SySS
THE PENTEST EXPERTS.

IT SECURITY KNOW-HOW

Matthias Deeg and Gerhard Klostermeier

Of Mice and Keyboards

On the Security of Modern Wireless Desktop Sets

June 2017

Schaffhausenstraße 77, 72072 Tübingen, Germany

+49 (0)7071 - 40 78 56-0

info@syss.de

www.syss.de

1 Introduction

Wireless desktop sets consisting of a wireless mouse, a wireless keyboard, and a USB dongle have become more popular and more widespread in the last couple of years. Seen as a potential target, those radio-based devices are of more interest to people with malicious intentions than their wired counterparts, due to the fact that they can also be attacked remotely from a safe distance via radio signals.

As wireless desktop sets represent an attractive target both allowing to take control of a computer system and to gain knowledge of sensitive data like passwords, they have been frequently analyzed for security vulnerabilities and were successfully attacked in the past. One well-known example for exploiting vulnerabilities in wireless keyboards is the open source wireless keyboard sniffer KeyKeriki by Dreamlab Technologies, the first version of which was presented back in 2009 for Microsoft keyboards using the 27 MHz ISM band. The second version of which also supported wireless keyboards using the 2.4 GHz ISM band and was presented in 2010 [1]. In 2015, Samy Kamkar published an Arduino-based wireless keyboard sniffer for Microsoft keyboards with known security weaknesses that extended the work of the KeyKeriki v2.0 project [2] and of Travis Goodspeed's research concerning Nordic Semiconductor's transceiver family nRF24 [3]. And in spring 2016, a collection of security vulnerabilities found in USB dongles of wireless desktop sets of different manufacturers was released by Bastille Networks Internet Security under the name of MouseJack which allowed keystroke injection attacks [4].

SySS GmbH started a research project about the security of modern wireless desktop sets using AES encryption in 2015, as there was no publicly available data concerning security issues in current wireless mice and keyboards. Until May 2016, several security vulnerabilities in modern wireless desktop sets of different manufacturers like Microsoft, Cherry, Logitech, Fujitsu, and Perixx have been found and reported in the course of our responsible disclosure program.

The found security vulnerabilities can be exploited within different attack scenarios from different attacker's perspectives. On the one hand, there are security issues which require one-time physical access to a keyboard or a USB dongle, for example, to extract cryptographic keys which can be used in further attacks or to manipulate the firmware. On the other hand, there are security issues that can be exploited remotely via radio communication, for instance, replay or keystroke injection attacks due to insecure implementations of the AES-encrypted data communication.

The results of our research show that the security levels of modern wireless desktop sets of different manufacturers are not equal and that some devices are more secure than others. Still, there was no wireless desktop set without any security issues.

In this paper, we will present the results of this research and will demonstrate ways in which modern wireless desktop sets of several manufacturers can be attacked by practically exploiting different security vulnerabilities.

2 Tested Devices and Used Technology

In the course of our research project, we analyzed five wireless desktop sets of different manufacturers that consisted of a keyboard, a mouse, and an USB dongle. All three components contained a radio transceiver for wireless communication. In Figure 1, the wireless desktop set Microsoft Wireless Desktop 2000 is exemplarily shown.

The five tested wireless desktop sets were:

1. Microsoft Wireless Desktop 2000
2. Cherry AES B.UNLIMITED
3. Fujitsu Wireless Keyboard Set LX901
4. Logitech MK520
5. Perixx PERIDUO-710W

Four of the five tested devices used low power 2.4 GHz nRF24 transceivers by Nordic Semiconductor, one used a low power 2.4 GHz transceiver by Cypress in combination with a low-voltage microcontroller (CY7C60123-PVXC). The identified transceivers or respectively systems-on-a-chip (SoCs) of all tested wireless keyboards and USB dongles are shown in Table 2. As most of the tested devices used nRF24 transceivers, we focused our research on that.

Product Name	Keyboard	USB Dongle
Cherry AES B.UNLIMITED	nRF24LE1	nRF24LU1+
Fujitsu Wireless Keyboard Set LX901	CYRF6936	CYRF6936
Logitech MK520	nRF24LE1	nRF24LU1+
Microsoft Wireless Desktop 2000	nRF24LE1H (OTP)	nRF24LU1+
Perixx PERIDUO-710W	nRF24LE1H (OTP)	nRF24LU1+

Table 2: Identified transceivers and SoCs

For analyzing the radio communication and sending radio signals to the components of the wireless desktop sets, we both used a software defined radio (e. g. HackRF One [5] and USRP B200 [6]) and an nRF24-based USB radio dongle Crazyradio PA [7].

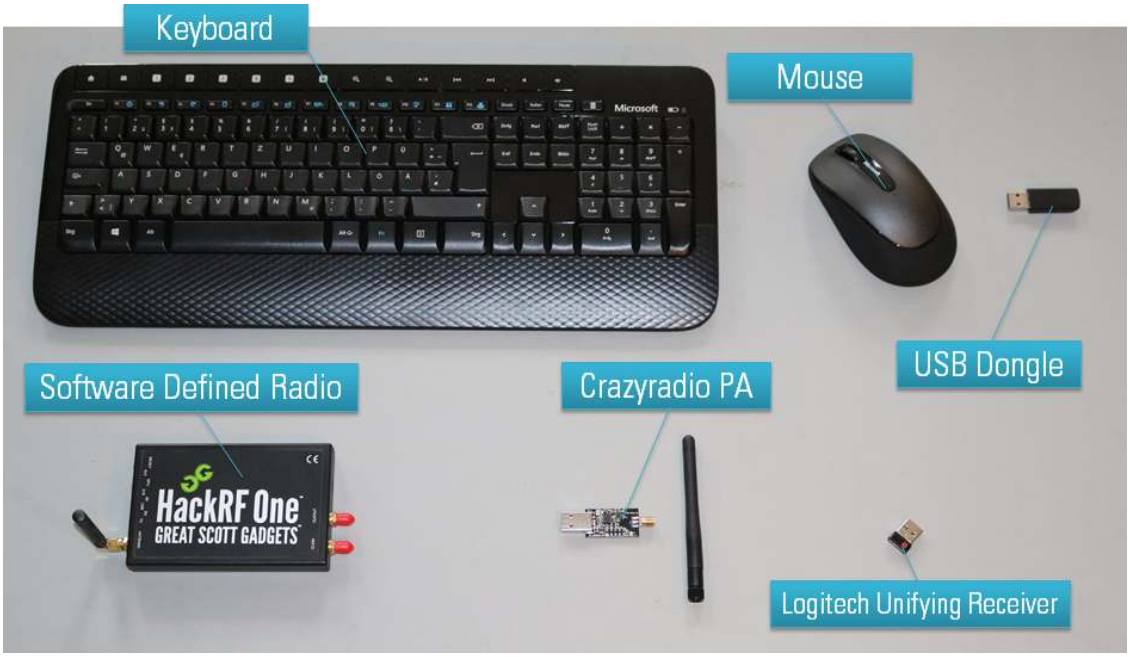

Figure 1: Overview of used hardware

We started to use the Crazyradio PA in combination with Bastille's nRF research firmware [8] and to develop Python tools for it after the MouseJack release in February 2016, because this tool set was superior to the one we used by then. Many thanks to Marc Newlin.

Figure 2 illustrates the general functionality of wireless desktop set, where both the keyboard and the mouse send radio signals (data packets with keystrokes or respectively mouse actions) to an USB dongle that is connected to a computer system. The radio communication is bidirectional as shown in the picture, as the USB dongle acknowledges received data packets.

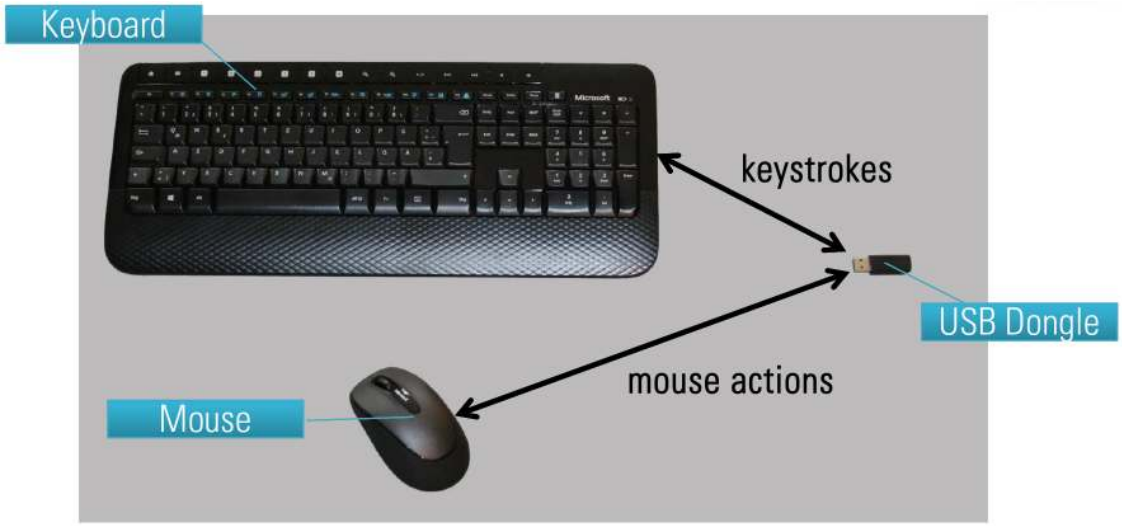

Figure 2: Simplified communication of wireless desktop sets

3 Test Methodology

For our security analysis of the tested wireless desktop sets, we used the following three-step analysis:

1. Hardware analysis
2. Firmware analysis
3. Radio-based analysis

In the course of the hardware analysis, we opened the keyboards, mice, and USB dongles, had a closer look at the printed circuit boards (PCBs), and identified interesting chips that are important for the functionality of the devices, for example, radio transmitters, receivers, or transceivers. After that, we thoroughly read the fine documentation of the identified chips and looked for test points, for instance, concerning Serial Peripheral Interface (SPI).

As Figures 3 and 4 illustrate, the nRF24LE1 transceiver, for example, has a serial peripheral interface (SPI).

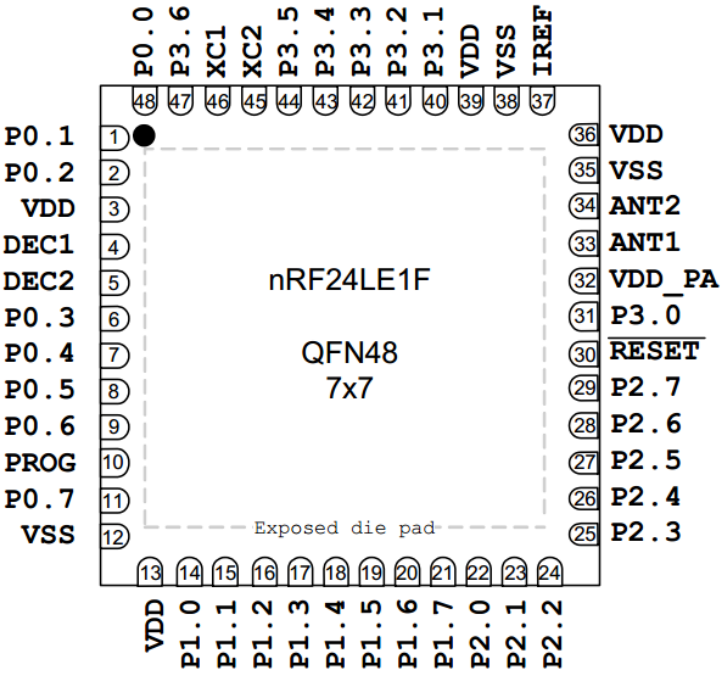

Figure 3: nRF24LE1F pin assignment for a QFN48 package [12]

	24pin-4×4	32pin-5×5	48pin-7×7
FCSN	P0.5	P1.1	P2.0
FMISO	P0.4	P1.0	P1.6
FMOSI	P0.3	P0.7	P1.5
FSCK	P0.2	P0.5	P1.2

Figure 4: Flash SPI slave physical interface for nRF24LE1 [12]

Regarding the nRF24-based devices, SPI can be used to read and write the internal flash memory, if the used configuration of the chips allows this.

Figure 5 exemplarily shows the PCB back side of a Microsoft wireless keyboard where all four SPI soldering points were kindly labeled (`FMISO`, `FMOSI`, `FCSN`, `FSCK`) which saved us some time for this test target (disabled readback protection).

Figure 5: PCB back side of a Microsoft wireless keyboard

Having found the SPI concerning the tested keyboards and USB dongles, we soldered some wires to the identified test points or directly to the SoC pins and tried to dump the firmware of the devices for further analysis. For this, we used a Bus Pirate [9] in combination with the software tool nrfprog [10]. Figure 6 exemplarily illustrates SPI read and write access to a Cherry wireless keyboard.

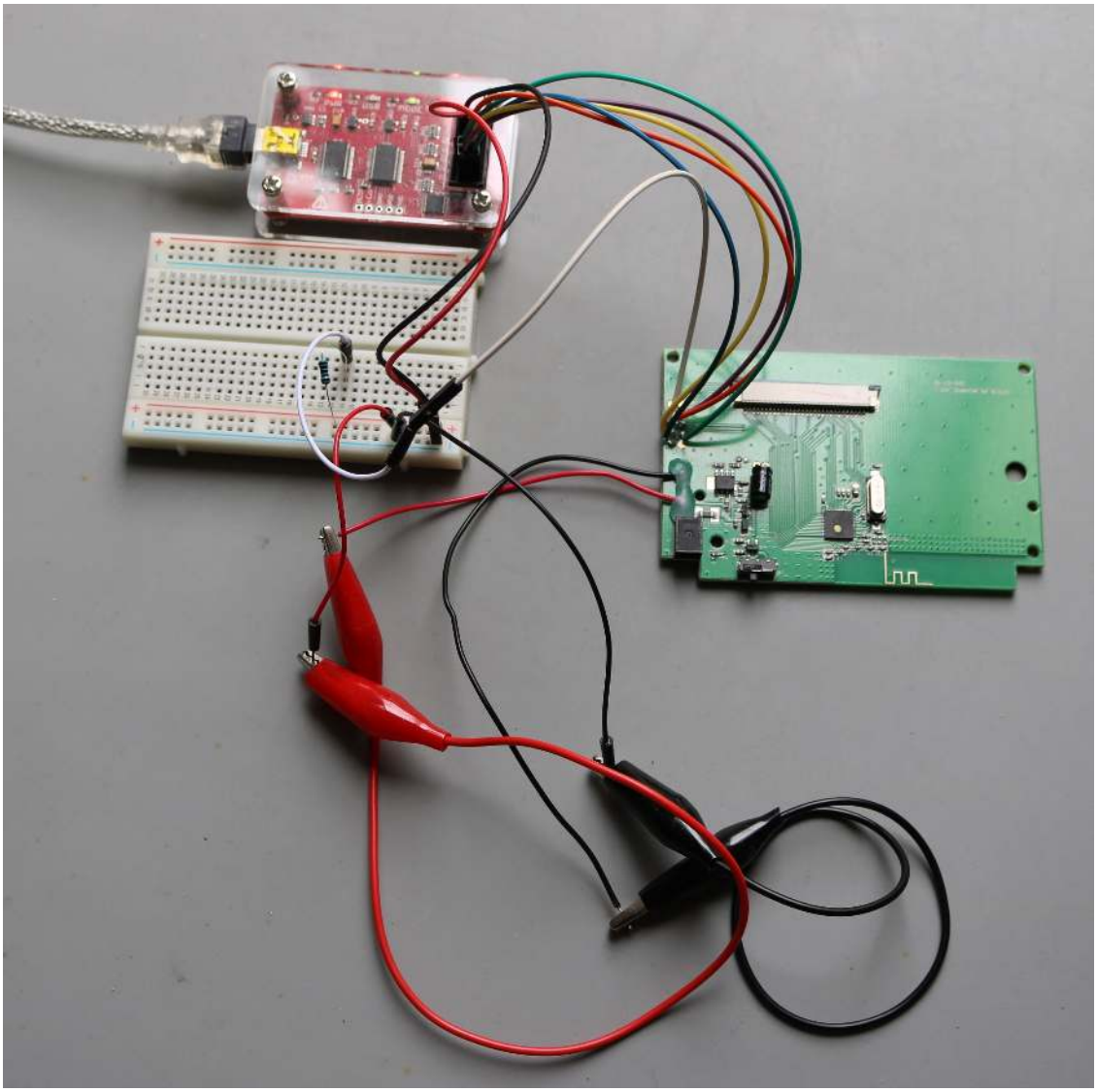

Figure 6: SPI read and write access to a Cherry wireless keyboard

Concerning some of the tested USB dongles, for instance, the one from Cherry, using SPI was a little bit more difficult due to the lack of SPI test points on the PCB and the small size of the used nRF24LU1+ package as Figure 7 illustrates. But with a helping hand of our colleague Alexander Straßheim, testing the SPI access for reading and/or writing the flash memory of the USB dongles was also made possible.

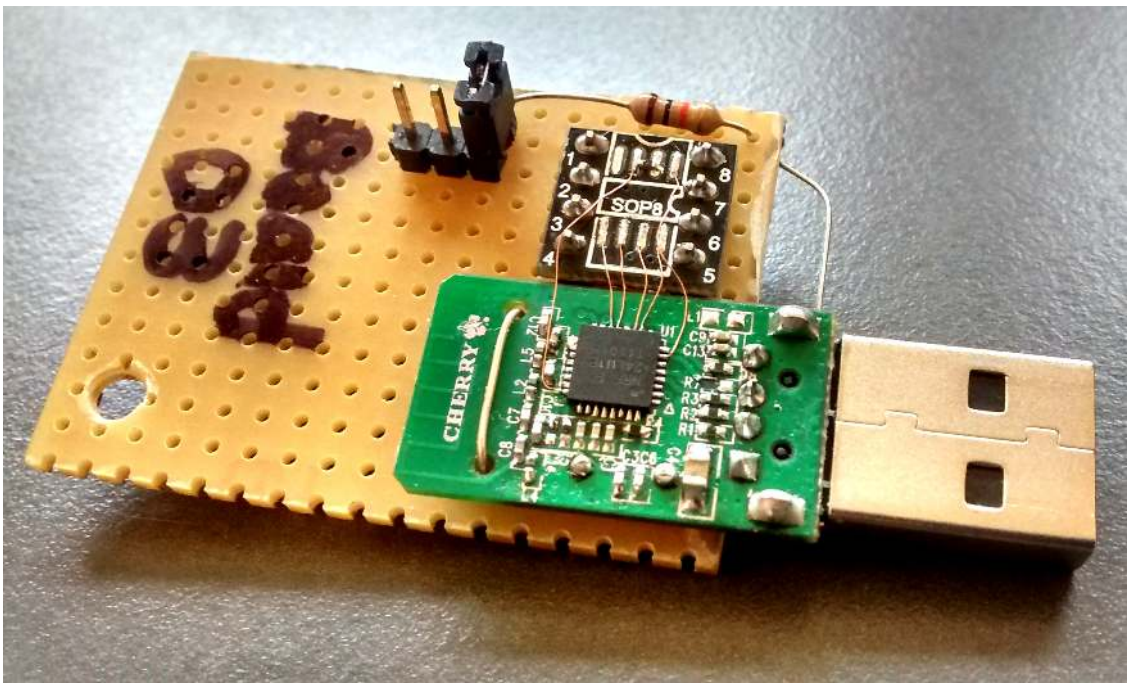

Figure 7: SPI read and write access to a Cherry USB dongle (thanks to Alexander Straßheim)

By being able to read out the firmware of some of the tested devices, we could analyze the dumped firmware images for security issues using reverse code engineering methods, e. g. static code analysis using the software disassembler IDA Pro [11]. The nRF24 SoC possesses an 8051 compatible microcontroller which is described in the Nordic Semiconductor nRF24LE Product Specification [12]. We also had a closer look at the code examples of the Nordic Semiconductor's nRF24 SDK [13] which were a very valuable resource in understanding how nRF24-based devices are programmed and which libraries are already available to developers.

Figure 7 shows an excerpt of an annotated Cherry disassembly of the cryptographic function `hal_aes_crypt` within the disassembler IDA Pro.

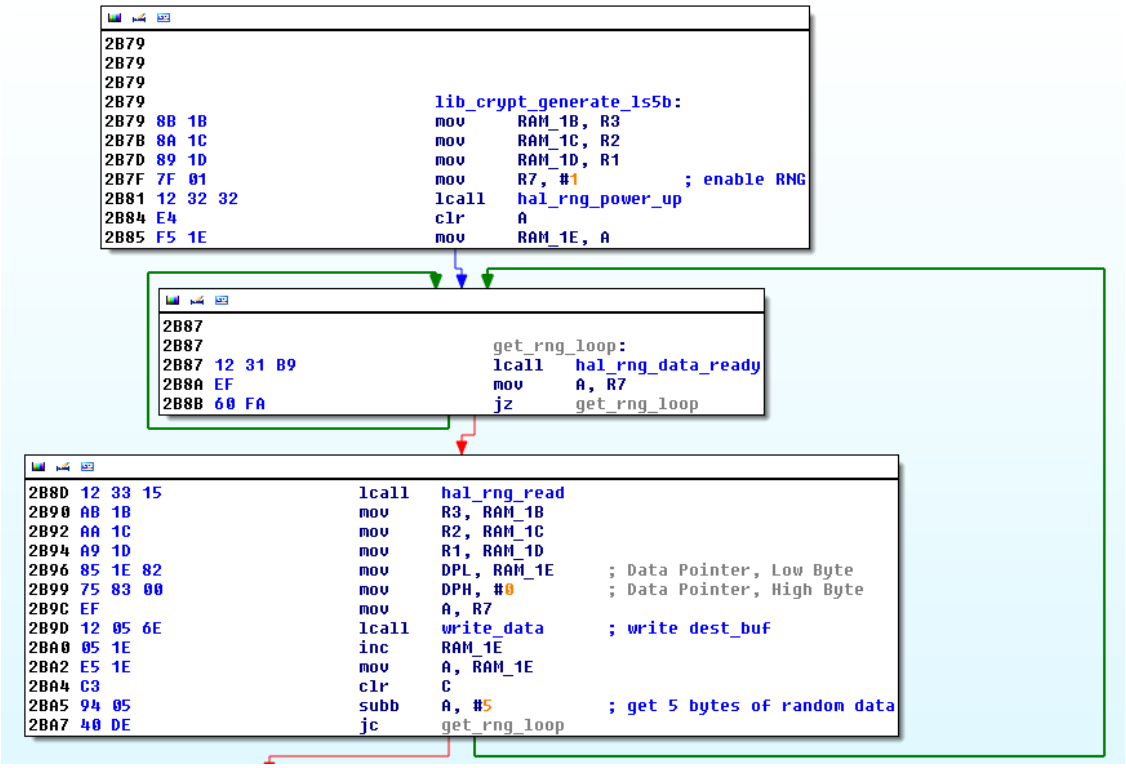

Figure 8: Excerpt of annotated Cherry firmware disassembly (function `hal_aes_crypt`)

In order to gain a better understanding of the nRF24 SoCs, we were not only reading available code provided with the SDK, but also writing and analyzing own compiled sample code like the simple `memcpy` firmware shown in Listing 3.1.

```
/* Really simple memory copy firmware */

#include <Nordic\reg24le1.h>
#include <hal_flash.h>

#define LENGTH    512

// data buffer
static uint8_t xdata buffer[LENGTH];

// Main routine
void main()
{
    uint16_t src_addr = 0xFA00;     // start of extended endurance data in NV memory
    uint16_t dest_addr = 0xFC00;    // start of normal endurance data in NV memory
    uint16_t len = LENGTH;

    // erase normal endurance memory pages (34 and 35)
    hal_flash_page_erase(34);
    hal_flash_page_erase(35);

    // read extended endurance data memory from 0xFA00 to buffer
    hal_flash_bytes_read(src_addr, buffer, len);

    // write buffer to to SPI-addressable NVM (normal endurance memory)
    hal_flash_bytes_write(dest_addr, buffer, len);

    // wait forever
    while(1) {}
}
```

Listing 3.1: Simple `memcpy` firmware for nRF24LE1

Eventually, we also performed a radio-based analysis of the 2.4 GHz wireless communication of individual components of the tested wireless desktop sets. At first, we used software defined radios in combination with GNU Radio [14] and a modified version of the software tool NRF24-BTLE-Decoder [15] for this purpose. Later, we also made use of the nRF24-based USB dongle Crazyradio PA with Bastille's nRF research firmware and our developed Python scripts and tools in order to test specific test cases.

Figure 9 exemplarily shows a GNU Radio Companion (GRC) flow graph that we used in combination with the modified version of NRF24-BTLE-Decoder in order to analyze the sent Enhanced ShockBurst data packets of the tested nRF24-based keyboards and USB dongles.

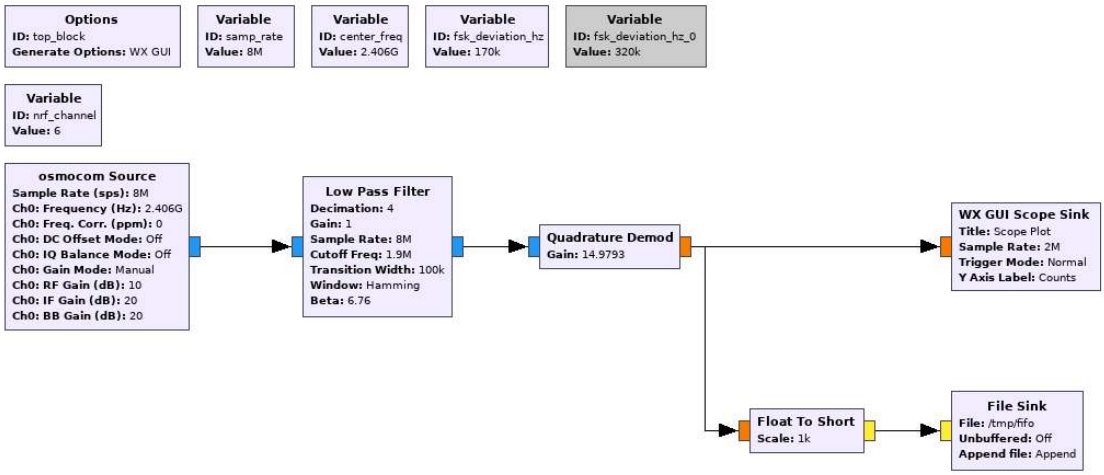

Figure 9: Simple GNU Radio Companion flow graph for use with modified version of NRF24-BTLE-Decoder

The following output shows Enhanced ShockBurst data packets with AES-encrypted payloads of a wireless Cherry keyboard.

```
$ cat /tmp/fifo | ./nrf24-decoder -d 1
nrf24-decoder, decode NRF24L01+ v0.1

(...)
Address: 0x6BB7E29E31 length:16, pid:0, no_ack:0, CRC:0x2D58 data:0294EF5368E70FB11AB685B818819388
Address: 0x6BB7E29E31 length:16, pid:0, no_ack:0, CRC:0x2D58 data:0294EF5368E70FB11AB685B818819388
Address: 0x6BB7E29E31 length:16, pid:0, no_ack:0, CRC:0x2D58 data:0294EF5368E70FB11AB685B818819388
(...)
Address: 0x6BB7E29E31 length:16, pid:0, no_ack:0, CRC:0x2D58 data:0294EF5368E70FB11AB685B818819388
Address: 0x6BB7E29E31 length:16, pid:0, no_ack:0, CRC:0x2D58 data:0294EF5368E70FB11AB685B818819388
Address: 0x6BB7E29E31 length:16, pid:0, no_ack:0, CRC:0x2D58 data:0294EF5368E70FB11AB685B818819388
(...)
```

But the easiest and most stable analysis of the nRF24 radio communication could be performed with the Crazyradio PA.

4 Attack Surface and Attack Scenarios

Concerning the tested wireless desktop sets, we considered two attack scenarios where an attacker either has unrestricted physical access to the components of the wireless desktop set or is only able to attack it via radio signals.

In the first scenario, the attack surface consists primarily of the pinout of the used system-on-a-chip (SoC). In the second scenario, the attack surface solely consists of radio communication.

In both attack scenarios, there are different possible attacks with specific security threats. The following enumeration shows the ones we were interested in during our research project.

1. Physical access to wireless desktop set
   - Extract firmware
   - Manipulate firmware
   - Extract cryptographic key material
   - Manipulate cryptographic key material
2. Attacking via radio signals
   - Exploiting unencrypted and unauthenticated radio communication
   - Replay attacks
   - Keystroke injection attacks
   - Decrypting encrypted data communication

5 Security Vulnerabilities

In the course of our research project, we could find the following five security vulnerabilities in the tested AES-encrypted wireless desktop sets:

1. Insufficient protection of code (firmware) and data (cryptographic key)
2. Unencrypted and unauthenticated data communication
3. Missing protection against replay attacks
4. Insufficient protection against replay attacks
5. Cryptographic issues

All found security vulnerabilities are described in more detail in the following sections and in our published security advisories for the affected devices [16–29].

5.1 Insufficient Protection of Code and Data

During our research, we were able to access both the firmware and the AES cryptographic key (shared secret) of three out of five tested wireless desktop sets, namely Cherry AES B.UNLIMITED, Microsoft Wireless Desktop 2000, and Perixx PERIDUA-710W.

Concerning those three nRF24-based devices, it was possible to access the flash memory via the Serial Peripheral Interface (SPI) as illustrated in Figures 6 and Figure 7 using the software tool nrfprog.

The following is an extract of a memory dump of a Cherry dongle that contains the used AES key.

```
(...)
00007430: 0000 0000 0000 0000 0000 3cdd 9cc7 db74
00007440: 675a c0b2 9796 a55b 913c 0000 0000 0000
00007450: 0000 0000 0000 0000 0000 0000 0000 0000
(...)
```

The following output is an extract of a memory dump of the corresponding Cherry keyboard of the same wireless desktop set that also contains the same AES key.

```
00000000: aa32 1d98 5ef9 3cdd 9cc7 db74 675a c0b2
00000010: 9796 a55b 913c ffff ffff ffff ffff ffff
00000020: ffff ffff ffff ffff ffff ffff ffff ffff
(...)
```

In contrast to the tested Microsoft and Perixx keyboard, the Cherry did not store the cryptographic key in the so-called normal endurance memory but in extended endurance memory. Regarding the nRF24LE1, the 1.5 kB non-volatile (NV) memory is divided into two 256-byte extended endurance pages and two 512 byte normal endurance pages, as described in [12]. And the extended endurance non-volatile memory (EENVM) is not accessible via SPI, as Figure 10 illustrates.

Data memory area	MCU address	SPI address	Page no.
Extended endurance data	0xFA00 - 0xFAFF	NA	32
	0xFB00 - 0xFBFF	NA	33
Normal endurance data	0xFC00 - 0xFDFF	0x4400 - 0x45FF	34
	0xFE00 - 0xFFFF	0x4600 - 0x47FF	35

Figure 10: nRF24LE1 memory mapping for MCU and SPI access

Thus, in order to gain read and write access to the extended endurance memory that contained the cryptographic key of the tested Cherry keyboard, we had to develop simple `memcpy` firmwares that just copied specific memory regions from the extended endurance memory to normal data memory that was accessible via SPI. An example of such a firmware is given in Listing 3.1.

Due to the insufficient protection of code (firmware) and data (cryptographic key), an attacker with physical access to an affected wireless desktop set can extract the used AES key within a couple of minutes and use it later on in radio-based attacks from a safe distance or manipulate the device firmware, for example, to weaken the used cryptography.

Concerning the tested Cherry and Perixx keyboard, it arbitrary read and write access was possible. But this was not true for the tested Microsoft keyboard, as it used a one-time programmable version of the nRF24LE1. This only allows to change a 1 bit to a 0 bit in the flash memory, but not vice versa. Nevertheless, modifications of code and data are still possible in a limited way.

A simple solution for a better protection of code and data of nRF24-based devices is to use the offered readback protection feature (RDISMB – Read DISable Main Block) shown in Figure 11.

RDISMB - Enable Read DISable of MainBlock)

SPI command `RDISMB` enables the readback protection of the flash. The command disables all read/erase and write access to the flash main block from any external interface (SPI or HW debug JTAG). It also disabled erase and write operations in the InfoPage, but read InfoPage read operations are still possible. This will protect code and data in the device from being retrieved through the external flash interfaces.

Figure 11: nRF24 readback protection

Of the four tested nRF24-based devices, only the Logitech keyboard used this feature in order to prevent simple read and write access to the flash memory via SPI.

5.2 Mouse Spoofing Attacks

During our research project, we found out that the radio communication of all tested wireless mice was unencrypted and unauthenticated. Thus, by knowing the used mouse data protocol, an attacker can spoof mouse actions like mouse movements or mouse clicks. This is rather old news and has been reported and demonstrated by different people in the last couple of years at different occasions – but nevertheless exciting. Yet, modern wireless desktop sets with encryption still only encrypt and/or authenticate the keyboard but not the mouse communication which poses a security risk.

By sending forged data packets, an attacker can remotely control the mouse pointer of a target system in an unauthorized way. And by using trial and error and good educated guesses regarding the target system (heuristic method), so-called mouse spoofing attacks can result in successful remote code execution on affected target systems.

The heuristics concern the following issues:

- Operating system (screen layout & content)
- Language settings (screen layout & content)
- Mouse settings (mouse pointer acceleration)
- Settings of the OS's virtual on-screen keyboard (window position)

In our experience, it is a good idea to stick to default settings of systems when it comes to heuristics, like the default mouse settings for Windows 7 and the default window position and dimensions of the Window 7 on-screen keyboard (OSK) that are illustrated in Figure 12.

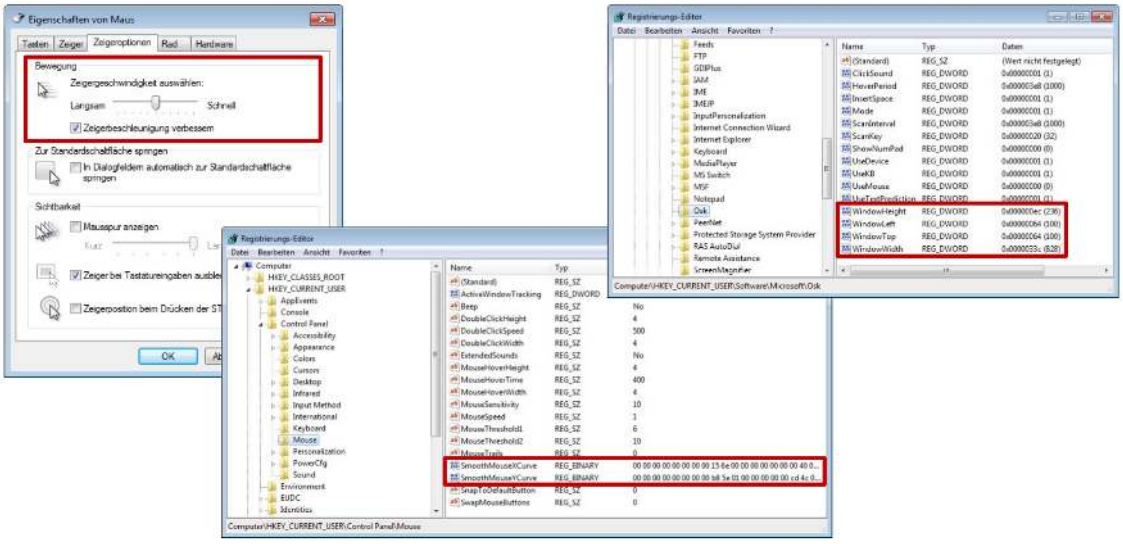

Figure 12: Default mouse settings for Windows 7

In the course of our research project, we developed a proof-of-concept software tool named Radioactive Mouse to perform mouse spoofing attacks. This software tool is part of our nRF24 Playset [31].

Pixel-perfect control over the mouse pointer sounded easy but could not be managed so far. There is more work needed concerning mouse acceleration (reverse engineering the actual algorithms), that is, for instance, used

in different Windows operating systems (implemented in `win32k.sys`), to achieve the desired deterministic behavior.

Thus, our proof-of-concept software tool currently uses handcrafted and slowed down mouse actions for more or less reliable attacks.

Radioactive Mouse in combination with the USB radio dongle Crazyradio PA and Bastille's nrf-reserch-firmware is able to successfully perform mouse spoofing attacks that result in code execution as Figures 13 and 14 illustrate.

Figure 13: Mouse spoofing attack using the software tool Radioactive Mouse

Figure 14: Successful remote code execution via mouse spoofing attack

This attack is also demonstrated in a short video named *Radioactive Mouse States the Obvious* that is available on YouTube [30].

5.3 Replay Attacks

In the course of our research project, we found out that all keyboards and mice of the tested wireless desktop sets were vulnerable to replay attacks due to missing or insufficient protection against this kind of attack.

The Microsoft Wireless Desktop 2000 has a replay protection, but according to our test results, the used window for valid packet counter values is large enough to perform replay attacks under certain conditions where there are only few keystrokes between the recording and replaying.

Concerning wireless desktop sets, replay attacks allow for the following two attacks:

1. Gaining unauthorized access to unattended screen-locked computer systems
2. Recovering clear-text keystrokes when having physical access to the USB dongle of the targeted wireless desktop set, e. g. to gain knowledge of passwords

Simple replay attacks can be performed using a software defined radio without knowing the actual communication protocol (black box). Figure 15 shows a test setup for simple replay attacks using a software defined radio.

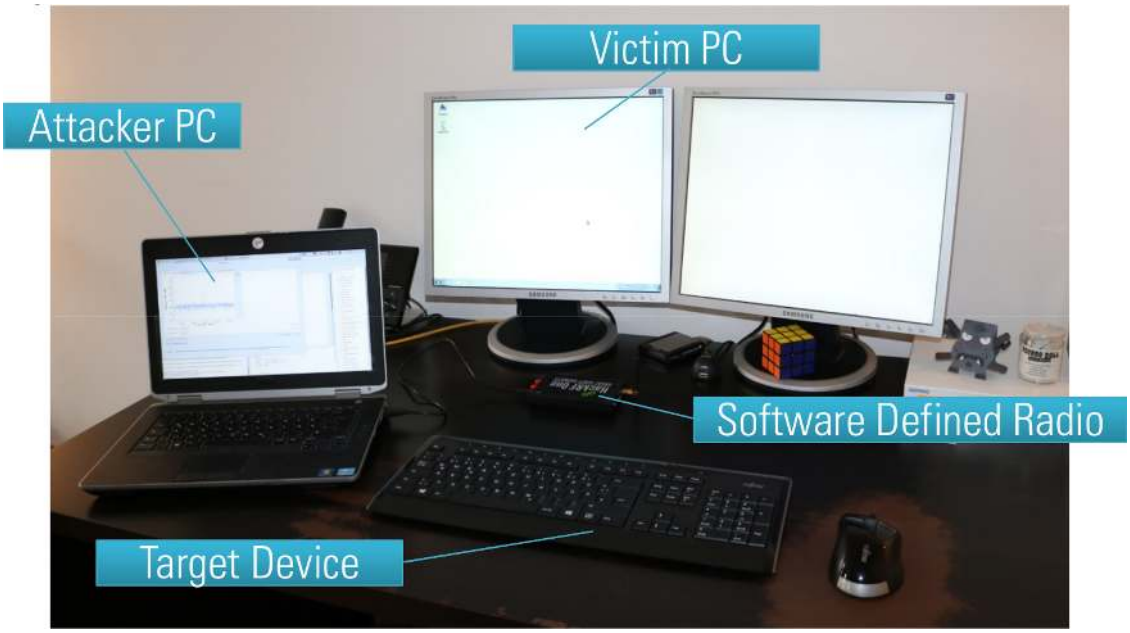

Figure 15: Test setup for simple replay attacks via software defined radio

Figure 16 exemplarily shows the simple GNU Radio Companion flow graph that we used to perform replay attacks against the tested Fujitsu wireless desktop set LX901.

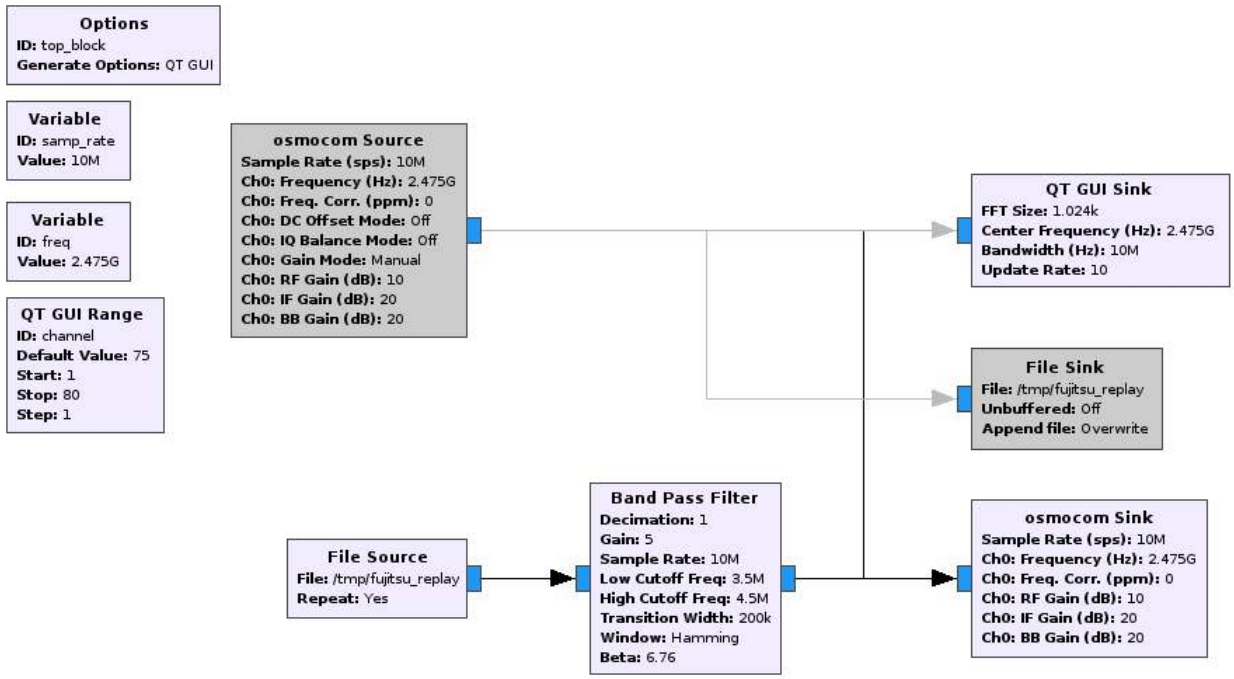

Figure 16: Simple GNU Radio Companion flow graph for replay attacks using a software defined radio

More sophisticated replay attacks, where only specific data packets are of interest and shall be used, can be easily performed using a simple software tool in combination with Crazyradio PA and Bastille's nrf-research-firmware.

During our research project, we have developed a Python software tool named Simple Replay Tool that we used to demonstrate missing or insufficient replay protections in the five tested devices. The following output illustrates a replay attack against a Microsoft Wireless 2000 keyboard.

```
# python simple_replay.py -c 25
        _____  ______ ___  _  _     _____  _                      _
       |  __ \|  ____|__ \| || |   |  __ \| |                    | |
  _ __ | |__) | |__     ) | || |_  | |__) | | __ _ _   _ ___  ___| |_
 | '_ \|  _  /|  __|   / /|__   _| |  ___/| |/ _` | | | / __|/ _ \ __|
 | | | | | \ \| |     / /_   | |   | |    | | (_| | |_| \__ \  __/ |_
 |_| |_|_|  \_\_|    |____|  |_|   |_|    |_|\__,_|\__, |___/\___|\__|
                                                    __/ |
                                                   |___/
Simple Replay Tool v0.2 by Matthias Deeg - SySS GmbH (c) 2016
[*] Configure nRF24 radio
[*] Scanning for wireless keyboard ...
[+] Received data: 083816010100f32a
[+] Found nRF24 device with address A9:A9:8F:EB:CD on channel 25
[?] Attack this device (y/n)? y
[*] Start recording (<CTRL+C> to stop recording)
[+] Received data: 09981601dea2f3157ec032fcfa34ce70dee330c9
[+] Received data: 09981601dea2f3157ec032fcfa34ce70dee330c9…
()
^C
[*] Stop recording
[*] Press <ENTER> to replay the recorded data packets or <CTRL+C> to quit ...
[+] Send data: 09981601dea2f3157ec032fcfa34ce70dee330c9
```

```
[+] Send data: 09981601dea2f3157ec032fcfa34ce70dee330c9…
()
```

By using this proof-of-concept software tool, we could successfully perform the two attacks mentioned at the beginning of this section. So, we were both able to gain unauthorized access to a screen-locked computer system and to recover clear-text keystrokes, for instance, password information, when having physical access to the USB dongle and recorded radio data communication of the corresponding keyboard.

5.4 Keystroke Injection Attacks

The last but not least security vulnerabilities we found during our research were cryptographic issues in three of the five tested wireless desktop sets that allow for keystroke injection attacks.

The tested Cherry, Perixx, and Logitech keyboards use AES with 128 bit keys in counter mode (AES-128-CTR). In general, the initialization vector (IV) consists of a nonce and a counter. The nonce of the tested Cherry keyboard, for example, consists of eleven NULL bytes (sequence of eleven bytes with the value 0x00) and the counter of a random five byte value.

By manipulating the firmware of the Cherry keyboard via SPI access (see Section 5.1), we were able to analyze and better understand the AES-encrypted radio communication that we had previously seen in our radio-based analysis. For analyzing the firmware, we also had a look at the nRF24 SDK and the provided source code, which was very valuable.

In the file `lib_crypt.h`, for instance, we found a very interesting comment about differences in the implementation of the cryptographic library between the nRF24LU1 and the nRF24LE1 device and about the used counter (LS5B) in the nRF24LE1 that is shown in Listing 5.1.

```
)
 /*
(...)
 * @brief Example implementation for encrypting/decrypting data
 *
 * The encryption is based on AES counter mode (CTR) where a 128 bit hybrid counter
 * is used for encryption/decryption. The counter is split in two, 11 bytes as MS11B
 * and 5 bytes as LS5B. The LS5B part is not secret and tells the receiver how
 * to decrypt an encrypted message.
(...)
 * Note that the security of the link will not be reduced as a consequence of sending
 * the counter value in plain text as long as the following criteria are met:
 *
 * - Cipher key used for encryption/decryption must be kept secret.
 * - The plain text counter (LS5B) must be modified for each transfer.
(...)
 * The library can be used on both nRF24LU1 and nRF24LE1 devices, but the implementation
 * is slightly different between these. In the nRF24LE1 implementation the LS5B is not
 * a counter, but random values generated by the embedded random number generator.
 * The reason for this is that the counter value would have to be stored in data memory
 * in between each packet, which is not possible when residing in "deep sleep" power save
 * mode.
(...)
*/
```

Listing 5.1: Source code excerpt from nRF24 SDK (`lib_crypt.h`)

Besides, we could also read in the SDK source code, how the random five byte counter value (LS5B) was actually generated, as Listing 5.2 illustrates.

```
(...)
void lib_crypt_generate_ls5b(uint8_t * dest_buf)
{
  uint8_t i;
  hal_rng_power_up(true);

  for(i=0;i<5;i++)
    {
    while(!hal_rng_data_ready())
    {}
    dest_buf[i] = hal_rng_read();
    }

  hal_rng_power_up(false);
}
(...)
```

Listing 5.2: Source code excerpt from nRF24 SDK (`lib_crypt_le1.c`)

We also found out that the plaintext of a key release packet only consists of NULL bytes, for example, eleven NULL bytes in case of the tested Cherry keyboard.

When considering the operation mode of the used AES counter mode encryption, an attacker knows all the values encircled in red, as depicted in Figure 17, concerning a key release packet.

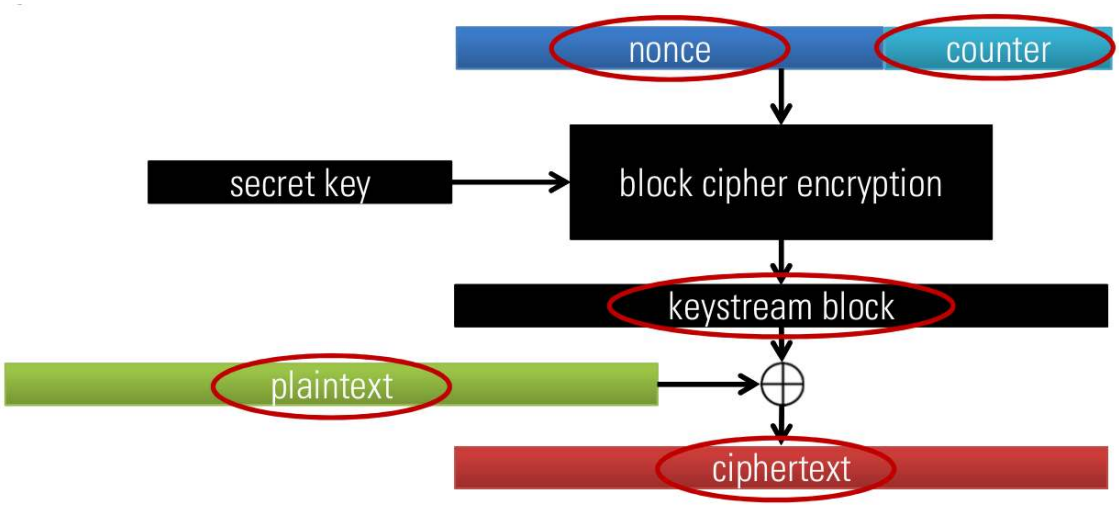

Figure 17: Counter mode encryption

Regarding the Cherry AES B.Unlimited, the known values are as follows:

1. The nonce is hard-coded to 11 NULL bytes.
2. The counter is a random five byte value that can be reused (see replay attacks in Section 5.3) and is sent in cleartext over the air via ShockBurst radio communication.
3. The ciphertext of a key release packet is sent over the air via ShockBurst radio communication.
4. The plaintext of a key release packet is 11 NULL bytes.
5. The keystream block of a key release packet with a specific IV (nonce + counter) is 11 NULL bytes (content of the key release packet), as $x \oplus 0 = x$ (exclusive or).

Thus, a key release packet can be arbitrarily manipulated by an attacker and used for keystroke injection attacks.

For this, an attacker has to know the actual data format for sending keystrokes. By having access to the firmware and analyzing the actual code or by trial and error, it is possible to find out what the used data format is and which bits and bytes have to modified for successful keystroke injections.

The tested Cherry keyboard AES B.Unlimited, for instance, uses the USB HID data format shown in Figure 18 for the actual keystroke data.

Figure 18: USB HID data format for sending keystrokes used by Cherry keyboard

Examples of valid modifiers and key codes for this USB HID data format are:

Modifier/Key	Modifier/Key Code
MODIFIER_NONE	0
MODIFIER_SHIFT_LEFT	1 « 1
MODIFIER_ALT_LEFT	1 « 2
KEY_A	0x04
KEY_B	0x05
KEY_C	0x06

Table 4: Examples of modifier and key codes

An example for a keystroke injection with the capital letter A is illustrated in Figure 19.

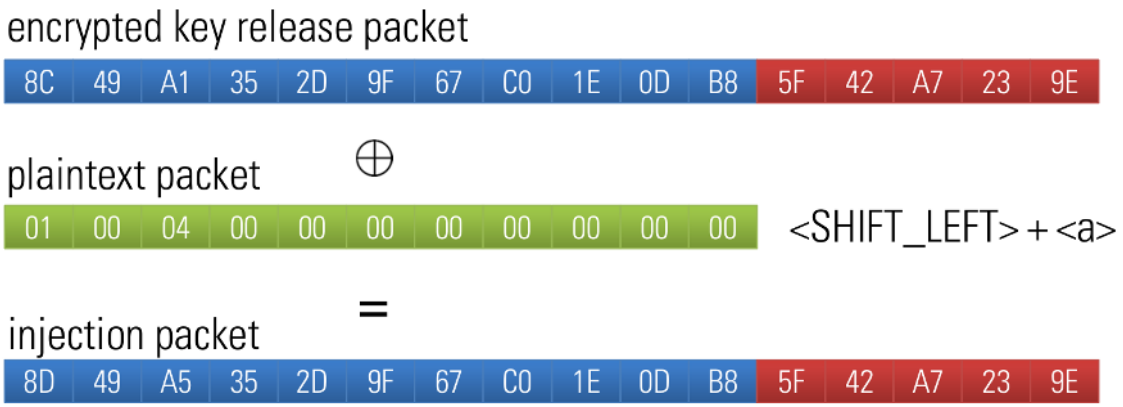

Figure 19: Example of keystroke injection

In general, a keystroke injection attack can be performed in these five easy steps:

1. Find target device (wireless keyboard)
2. Find key release packet (heuristic method)

3. Do simple math (XOR)
4. Send modified key release packet for keystroke injection
5. Repeat steps 3 & 4 until attack is completed

In the course of our research project, we developed several proof-of-concept software tools that allow an easy performance of keystroke injection attacks against affected devices.

The following output exemplarily shows our software tool Keystroke Injector used against a Perixx keyboard.

```
# python keystroke_injector.py -c 6 11 28 32 39 65 75 80 -d perixx
        _____  ______ ___  _  _     _____  _                      _
       |  __ \|  ____|__ \| || |   |  __ \| |                    | |
  _ __ | |__) | |__     ) | || |_  | |__) | | __ _ _   _ ___  ___| |_
 | '_ \|  _  /|  __|   / /|__   _| |  ___/| |/ _` | | | / __|/ _ \ __|
 | | | | | \ \| |     / /_   | |   | |    | | (_| | |_| \__ \  __/ |_
 |_| |_|_|  \_\_|    |____|  |_|   |_|    |_|\__,_|\__, |___/\___|\__|
                                                     __/ |
                                                    |___/
Keystroke Injector v0.7 by Matthias Deeg - SySS GmbH (c) 2016
[*] Configure nRF24 radio
[*] Set keyboard: Perixx
[*] Scanning for wireless keyboard ...
[+] Found nRF24 device with address 15:25:D8:AB:23 on channel 65
[?] Attack this device (y/n)? y
[*] Search for crypto key (actually a key release packet) ...
[+] Found crypto key
[*] Please choose your attack vector (0 to quit)
    1) Open calc.exe
    2) Open cmd.exe
    3) Classic download & execute attack
    0) Exit
[?] Select keystroke injection attack: 3
[*] Start keystroke injection ...
[*] Done.
```

Figure 20 depicts our software tool Cherry Attack PoC that can be used to attack vulnerable Cherry keyboards.

```
[root@hackbox nrf24_playset]# python cherry_attack.py
[2016-10-10 12:38:35.779]  Start Cherry Attack v1.0
[2016-10-10 12:38:40.409]  Found keyboard with address 6B:B7:E2:9E:31
[2016-10-10 12:38:41.456]  Received payload: 88e0f93414916ad7c8ca531dfbd663d3
[2016-10-10 12:38:41.546]  Received payload: 8b97b4c62f2fce74f2021ff90870177a
[2016-10-10 12:38:42.272]  Received payload: 596e29b11353aa645341eb30a24ac78b
[2016-10-10 12:38:42.393]  Received payload: c1b16d5ab68ba9f5211ffbdc54f4e3e2
[2016-10-10 12:38:43.697]  Received payload: e4cf505f1d5d106361f9fcb3fe81636f
[2016-10-10 12:38:43.748]  Received payload: eda153b3b8e35d5ecf8837d2dca1436d
[2016-10-10 12:38:45.748]  Got crypto key!
[2016-10-10 12:38:45.748]  Initialize keyboard
[2016-10-10 12:38:55.217]  Received payload: 428ea391a48dbc1c144065c16d08c424
[2016-10-10 12:38:55.255]  Received payload: c9fed3bcfb2180b71d9e079626ada8c8
[2016-10-10 12:38:55.631]  Received payload: 0d46706d0989ac01f2542477e9e4d553
[2016-10-10 12:38:55.691]  Received payload: 91d07c67ed626a89b5d06730102fea57
[2016-10-10 12:38:55.871]  Received payload: 7883d4eb40984f1cd8ece6ea85528614
[2016-10-10 12:38:55.940]  Received payload: bcc20df7b43ee11bab74bb40e9929acd
[2016-10-10 12:38:56.151]  Received payload: 6f9210a70a74bf2a4419accde790f1f1
[2016-10-10 12:38:56.208]  Received payload: cc4d863733b389db7ccf406e517dd19e
[2016-10-10 12:39:02.632]  Received payload: be15865ba027a73287351e7ccf1d314a
[2016-10-10 12:39:02.690]  Received payload: 5c671e64b5ff27d73731859f10dad4e5
[2016-10-10 12:39:02.752]  Received payload: e9ec98ef38129941643a26b1bbe55500
[2016-10-10 12:39:02.897]  Received payload: 122080c94dfd0beb3f0fe33e1b2dec19
[2016-10-10 12:39:02.975]  Received payload: ab95951dad0495919aa4e6893d5deb64
[2016-10-10 12:39:03.028]  Received payload: 01a0bf262707945c2e43d4f5b79b3a78
[2016-10-10 12:39:03.074]  Received payload: b197e871a45a2f1629bb89e5a04cf60d
[2016-10-10 12:39:03.124]  Received payload: f42720acfffdf0f52ba4da2d02af0fd9
[2016-10-10 12:39:03.193]  Received payload: ac230d666e6312eff27ea1df1ac2b675
[2016-10-10 12:39:03.301]  Received payload: 05bb4b188bfc8a423028a52a3c4327fb
[2016-10-10 12:39:03.353]  Received payload: edddb970bb1eec7444a8672158d98084
[2016-10-10 12:39:03.473]  Received payload: 8967ea565c8195e24729d10615b86dc9
[2016-10-10 12:39:03.504]  Received payload: 9155046f0c85bd599c1c2b8a73b9b844
```

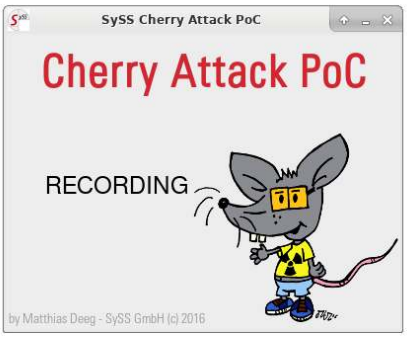

Figure 20: SySS Cherry Attack PoC software tool

The source code excerpt shown in Listing 5.3 illustrates how simple the actual attack exploiting the described cryptographic issues of the used AES counter mode is.

```
def keyCommand(self, modifiers, keycode1, keycode2 = KEY_NONE, keycode3 = KEY_NONE, keycode4 = KEY_NONE, keycode5 = KEY_NONE, keycode6 = KEY_NONE):
    """Return AES encrypted keyboard data"""

    # generate HID keyboard data
    plaintext = pack("8B", modifiers, 0, keycode1, keycode2, keycode3,
    keycode4, keycode5, keycode6)

    # encrypt the data with the set crypto key
    ciphertext = ""
    i = 0
    for b in plaintext:
        ciphertext += chr(ord(b) ^ ord(self.cryptoKey[i]))
        i += 1

    return ciphertext + self.counter
```

Listing 5.3: Source code excerpt from SySS Cherry PoC tool (`keyboard.py`)

For demonstration purposes of the found replay and keystroke injection vulnerabilities, we also built a simple device called SySS Radio Hack Box [32] whose first prototype is shown in Figure 21.

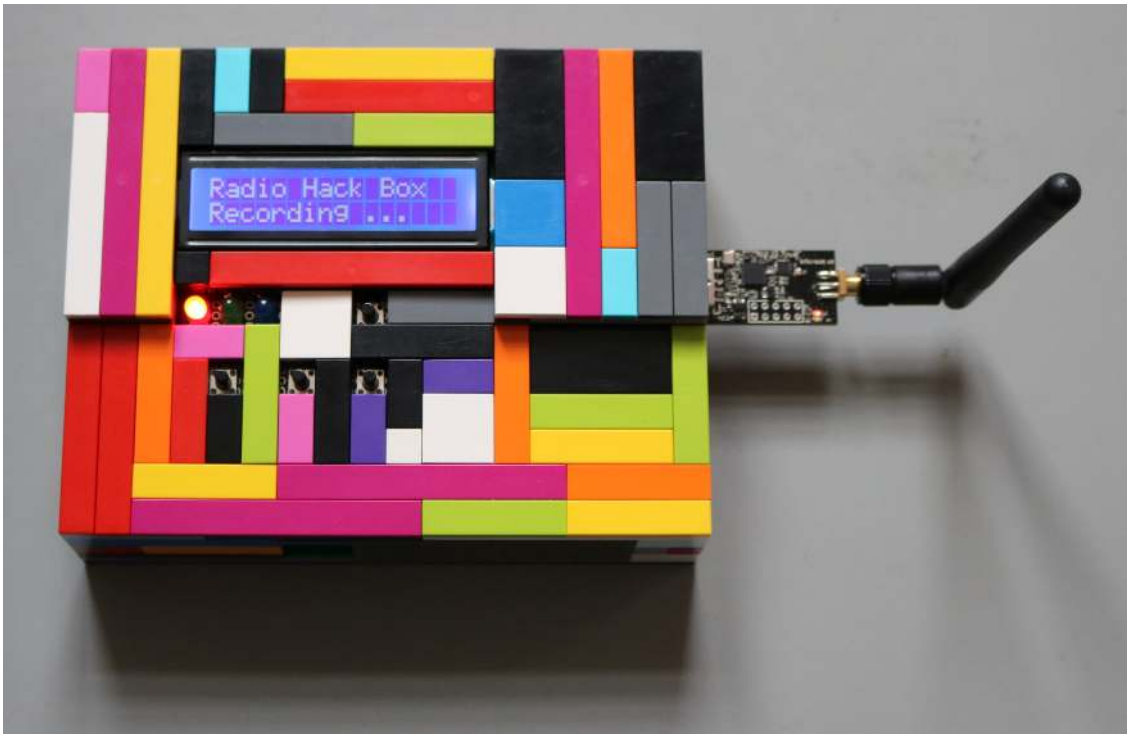

Figure 21: SySS Radio Hack Box prototype

It is a simple Raspberry Pi-based device with a very simple hand-soldered Raspberry Pi shield that is shown in Figure 22 using a Crazyradio PA with a custom Python software.

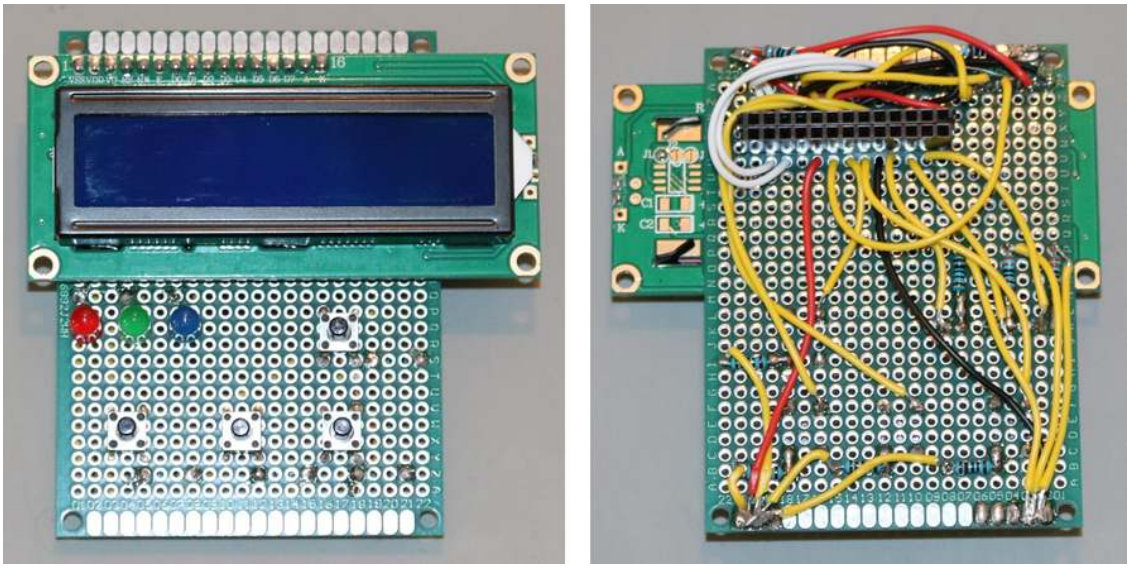

Figure 22: Very simple Raspberry Pi shield

A demo video showing a replay and keystroke injection attack using the SySS Radio Hack Box is available on YouTube [33].

6 Manufacturer Statements and Feedback

In the course of our responsible disclosure program, we reported all found security vulnerabilities of this research project via security advisories to all manufacturers of affected devices.

At a later date, all security advisories were made publicly available according to our responsible disclosure policy [34].

In the following sections, received manufacturer statements and feedback are quoted.

6.1 Perixx

There was no response from Perixx concerning our reported security issues in the tested wireless desktop set Perixx PERIDUO-710W.

6.2 Microsoft

Concerning the insufficient protection of code and data found in the wireless desktop set Microsoft Wireless Desktop 2000, we received the following feedback from Microsoft:

> *As you called out in your email, given each wireless desktop set has different cryptographic key which makes this attack not generic at all. It also requires physical access to the keyboard and sniffer to capture packets to decrypt with obtained key. If you can open keyboard and dump flash from it you can as well change the whole board. Hence, this doesn't meet security servicing bugbar. We have opened a bug in the next version of the product for the core team to evaluate.*

Regarding mouse spoofing attacks, we received the following statement from Microsoft:

> *This behavior is by design and that there will be no security update.*

And concerning replay attacks, Microsoft sent us the following feedback:

> *We are verifying our fix for this issue, hopefully it will be the necessary solution. In that it lies in the dongle firmware, we are still coming to an understanding on whether this will be go-forward only for keyboard dongles or whether there are options for making the fix available for already manufactured dongles.*

6.3 Logitech

We received the following feedback from Logitech concerning reported security issues in the tested wireless desktop set Logitech MK520:

> *Please thank them a lot of their notification and let them know that Logitech is working to provide a better encryption for future products.*

6.4 Fujitsu

Fujitsu sent us the following statement concerning the reported replay attack found in the wireless desktop set Fujitsu Wireless Keyboard Set LX901:

> *Thank you very much for your information about our wireless keyboard. As we have already pointed out, we believe that the described scenario is not easy to perform under real conditions due to the radio protocol used. As mentioned, our product is not destined to sell security, but convenience in the first place (without the security drawbacks of unencrypted wireless keyboards). Any new information and insights will be incorporated into the already planned successor product.*

6.5 Cherry

Regarding the reported security vulnerabilities we had found in the tested wireless desktop set Cherry AES B.Unlimited, we received the following statement from Cherry:

> *We have examined the 'security flaws' you reported to us. As a result, we decided, until further notice, to no longer refer to AES encryption in order to promote the affected product. At the moment, we are currently working on a successor product. As we already did in the past, we recommend to our customers having particularly high security demands using wired products which, depending on the requirements, should be CC certified.*

7 Conclusion

In the course of our research project, we could find one or more security vulnerabilities in all five tested modern wireless desktop sets with AES encryption.

Overall, the following five security issues were found that allow for specific attacks:

1. Insufficient protection of code (firmware) and data (cryptographic key)
   ⇒Access to sensitive data
2. Unencrypted and unauthenticated data communication
   ⇒Mouse spoofing attacks
3. Missing protection against replay attacks
   ⇒Replay attacks
4. Insufficient protection against replay attacks
   ⇒Replay attacks
5. Cryptographic issues
   ⇒Keystroke injection attacks

Table 6 summarizes the results of our research project.

Product Name	Insufficient Code/ Data Protection	Mouse Spoofing	Replay	Keystroke Injection
Cherry AES B.UNLIMITED	✓	✓	✓	✓
Fujitsu Wireless Keyboard Set LX901	?	?	✓	?
Logitech MK520	X	✓	✓	✓*
Microsoft Wireless Desktop 2000	✓	✓	✓	?
Perixx PERIDUO-710W	✓	✓	✓	✓

Table 6: Summary of our research results

✓ security issue exists

X security issue does not exist

? security issue may exist (more work required)

* first found reported to Logitech by Bastille Networks

In our opinion and experience, all of those found security vulnerabilities can be exploited in real world attack scenarios and are not only exploitable in a lab environment. As a matter of fact, we already have exploited some of the security vulnerabilities during penetration tests within the last couple of months.

Some of the found security vulnerabilities cannot or will not be fixed in the tested product versions, but maybe in future ones.

The results of our research show that the security levels of modern wireless desktop sets of different manufacturers are not equal and that some devices are more secure than others. Yet, there was no wireless desktop set without any security issues.

SySS recommends that wireless desktop sets with known security vulnerabilities not be used in security-related environments.

During our research project, the Bastille Threat Research Team independently also found security vulnerabilities in several wireless desktop sets with AES encryption of different manufacturers. You can find the results of their research project named KeyJack on their website [35].

In 2016, we presented the results of our research project at different IT security conferences. For instance, slides and a video recording of our talk *Of Mice and Keyboards: On the Security of Modern Wireless Desktop Sets* from the IT security conference Hack.lu 2016 can be found at [36, 37].

References

[1] Dreamlab Technologies, KeyKeriki v2.0 – 2.4 GHz, http://www.remote-exploit.org/articles/keykeriki_v2_0__8211_2_4ghz/, 2010 (Cited on page 1.)

[2] Samy Kamkar, KeySweeper, http://samy.pl/keysweeper, 2015 (Cited on page 1.)

[3] Travis Goodspeed, Promiscuity is the nRF24L01+'s Duty, http://travisgoodspeed.blogspot.de/2011/02/promiscuity-is-nrf24l01s-duty.html, 2011 (Cited on page 1.)

[4] Bastille Networks Internet Security, MouseJack, https://www.mousejack.com/, 2016 (Cited on page 1.)

[5] Great Scott Gadgets, HackRF One, https://greatscottgadgets.com/hackrf/ (Cited on page 2.)

[6] Ettus Research, USRP B200, https://www.ettus.com/product/details/UB200-KIT (Cited on page 2.)

[7] Bitcraze, Crazyradio PA, https://www.bitcraze.io/crazyradio-pa/ (Cited on page 2.)

[8] Bastille Networks Internet Security, nrf-research-firmware, https://github.com/BastilleResearch/nrf-research-firmware, 2016 (Cited on page 3.)

[9] Dangerous Prototypes, Bus Pirate, http://dangerousprototypes.com/docs/Bus_Pirate, (Cited on page 5.)

[10] Dangerous Prototypes, nrfprog, https://github.com/JoseJX/nrfprog, (Cited on page 5.)

[11] Hex-Rays, Interactive Disassembler Pro, https://www.hex-rays.com/products/ida/index.shtml (Cited on page 7.)

[12] Nordic Semiconductor, Nordic Semiconductor nRF24LE Product Specification v1.6, http://www.nordicsemi.com/eng/content/download/2443/29442/file/nRF24LE1_Product_Specification_rev1_6.pdf (Cited on pages 4, 5, 7, and 12.)

[13] Nordic Semiconductor, nRF24L01+ Evaluation Kit, https://www.nordicsemi.com/eng/nordic/download_resource/9620/7/85171947/2434 (Cited on page 7.)

[14] GNU Radio https://en.wikipedia.org/wiki/GNU_Radio (Cited on page 9.)

[15] Omri Iluz, NRF24-BTLE-Decoder, https://github.com/omriiluz/NRF24-BTLE-Decoder, 2016 (Cited on page 9.)

[16] Gerhard Klostermeier and Matthias Deeg, SySS Security Advisory SYSS-2016-031, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-031.txt, 2016 (Cited on page 11.)

[17] Gerhard Klostermeier and Matthias Deeg, SySS Security Advisory SYSS-2016-032, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-032.txt, 2016 (Cited on page 11.)

[18] Gerhard Klostermeier and Matthias Deeg, SySS Security Advisory SYSS-2016-033, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-033.txt, 2016 (Cited on page 11.)

[19] Matthias Deeg and Gerhard Klostermeier SySS Security Advisory SYSS-2016-038, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-038.txt, 2016 (Cited on page 11.)

[20] Matthias Deeg and Gerhard Klostermeier SySS Security Advisory SYSS-2016-043, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-043.txt, 2016 (Cited on page 11.)

[21] Gerhard Klostermeier and Matthias Deeg, SySS Security Advisory SYSS-2016-044, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-044.txt, 2016 (Cited on page 11.)

[22] Gerhard Klostermeier and Matthias Deeg, SySS Security Advisory SYSS-2016-045, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-045.txt, 2016 (Cited on page 11.)

[23] Matthias Deeg and Gerhard Klostermeier, SySS Security Advisory SYSS-2016-046, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-046.txt, 2016 (Cited on page 11.)

[24] Matthias Deeg and Gerhard Klostermeier, SySS Security Advisory SYSS-2016-047, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-047.txt, 2016 (Cited on page 11.)

[25] Matthias Deeg and Gerhard Klostermeier, SySS Security Advisory SYSS-2016-058, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-058.txt, 2016 (Cited on page 11.)

[26] Matthias Deeg and Gerhard Klostermeier, SySS Security Advisory SYSS-2016-059, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-059.txt, 2016 (Cited on page 11.)

[27] Gerhard Klostermeier and Matthias Deeg, SySS Security Advisory SYSS-2016-060, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-060.txt, 2016 (Cited on page 11.)

[28] Gerhard Klostermeier and Matthias Deeg, SySS Security Advisory SYSS-2016-061, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-061.txt, 2016 (Cited on page 11.)

[29] Matthias Deeg and Gerhard Klostermeier, SySS Security Advisory SYSS-2016-068, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-068.txt, 2016 (Cited on page 11.)

[30] SySS GmbH, Radioactive Mouse States the Obvious – Proof-of-Concept Video, https://www.youtube.com/watch?v=PkR8EODee44, 2016 (Cited on page 15.)

[31] Matthias Deeg and Gerhard Klostermeier, nRF24 Playset, https://github.com/SySS-Research/nrf24-playset, 2017 (Cited on page 13.)

[32] Matthias Deeg and Gerhard Klostermeier, SySS Radio Hack Box, https://github.com/SySS-Research/radio-hackbox, 2017 (Cited on page 21.)

[33] SySS GmbH, SySS Radio Hack Box a.k.a. SySS Cherry Picker Demo Video, https://www.youtube.com/watch?v=KMlmd-LhMmo, 2016 (Cited on page 22.)

[34] SySS GmbH, SySS Responsible Disclosure Policy, https://www.syss.de/en/responsible-disclosure-policy/ (Cited on page 23.)

[35] Bastille Networks Internet Security, KeyJack, https://www.bastille.net/research/vulnerabilities/keyjack/keyjack-intro/, 2016 (Cited on page 27.)

[36] Matthias Deeg and Gerhard Klostermeier, Of Mice and Keybords (Slides), Hack.lu 2016, http://archive.hack.lu/2016/Of_Mice_and_Keyboards-Hack.lu_2016.pdf, 2016 (Cited on page 27.)

[37] Matthias Deeg and Gerhard Klostermeier, Of Mice and Keybords (Talk), Hack.lu 2016, https://www.youtube.com/watch?v=Ja_VgUMz43Q, 2016 (Cited on page 27.)

THE PENTEST EXPERTS

SySS GmbH 72072 Tübingen Germany +49 (0)7071 - 40 78 56-0 info@syss.de

WWW.SYSS.DE