, and thus software should ignore it [17].

Keyboard report can indicate up to 6 keypresses. All these values are unsigned 8-bit values [17].

The exact description of keypresses is in the specification of HID Usage Tables for Universal Serial Bus version 1.22 expressed by Usage ID.

Usage IDs are part of the HID Report descriptor and supply an application developer with information about what a control is actually measuring or reporting. During HID attack Usage ID determines key codes to be used in implementing a USB keyboard [19].

For instance, sending a Capital H in normal keyboard requires press Right shift and small h. Right shift keystroke requires the fifth bit set to 1. Binary number 00100000 is converted to hexadecimal number 20. Usage ID of keystroke of small h letter is 0B. These values are pasted into the correct location in the keyboard report. The report can be sent to /dev/hidg0 device using long version or short version of the report.

Long version of the report:
\0x20\0x00\0x0B\0x00\0x00\0x00\0x00\0x00

Short version the report:
\x20\0\x0B\0\0\0\0\0

3.2 USB mouse X, Y movement

During creating a USB HID report is important to take account that graphical pattern is not a set of random numbers or characters like at PINs or PASSWORDs.

The structure of the PATTERN can be imagined as a set of consecutive lines that are connect at a common point.

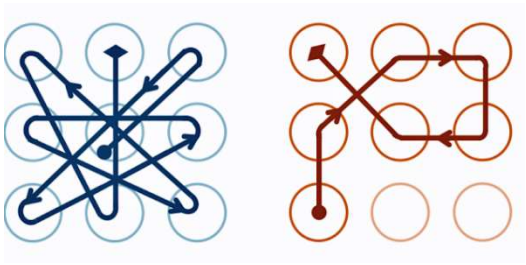

Fig. 1 Example of pattern
Source: [20].

Each line can be drawn using a Cartesian coordinate system with X and Y coordinates.

Therefore, each line will be the separate input in the proper format.

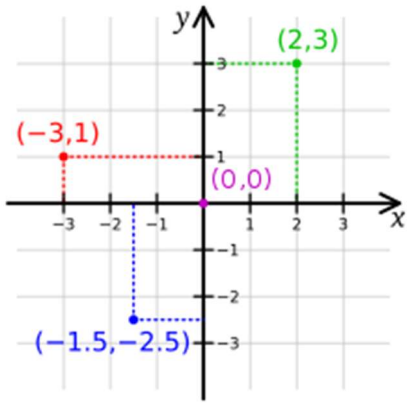

Fig. 2 Cartesian coordinate system
Source: [21].

USB mouse, just like any other HID device, communicate with the software using reports, which are sent via endpoints. Only the first three bytes of the USB mouse report are defined. The remaining bytes, if exist, may be used for device-specific features [17].

Tab. 10 B.2 Protocol 2 (Mouse)

Byte	Bits	Descriptions
0	0	Button 1
	1	Button 2
	2	Button 3
	4 to 7	Device-specific
1	0 to 7	X displacement
2	0 to 7	Y displacement
3 to n	0 to 7	Device specific (optional)

Source: [18].

When pattern is drawing to smartphone screen, finger is in contact with the screen all the time. It is equivalent to create a line in computer screen when the left button of the computer´s mouse be pressed during a mouse movement. In the USB mouse report is button status set in the first byte. This byte is a bitfield, in which the lowest three bits are standard format. The remaining 5 bits may be used for device-specific purposes [17].

Tab. 11 The bit structure of button status

Bit	Bit Length	Description
0	1	When set to 1, indicates the left mouse button is being clicked.
1	1	When set to 1, indicates the right mouse button is being clicked.
2	1	When set to 1, indicates the middle mouse button is being clicked.
3	5	These bits are reserved for device-specific features.

Source: [17].

The HID attack on PATTERN requires a left mouse button to be pressed during each attempt. The direction and length of the line are determined by X and Y movements.

X movement is a 8-bit signed integer (0x00) that represents the X movement. When this value is negative, the mouse is being moved to the left. When this value is positive, the mouse is being moved to the right [17].

Y movement is a 8-bit signed integer (0x00) that represents the Y movement. When this value become negative, the mouse was moved up. When the value is positive, the mouse is being moved down [17].

In decimal notation, the convention is to precede the number with „+“ or „-“ to indicate whether it´s positive or negative, usually omitting the „+“ to simplify the notation for positive numbers. In binary this problem is solved by signed magnitude [22].

For instance, sending a mouse movement 300 pixels right and holding left mouse button at the same time requires proper formatted input below.
The report can be sent to /dev/hidg1 device using long version or short version of the report.

Long version of the report:
\0x01\0xFED4\0x00\0x00

Short version of the report:
\x01\0\xFED4\0\0

4 HID ATTACK IMPLEMETATION

Android-PIN-Bruteforce is an open source solution developed by Adam Horton. The solution uses a USB OTG cable to connect the locked phone to the Nethunter device. It emulates a keyboard, automatically tries PINs and waits after trying too many wrong guesses. The USB HID Gadget driver provides emulation of USB Human Interface Devices. This enables an Android Nethunter device to emulate keyboard input to the locked phone, like plugging a keyboard into the locked phone and pressing keys [4].

Fig. 3 Involvement of HID attack
Source: [4].

Required components: [4]

- A locked Android
- A Nethunter phone (or any rooted Android with HID kernel support)
- USB OTG (On the Go) cable/adapter (USB male Micro-B/C to female USB A), and a standard charging cable (USB male Micro-B/C to male A).

Advantages: [4]

- No need to enable USB debugging
- The locked phone does not need to be rooted
- Special hardware is not required
- Backoff time to crack other types of devices is configurable
- Detects when the phone is powered off (Low Power warning pop-ups)
- Detects when the phone is unplugged and waits while retrying every 5 seconds
- Optimised PIN list sorted by probability
- Log file

Disadvantages: [4]

- Only for Android
- Only for PINS
- No log of the correct guessed PIN

Android-PIN-Bruteforce was executed on 10 devices of different version of Android. In the measurement, only the executability on device was examined not total time of execution. The first point of measurement was whether Android-PIN-Bruteforce could repeat the attack with another PIN according to the configuration after an incorrect attempt on measured device. The second point measured whether the device was properly unlocked after sending the correct PIN. The table below shows that the Android-PIN-Bruteforce could not be launched only on devices with a version of Android lower than 5.0.

No.	Smartphone/tablet type	Android version	USB type	Executable
1	Smartphone Samsung galaxy S4 mini	4.4.2	USB B	NO
2	Tablet Lenovo Yoga 2-10 50F	5.0.1	USB B	YES
3	Tablet Huawei MediaPad T5	8.0.0	USB B	YES
4	Tablet Lenovo Touchpad 2016	8.1.0	USB B	YES
5	Smartphone Samsung galaxy J7	9	USB B	YES
6	Xiaomi Mi A2 Light	10	USB B	YES
7	Samsung galaxy A10	10	USB C	YES
8	Samsung galaxy S10x	11	USB C	YES
9	Samsung galaxy A71	11	USB C	YES
10	Samsung galaxy S10x	12	USB C	YES

Tab. 12 Measurements of HID attack.
Source: author.

5 CONCLUSION

This work presents an overview of time complexity of all types of Android lock screen protections.

PATTERN is vulnerable to HID attack due to its low computational complexity.

In this paper it has been presented that PIN and PASSWORD time complexity grow exponentially.

The HID attack is effective only for 4 and 5 digit PINs.

For PASSWORD it is beneficial to create a table of the most commonly used passwords, like in the dictionary attack.

The paper also describes creation of proper formatted inputs, which an attacker has to send to the locked device.

The attack is executable on the vast majority of devices with a higher version of the operating system of Android.

Further work might be focused on creating a cracking device and executable scripts, which can bypass not only PIN, but also PASSWORD and PATTERN protections

We can study:

- Using other methods of generating inputs. For example, using the Linux USB HID gadget driver.
- Testing them on different types of smartphones or tablets with different timeouts from different manufacturers.
- Determine that a HID attack has been performed on a device using forensic analysis.
- Design of a method to protect your phone from HID attacks.

References

[1] TAMMA, H., H. SKULKIN, H. MAHALIK and S. BOMMISETY. *Practical Mobile Forensics Fourth Edition.* Birmingham: Packt Publishing, 2020. s. 604. ISBN 978-1-83864-752-0.

[2] SYED MUQARRAB-UL-AHAD ZAIDI. *What are HID Attacks? How to perform HID Attacks using Kali NetHunter?* [online]. Pakistan: U.S. University of Agriculture, 2018. Available at: https://www.researchgate.net/publication/323111273_What_are_HID_Attacks_How_to_perform_HID_Attacks_using_Kali_NetHunter

[3] *Demystifying the Big O Notation.* [online]. [accesed 27. October 2021]. Available at: https://www.digitalonus.com/demystifying-the-big-o-notation/

[4] *Android PIN Bruteforce.* [online]. [cit. 30. August 2021]. Available at: https://github.com/urbanadventurer/Android-PIN-Bruteforce

[5] *Incorrect Passcode and the Android pattern lock.* [online]. [cit. 17. September 2021]. Available at: https://personal.support.lookout.com/hc/en-us/articles/202929734-Incorrect-Passcode-and-the-Android-pattern-lock

[6] SUN, Ch., Y. WANG and J. ZHENG. Dissecting pattern unlock: The effect of pattern strength meter on pattern selection. [online]. In *Journal of Information Security and Applications*, 2014, 19. 4-5: 308-320. Available at: https://doi.org/10.1016/j.jisa.2014.10.009

[7] Aviv AJ, Gibson K, Mossop E, Blaze M, Smith JM. Smudge attacks on smartphone touch screens. In: Proceedings of the 4th USENIX conference on Offensive technologies, WOOT'10. Berkeley, CA, USA: USENIX Association; 2010. Available at: https://dl.acm.org/doi/10.5555/1925004.1925009

[8] *Letters in the alpha bet.* [online]. [cit. 10. September 2021]. Available at:

https://www.worldometers.info/languages/how-many-letters-alphabet/

[9] LØGE, M. D. *Tell Me Who You Are and I Will Tell You Your Unlock Pattern*. [online]. U.S. Master's Thesis. Norwegian University of Science and Technology, Department of Computer and Information Science July, 2015. Available at: https://core.ac.uk/download/pdf/154670387.pdf

[10] *PIN analysis.* [online]. [cit. 30. September 2021]. Available at: https://datagenetics.com/blog/september32012/index.html

[11] MARKERT, P. et al. This pin can be easily guessed: Analyzing the security of smartphone unlock pins. In *2020 IEEE Symposium on Security and Privacy (SP).* IEEE, 2020. p. 286-303. Available at: https://doi.org/10.1109/SP40000.2020.00100

[12] *USB rubber ducky.* [online]. [cit. 17. December 2021]. Available at: http://www.sigint.sk/e-shop/758/5/radio-prijimace-vysielace/rubber-ducky-detail

[13] *Teensy® USB Development Board.* [online]. [cit. 17. December 2021]. Available at: https://www.pjrc.com/teensy/

[14] *Cellebrite.* [online]. [cit. 17. December 2021]. Available at: https://cellebrite.com/

[15] *XPin Clip.* [online]. [cit. 17. December 2021]. Available at: https://xpinclip.com/

[16] *Linux USB HID gadget driver.* [online]. [cit. 8. October 2021]. Available at: https://xpinclip.com/https://www.kernel.org/doc/html/latest/usb/gadget_hid.html>

[17] *USB Human Interface Devices.* [online]. [cit. 15. September 2021]. Available at: https://wiki.osdev.org/USB_Human_Interface_Devices

[18] USB IMPLEMENTS' FORUM. *Device Class Definition for Human Interface Devices (HID) Firmware Specification* - 6/27/01 Version 1.11, 1996-2001. 97 s.

[19] *HID Usage Tables for Universal Serial Bus version 1.22.* [online]. [cit. 5. September 2021]. Available at: https://www.usb.org/sites/default/files/hut1_22.pdf

[20] *Hacking Android Pattern Lock (ALP).* [online]. [cit. 12. November 2021]. Available at: https://www.hackcave.net/2015/08/hacking-android-pattern-lock.html

[21] *Cartesian coordinate system*. [online]. [cit. 16. November 2021]. Available at: https://en.wikipedia.org/wiki/Cartesian_coordinate_system

[22] *Two's complement*. [online]. [cit. 2. September 2021]. Available at: https://en.wikipedia.org/wiki/Two%27s_complement

Lt. Dipl. Eng. Sebastián **POTOCKÝ** (PhD. student)
Armed Forces Academy of General M. R. Štefánik
Department of Computer Science
Demänová 393
031 01 Liptovský Mikuláš
Slovak Republic
E-mail: sebastian.potocky@gmail.com

Prof. Dipl. Eng. Jozef **ŠTULRAJTER**, CSc.
Armed Forces Academy of General M. R. Štefánik
Department of Computer Science
Demänová 393
031 01 Liptovský Mikuláš
Slovak Republic
E-mail: jozef.stulrajter@aos.sk

Sebastián Potocký was born in Slovakia in 1994. He received his engineering degree from the Armed Forces Academy of General M. R. in the field of Military Communication and Information Systems. He is currently a web applications group commander - the base of stationary communication and information systems. His research is focuses on development of application, reverse engineering, cyber security.

Jozef Štulrajter works as a professor at the Department of Informatics, Armed Forces Academy of General M. R. Štefánik in Liptovský Mikuláš. He graduated (Ing.) at the Military Technical College in 1974. He obtained the degree of CSc. diploma in Theoretical Electrical Engineering - Theory of Circuits and Systems of the Military Academy in Liptovský Mikuláš in 1992. His research interests include Information and Communication Technology (ICTs), computer architectures, image coding, computer security.