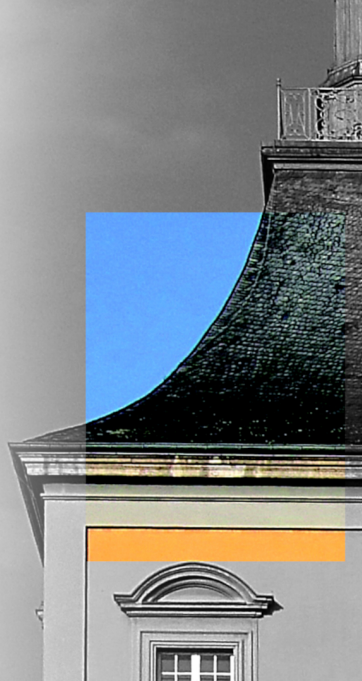
UNIVERSITÄT BONN

# IT Security

## Malware Analysis

Timo Pohl
pohl@cs.uni-bonn.de
University of Bonn | Institute of Computer Science 4





UNIVERSITÄT BONN

# Contents

UNIVERSITÄT BONN

# Motivation

- There is a full module on malware analysis
- This is a speedrun through some interesting topics
- If you're interested, visit the module

UNIVERSITÄT BONN

# Why Malware Analysis?

## Cyber Attack: Targobank Blocks access to thousands of accounts

After attempted access by malicious players to online banking, Targobank blocked the access of thousands of customers. New access data is created

Reading time: **2 min.** Save in Pocket

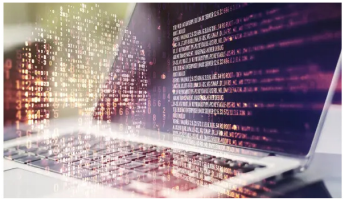
13.11.2023 1

*By Dirk Knop*

## After ransomware attack: South Westphalia IT and municipalities refuse ransom payment

The municipal IT service provider Südwestfalen-IT rejects the payment of a ransom money after a ransomware attack in consultation with the affected municipalities.

Reading time: **2 min.** Save in Pocket



(Image: Pixels Hunter/Shutterstock.com)

11.11.2023 17:45 Watch

## DP World: Safety incident paralyses key port operators in Australia

A cybersecurity incident has paralysed the port operator DP World Australia. The government considers the situation to be "very serious".

Reading time: **2 min.** Save in Pocket

## Dena: Cyber attack on the German Energy Agency

The Dena has been incapacitated since Sunday afternoon – due to a cyber attack, as the Bund-owned GmbH announced on Tuesday afternoon.

Reading time: **2 min.** Save in Pocket


(Image: Dena)

UNIVERSITÄT BONN

# Why Malware Analysis?

- Find out how malware works
- Stop ongoing campaigns
- Develop countermeasures

UNIVERSITÄT BONN

# Getting Infected

- 💪 Motivation
- 🦠 Getting Infected
  - PDF Analysis
- 🔎 PE Analysis
  - Obfuscation
  - Anti-Debugging
  - Anti-Sandboxing
- 🧑‍🎓 Outlook

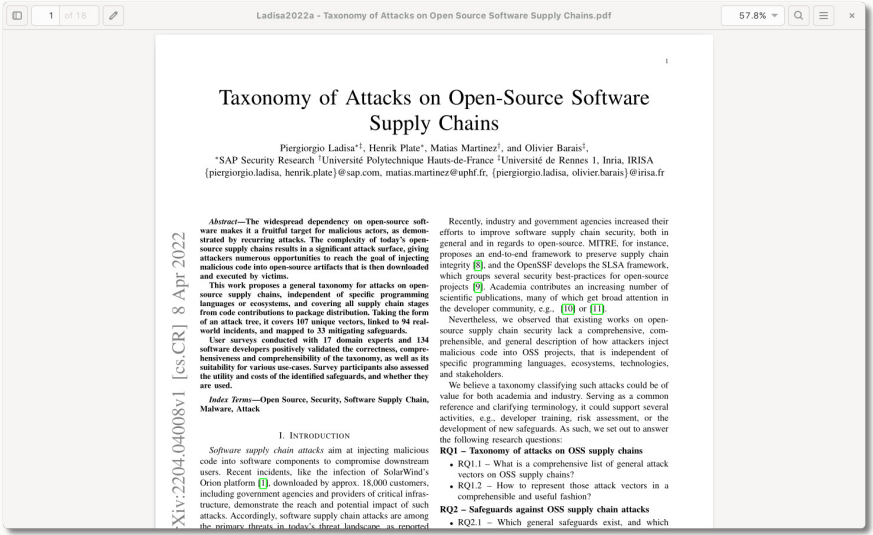
UNIVERSITÄT BONN

# Getting infected

- PDFs
- Phishing
- Malicious updates
- Office macros
- …

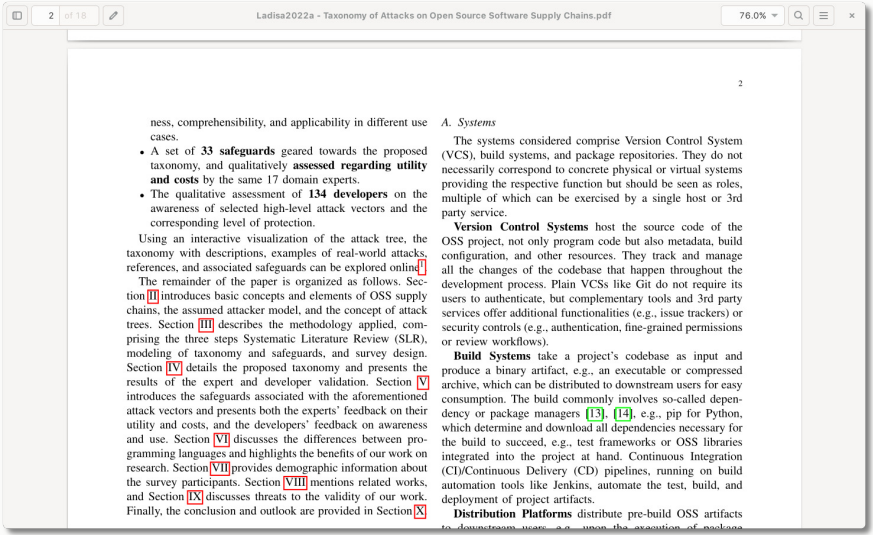
UNIVERSITÄT BONN

# PDFs show Documents!



UNIVERSITÄT BONN

# PDFs are interactive.




ness, comprehensibility, and applicability in different use cases.

- A set of **33 safeguards** geared towards the proposed taxonomy, and qualitatively **assessed regarding utility and costs** by the same 17 domain experts.
- The qualitative assessment of **134 developers** on the awareness of selected high-level attack vectors and the corresponding level of protection.

Using an interactive visualization of the attack tree, the taxonomy with descriptions, examples of real-world attacks, references, and associated safeguards can be explored online[1].

The remainder of the paper is organized as follows. Section II introduces basic concepts and elements of OSS supply chains, the assumed attacker model, and the concept of attack trees. Section III describes the methodology applied, comprising the three steps Systematic Literature Review (SLR), modeling of taxonomy and safeguards, and survey design. Section IV details the proposed taxonomy and presents the results of the expert and developer validation. Section V introduces the safeguards associated with the aforementioned attack vectors and presents both the experts' feedback on their utility and costs, and the developers' feedback on awareness and use. Section VI discusses the differences between programming languages and highlights the benefits of our work on research. Section VII provides demographic information about the survey participants. Section VIII mentions related works, and Section IX discusses threats to the validity of our work. Finally, the conclusion and outlook are provided in Section X.

*A. Systems*

The systems considered comprise Version Control System (VCS), build systems, and package repositories. They do not necessarily correspond to concrete physical or virtual systems providing the respective function but should be seen as roles, multiple of which can be exercised by a single host or 3rd party service.

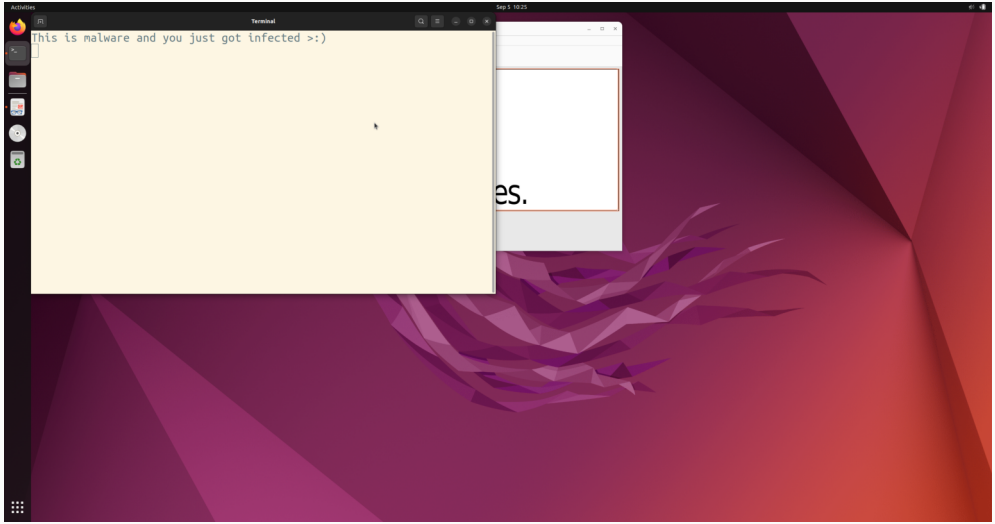
**Version Control Systems** host the source code of the OSS project, not only program code but also metadata, build configuration, and other resources. They track and manage all the changes of the codebase that happen throughout the development process. Plain VCSs like Git do not require its users to authenticate, but complementary tools and 3rd party services offer additional functionalities (e.g., issue trackers) or security controls (e.g., authentication, fine-grained permissions or review workflows).

**Build Systems** take a project's codebase as input and produce a binary artifact, e.g., an executable or compressed archive, which can be distributed to downstream users for easy consumption. The build commonly involves so-called dependency or package managers [13], [14], e.g., pip for Python, which determine and download all dependencies necessary for the build to succeed, e.g., test frameworks or OSS libraries integrated into the project at hand. Continuous Integration (CI)/Continuous Delivery (CD) pipelines, running on build automation tools like Jenkins, automate the test, build, and deployment of project artifacts.

**Distribution Platforms** distribute pre-build OSS artifacts to downstream users, e.g., upon the execution of package

UNIVERSITÄT BONN

# PDFs… run code?



UNIVERSITÄT BONN

# PDFs are (mostly) readable

```
%PDF-1.5
1 0 obj
<</Type/Pages/Kids[8 0 R]/Count 1/Resources 3 0 R/MediaBox[0 0 700 200]>>
endobj
2 0 obj
<</Type/Font/Subtype/Type1/BaseFont/Noto#20Sans>>
endobj
3 0 obj
<</Font<</F1 2 0 R>>>>
endobj
4 0 obj
<</Length 60>>stream
```

UNIVERSITÄT BONN

# PDF Structure

- Large collection of objects
- Mostly readable
- Direct objects like 1, `(hello)` or `[null true false]`
- Indirect objects starting with `<ID> <Gen-No> obj` and ending with `endobj`

UNIVERSITÄT BONN

# PDF Structure Example

```
%PDF-1.5
1 0 obj
<</Type/Pages/Kids[8 0 R]/Count 1/Resources 3 0 R/MediaBox[0 0 700 200]>>
endobj
2 0 obj
<</Type/Font/Subtype/Type1/BaseFont/Noto#20Sans>>
endobj
3 0 obj
<</Font<</F1 2 0 R>>>>
endobj
4 0 obj
<</Length 60>>stream
```

UNIVERSITÄT BONN

# Common Object Types

- Name objects — `/Name`, `/anything`
- Reference objects — `8 0 R`
- Strings — `(as literals)` or `<68656c6c6f>` hex encoded
- Numbers — `0` or `123994`
- Arrays — `[(hello) /world]`
- Dictionaries — `<</Key1 /AnyObj /Key2 8 0 R>>`

UNIVERSITÄT BONN

# PDF Types Example

```
3  <<
4      /Type /Pages
5      /Kids [8 0 R]
6      /Count 1
7      /Resources 3 0 R
8      /MediaBox [0 0 700 200]
9      /MadeUpKey ("String literal in here!")
10 >>
```

UNIVERSITÄT BONN

# Start of a PDF

```
trailer
<<
    /Size 713
    /Root 711 0 R
    /Info 712 0 R
>>
startxref
281353
%%EOF
```

<- Byte offset of "start" dict

```
10 0 obj
<<
    /Root 9 0 R
    /Type /XRef
    /Size 11
    /W [1 4 2]
    /Index [1 10]
    /Length 70
>>
endobj

startxref
816
%%EOF
```

# PDF Interactive Features

UNIVERSITÄT BONN

# PDF Action Dictionaries

```
6 0 obj
<<
    /Type /Action
    /S /Launch
    /F (/tmp/malware.py)
>>
endobj
7 0 obj
<<
    /Type /Action
    /S /JavaScript
    /JS (this.exportAsText(true, '/tmp/file.txt');)
>>
```

UNIVERSITÄT BONN

# PDF Interesting Actions

- `/Launch`
- `/JavaScript`
- `/SubmitForm`
- `/URI`

Launch an application. Execute JavaScript. Send data to a specified URL. Open a URI.

UNIVERSITÄT BONN

# PDF OpenAction

```
1 9 0 obj
2 <<
3       /Type /Catalog
4       /Pages 1 0 R
5       /OpenAction 5 0 R
6 >>
```

UNIVERSITÄT BONN

# PDF OpenAction

- PDFs have `Catalog` dictionaries
- `Catalog` dictionary has “content catalog”
- `OpenAction` defined in `Catalog` dictionary
- `OpenAction` executed when opening document

UNIVERSITÄT BONN

# PDF Full Example

```
10 0 obj
    <<
        /Root 9 0 R
        /Type /XRef
        /Size 11
        /W [1 4 2]
        /Index [1 10]
        /Length 70
    >>
    stream
        ^A^@^@^@        ^@^@^A^@^@^@b^@^@^A^@^@^@E^@^@^A^@^@^@E^@^@^A^@^@^A5
    endstream
endobj

startxref
754
%%EOF
```

UNIVERSITÄT BONN

# PDF Full Example

- We start at byte offset 754
- Points at object `10 0`
- `Root` dict points at object `9 0`

UNIVERSITÄT BONN

# PDF Full Example

```
50 9 0 obj
51     <<
52         /Type /Catalog
53         /Pages 1 0 R
54     >>
55 endobj
```

UNIVERSITÄT BONN

# PDF Full Example

- Object `9 0` is `Catalog` dictionary

- Has pages object at `1 0`

UNIVERSITÄT BONN

# PDF Full Example

```
%PDF-1.5
1 0 obj
    <<
        /Type /Pages
        /Kids [8 0 R]
        /Count 1
        /Resources 3 0 R
        /MediaBox [0 0 700 200]
    >>
endobj
```

UNIVERSITÄT BONN

# PDF Full Example

- Pages object shows one page object at `8 0`
- `Resources` object `3 0` is irrelevant for us

UNIVERSITÄT BONN

# PDF Full Example

```
33 8 0 obj
34     <<
35         /Type /Page
36         /Parents 1 0 R
37         /Contents 4 0 R
38         /Annots [
39             <<
40                 /Type /Annot
41                 /Subtype /Link
42                 /Open true
43                 /A 6 0 R
44                 /H /N
45                 /Rect [0 0 700 200]
46             >>
47         ]
48     >>
49 endobj
```

UNIVERSITÄT BONN

# PDF Full Example

- `Parents` object is where we come from
- `Contents` object at `4 0` defines page content
- `Annots` object defines annotations
- Annotation defines an action at `6 0`

UNIVERSITÄT BONN

# PDF Full Example

```
4 0 obj
    <<
        /Length 60
    >>
    stream
        BT
        /F1 48 Tf
        0 10 Td
        (Click me for free cat pictures.) Tj
        ET
    endstream
endobj
```

UNIVERSITÄT BONN

# PDF Full Example

- Content object has no relevant information for us

- We see the displayed text in the byte stream

UNIVERSITÄT BONN

# PDF Full Example

```
25 6 0 obj
26 <<
27     /Type /Action
28     /S /Launch
29     /F (/tmp/malware.py)
30 >>
```

UNIVERSITÄT BONN

# PDF Full Example

- Annotation action is indeed an action dict
- Contains a Launch action
- Launches `/tmp/malware.py`

UNIVERSITÄT BONN

# PDFs can be “obfuscated”

- Different string representation
- Stream compression
- Object encryption

```
6 0 obj
<<
    /Type /Action
    /S /Launch
    /F (/tmp/malware.py)
>>
endobj
6 0 obj
<<
    /Type /Action
    /S /Launch
    /F <2F746D702F6D616C776172652E7079>
>>
endobj
```

UNIVERSITÄT BONN

# PDF analysis

- Your favourite text editor
- qpdf for “deobfuscation”
- peepdf for a quick overview

```
┌──(kali㉿kali)-[~/peepdf]
└─$ python2 peepdf.py ~/one-hundred-percent-benign.pdf
Warning: PyV8 is not installed!!
Warning: pylibemu is not installed!!
Warning: Python Imaging Library (PIL) is not installed!!

File: one-hundred-percent-benign.pdf
MD5: a1fb993f95c3fdf012061f7b8a4fc64a
SHA1: 0fe21d975fd03f571b41e9ae1546e9fce24427ea
SHA256: 9e023fb09207ac61a1bb0f389467a9f9d2d1242be461385814628ba1ce8c88e5
Size: 1025 bytes
Version: 1.5
Binary: False
Linearized: False
Encrypted: False
Updates: 0
Objects: 10
Streams: 2
URIs: 1
Comments: 0
Errors: 1

Version 0:
        Catalog: 9
        Info: No
        Objects (10): [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
        Streams (2): [4, 10]
                Xref streams (1): [10]
                Encoded (0): []
        Objects with URIs (1): [5]
        Objects with JS code (1): [7]
        Suspicious elements:
                /OpenAction (1): [9]
                /JavaScript (1): [7]
                /Launch (1): [6]
```

UNIVERSITÄT BONN

# Summary

- PDFs are readable as text
- PDFs can perform actions
- Analysis is relatively simple
- Most mainstream PDF viewers are secure against common attacks

# Questions?

UNIVERSITÄT BONN

# PE Analysis

- 💪 Motivation
- 🦠 Getting Infected
  - PDF Analysis
- 🔎 PE Analysis
  - Obfuscation
  - Anti-Debugging
  - Anti-Sandboxing
- 🧑‍🎓 Outlook

UNIVERSITÄT BONN

# Comparison to ELF Analysis

- ELF analysis known from previous lecture
- Windows malware is most prevalent
- Windows uses PE instead of ELF
- `stdcall` and `Microsoft x64` instead of `cdecl` or `System V`
- x64dbg instead of gdb
- Similar tools for static analysis (ghidra, ida, binja, …) + some for quick overviews (PE Studie, Detect it Easy, …)
- Focus on strings, network access, … — we want to understand behaviour

UNIVERSITÄT BONN

# General approach

- Goal: understand malware behaviour
- Top-Down approach: start at entry point, follow control flow
- Bottom-Up approach: search for interesting things like API calls, strings, encryption routines, … and backtrace from there

UNIVERSITÄT BONN

# Obfuscation

- 💪 Motivation
- 🦠 Getting Infected
  - PDF Analysis
- 🔎 PE Analysis
  - Obfuscation
  - Anti-Debugging
  - Anti-Sandboxing
- 🧑‍🎓 Outlook

UNIVERSITÄT BONN

# What is Obfuscation?

"carlini" from the 27th IOCCC

```
int*d,D[9999],N=20,L=4,n,m,k,a[3],i;char*p,*q,S[2000]="L@X-SGD-HNBBB-AD-VHSG-\
-XNT\x1b[2J\x1b[H",*s=S,*G="r2zZX!.+@KBK^yh.!:%Bud!.+Jyh.!6.BHBhp!6.BHBh!:%Bu\
v{VT!.hBJ6p!042ljn!284b}`!.hR6Dp!.hp!h.T6p!h.p6!2/LilqP72!h.+@QB!~}lqP72/Lil!\
h.+@QBFp!:)0?F]nwf!,82v!.sv{6!.l6!,j<n8!.xN6t!&NvN*!.6hp";/*Stay_on_target.**/
#include/**/<complex.h>/**//*Oh,my_dear_friend.How_I've_missed_you.--C-3PO***/
typedef/**/complex/**/double(c);c(X)[3],P,O;c/**/B(double t){double s=1-t,u;P=
s*s*X[1]            +2        *s*t*        *X+t        *t*X       [2]+O;u=I*P;
return+48*((    s=P)+   48*I   )/(   1<u?   u:   1);}   /*   1977  IOCCC2020*/
#include/**    Do.Or   do_not   .   There_is_   no_try...   --Yoda**/<stdio.h>
void/**/b(    double   t,/***   *   **/double   u){double   s=P=B(t)-B(u);(s=P
*(2*s-P))    <1?m=P=B   ((t+   u)/   2),k   =-   I*P,   m>   -41&&m<39&&9<k&&k
<48?            m+=k/        2*80+        73,S        [m]=              S[m]
-73?k%2?S[m]-94?95:73:S[m]-95?94:73:71:1:(b(t,(t+u)/2),b((t+u)/2,u),0);}/*<oo>
_No.            _I_am_          IOCCC          1977          ***/
#include/*****    your   father..   --DarthVader   **/   <time.h>/****   ****/
int(main)(int    (x),   char**V){;   clock_t(c)=   /*   */clock();;;   for(d=D
;m<26;m++,s    ++)*s>   63?*d++=m%   7*          16-7   *8,*d++=m/   7*25,*d++
=*s-64:0;;    if(V[1])   {;;;FILE   *F   =fopen(V[+1],   "r");for   (d=D,L=N=m
```

UNIVERSITÄT BONN

# Obfuscation techniques

- String obfuscation
- Control flow obfuscation
- Packing
- Bloating
- …

UNIVERSITÄT BONN

# Why obfuscate strings?

- Strings often have sensitive / important info
- Strings are easy to locate

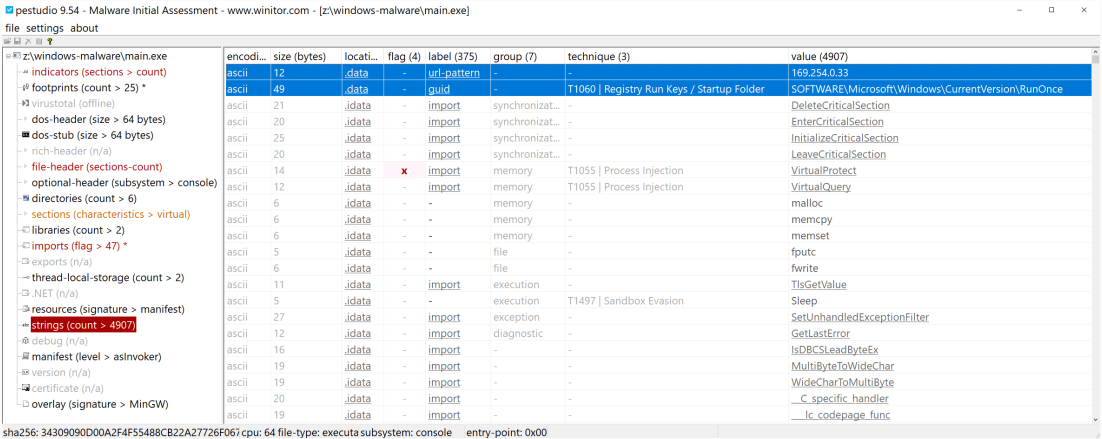
UNIVERSITÄT BONN

# Unobfuscated example

```
1  char ip[] = "169.254.0.33";
2  char registry_key[] = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\
       RunOnce";
3
4  int steal_content_from_registry(char *ip, char *registry_key) {
5    printf("%s: %s\n", ip, registry_key);
6    return 1;
7  }
8
9  int main(void) {
10   steal_content_from_registry(ip, registry_key);
11 }
```

UNIVERSITÄT BONN

# Unobfuscated strings are easy to find



UNIVERSITÄT BONN

# String obfuscation — encryption

```
unsigned char registry_key[] = {161, 189, 180, 166, 165, 179, 160, 183, 174, 191, 155, 145, 128,
                                157, 129, 157, 148, 134, 174, 165, 155, 156, 150, 157, 133, 129,
                                174, 177, 135, 128, 128, 151, 156, 134, 164, 151, 128, 129, 155,
                                157, 156, 174, 160, 135, 156, 189, 156, 145, 151, 0};

void xor_decrypt(unsigned char key, unsigned char *content) {
  for (int i = 0; content[i] != 0; ++i) {
    content[i] = content[i] ^ key;
  }
}

int main(void) {
  unsigned char key = 242;
  xor_decrypt(key, registry_key);
  printf("%s\n", registry_key);
  // prints: SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
}
```

UNIVERSITÄT BONN

# String obfuscation — encryption

- Strings are just arrays of `char`s
- Each `char` can be individually encrypted
- Different encryption techniques possible

UNIVERSITÄT BONN

# String obfuscation — stack strings

```
1  int main(void) {
2    unsigned int ip_0 = 0;
3    unsigned int ip_1 = 0x33332e30;
4    unsigned int ip_2 = 0x2e343532;
5    unsigned int ip_3 = 0x2e393631;
6
7    xor_decrypt(key, registry_key);
8    // int steal_content_from_registry(char *ip, char *registry_key) {};
9    steal_content_from_registry((unsigned char *) &ip_3, registry_key);
10 }
```

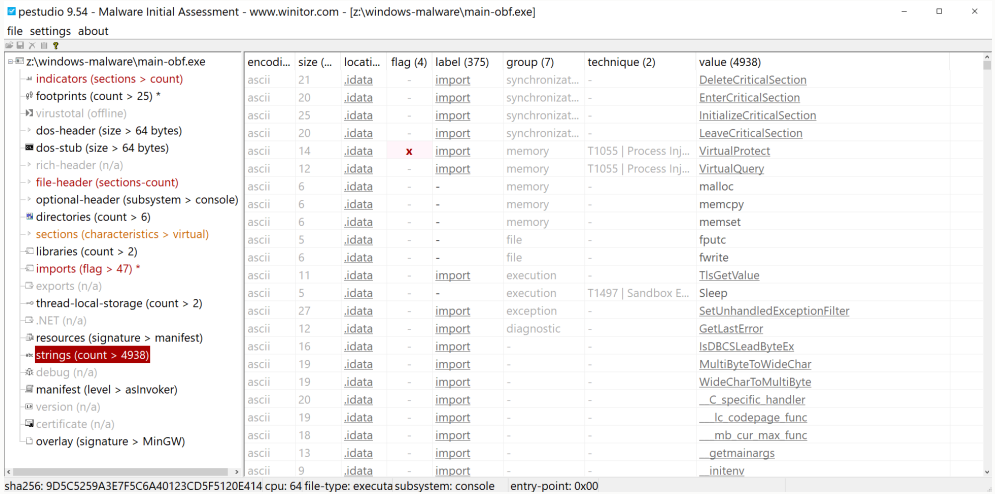
UNIVERSITÄT BONN

# String obfuscation — stack strings

- Strings are just arrays of `char`s
- Arrays are continuous areas of memory
- Stack variables next to each other can be interpreted as an array

UNIVERSITÄT BONN

# Obfuscated strings are hard to find



UNIVERSITÄT BONN

# Deobfuscation

- Static analysis
- Emulation
- Debugger
- ...

UNIVERSITÄT BONN

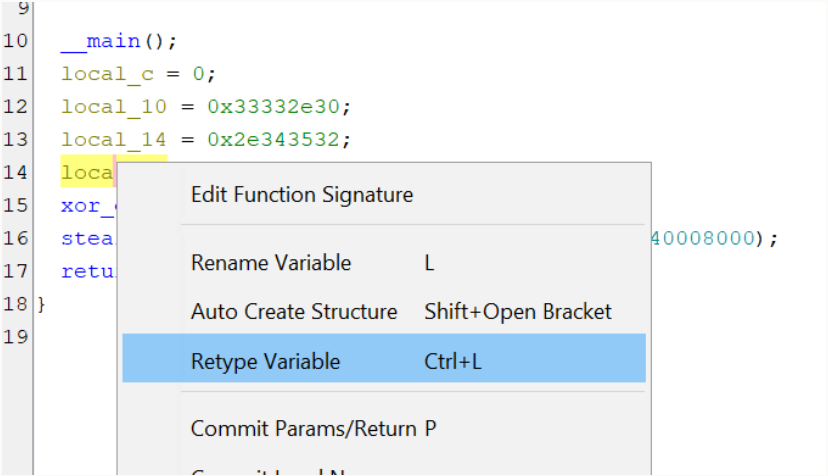
# Deobfuscating stack strings via retyping

```
MOV        dword ptr [RBP + local_c],0x0

MOV        dword ptr [RBP + local_10],0x33332e30

MOV        dword ptr [RBP + local_14],0x2e343532

MOV        dword ptr [RBP + local_18],0x2e393631
```

```
10  __main();
11  local_c = 0;
12  local_10 = 0x33332e30;
13  local_14 = 0x2e343532;
14  local_18 = 0x2e393631;
15  xor_decrypt((byte)key,0x140008000);
16  steal_content_from_registry(&local_18,&DAT_140008000);
17  return 0;
18 }
19
```

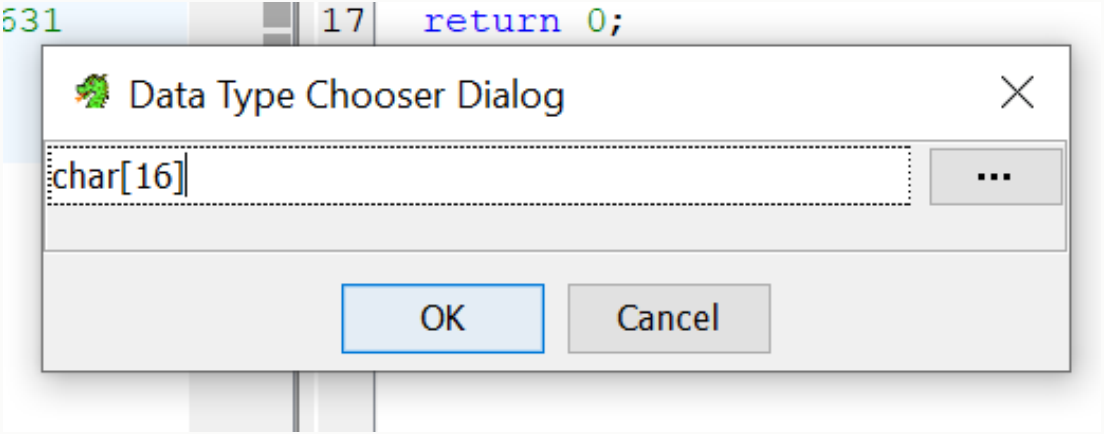
UNIVERSITÄT BONN

# Deobfuscating stack strings via retyping



UNIVERSITÄT BONN

# Deobfuscating stack strings via retyping



UNIVERSITÄT BONN

# Deobfuscating stack strings via retyping

```
7   __main();
8   local_18[12] = '\0';
9   local_18[13] = '\0';
10  local_18[14] = '\0';
11  local_18[15] = '\0';
12  local_18[8] = '0';
13  local_18[9] = '.';
14  local_18[10] = '3';
15  local_18[11] = '3';
16  local_18[4] = '2';
17  local_18[5] = '5';
18  local_18[6] = '4';
19  local_18[7] = '.';
20  local_18[0] = '1';
21  local_18[1] = '6';
22  local_18[2] = '9';
23  local_18[3] = '.';
24  xor_decrypt((byte)key,0x140008000);
25  steal_content_from_registry(local_18,&DAT_140008000);
26  return 0;
27 }
```

UNIVERSITÄT BONN

# Deobfuscating encrypted strings via emulation

## Original C code from our example malware:

```
1 void xor_decrypt(byte param_1, char *param_2) {
2   for (int local_c = 0; param_2[local_c] != '\0'; local_c = local_c + 1) {
3     param_2[local_c] = param_2[local_c] ^ param_1;
4   }
5 }
```

UNIVERSITÄT BONN

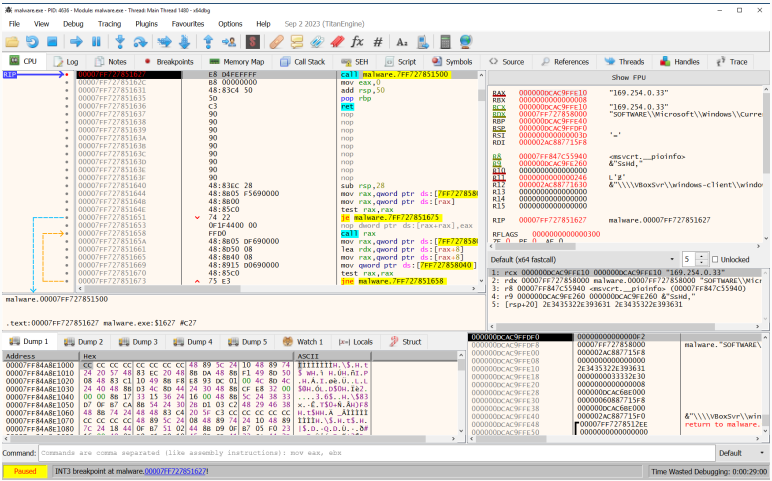
# Deobfuscating encrypted strings via emulation

## Re-implemented python version:

```python
def xor_decrypt(key: int, content: list[int]) -> None:
    i = 0
    while content[i] != 0:
        content[i] = content[i] ^ key
        i += 1

secret = [ 161, 189, 180, 166, 165, 179, 160, 183, 174, 191, 155, 145, 128,
           157, 129, 157, 148, 134, 174, 165, 155, 156, 150, 157, 133, 129,
           174, 177, 135, 128, 128, 151, 156, 134, 164, 151, 128, 129, 155,
           157, 156, 174, 160, 135, 156, 189, 156, 145, 151, 0 ]
key = 242
xor_decrypt(key, secret)
print(''.join(chr(c) for c in secret))
# >>> SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
```

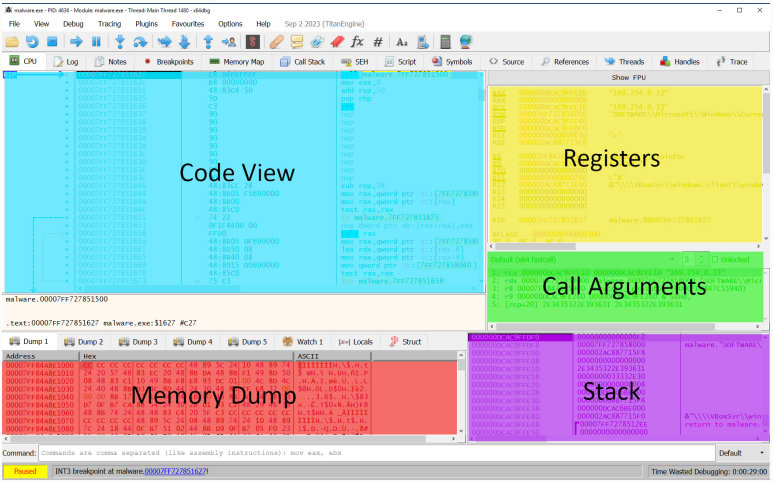
UNIVERSITÄT BONN

# Deobfuscating with x64dbg



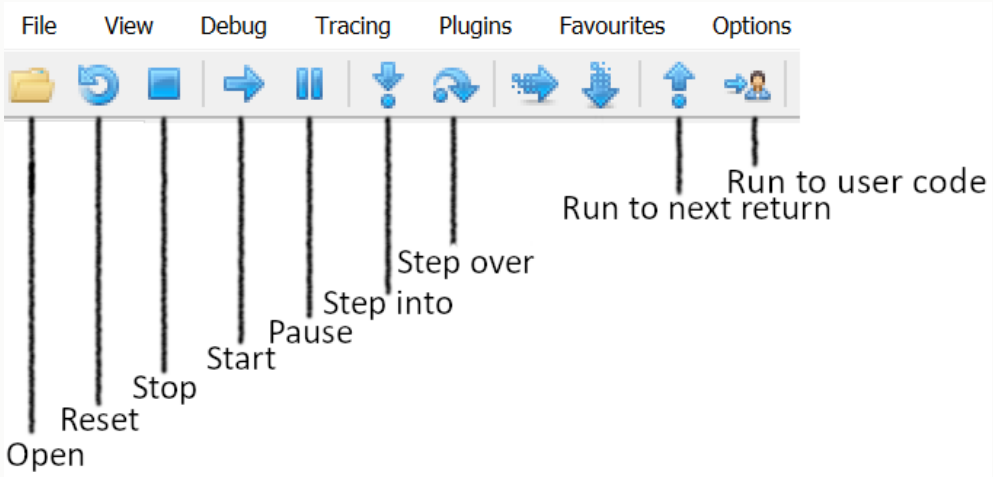
UNIVERSITÄT BONN

# Deobfuscating with x64dbg



UNIVERSITÄT BONN

# Deobfuscating with x64dbg



UNIVERSITÄT BONN

# Deobfuscating with x64dbg

```
1  unsigned char registry_key[] = {161, 189, ...};
2
3  int main(void) {
4    unsigned int ip_0 = 0;
5    unsigned int ip_1 = 0x33332e30;
6    unsigned int ip_2 = 0x2e343532;
7    unsigned int ip_3 = 0x2e393631;
8
9    xor_decrypt(key, registry_key);
10   steal_content_from_registry((unsigned char *) &ip_3, registry_key);
11 }
```

UNIVERSITÄT BONN

# Deobfuscating with x64dbg

- Run until we expect to find clear text data
- Function call at line `10` seems like a good candidate
- Use x64dbg's registers view to extract strings
- How do we know the function call position within the binary?

UNIVERSITÄT BONN

# Find specific address in x64dbg

1.

```
140001586 e8 75 ff ff ff        CALL        steal_content_from_registry
```

2.

CPU | Log | Notes | Breakpoints | Memory Map | Call Sta

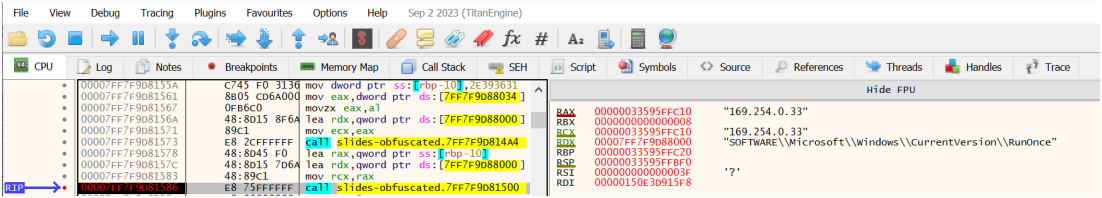
| Address | Size | Party | Info |
|---|---|---|---|
| 00007FF5F0F50000 | 0000000000001000 | User | |
| 00007FF5F0F60000 | 0000000000001000 | User | |
| 00007FF5F0F70000 | 0000000000023000 | User | |
| 00007FF7F9D80000 | 0000000000001000 | User | slides-obf |
| 00007FF7F9D81000 | 0000000000007000 | User | ".text" |

3.

```
00007FF7F9D81583    48:89C1          mov rcx,rax
00007FF7F9D81586    E8 75FFFFFF      call slides-obfuscated.7FF7F9D81500
00007FF7F9D8158B    B8 00000000      mov eax,0
```

UNIVERSITÄT BONN

# Deobfuscating with x64dbg



UNIVERSITÄT BONN

# Deobfuscating with x64dbg

- Last 2 bytes of address are usually static
- Find them e.g. in ghidra
- Look up start of `.text` section in memory map on x64dbg
- Set breakpoint at desired offset
- Function arguments are found in the registers section

UNIVERSITÄT BONN

# Summary

- Obfuscation makes analysis hard
- Static analysis can help for simple things
- Dynamic analysis can be more powerful, but also more challenging
- There is no silver bullet

# Questions?

UNIVERSITÄT BONN

# Anti-Debugging

- 💪 Motivation
- 🦠 Getting Infected
  - PDF Analysis
- 🔎 PE Analysis
  - Obfuscation
  - Anti-Debugging
  - Anti-Sandboxing
- 🧑‍🎓 Outlook

UNIVERSITÄT BONN

# About Anti-Debugging

- Makes dynamic analysis harder
- Various techniques
  - Windows' Win32 API
  - Windows internal flags
  - Many, many side effects (see `https://anti-debug.checkpoint.com`)

UNIVERSITÄT BONN

# Anti-Debug — Windows' Win32 API

- Built-In Windows API
- Function `IsDebuggerPresent()` from `debugapi.h`
- Intended to enable applications to provide additional information

UNIVERSITÄT BONN

# Anti-Debug — Windows' Win32 API

```c
#include <windows.h>

int main(void) {
    if (IsDebuggerPresent()) {
        do_some_evasive_stuff_like_exit();
    } else {
        do_evil_stuff();
    }
}
```

UNIVERSITÄT BONN

# Anti-Debug — Process Environment Block

- Contains some process context information
- For example the `BeingDebugged` flag
- Internal structure $\rightarrow$ no stable API

UNIVERSITÄT BONN

# Anti-Debug — PEB

```c
#include <windows.h>

int main(void) {
    // decompilation shows something like `(unaff_GS_OFFSET + 0x60)`
    PPEB pPeb = (PPEB)__readgsqword(0x60);
    if (pPeb->BeingDebugged) {
        do_some_evasive_stuff_like_exit();
    } else {
        do_evil_stuff();
    }
}
```

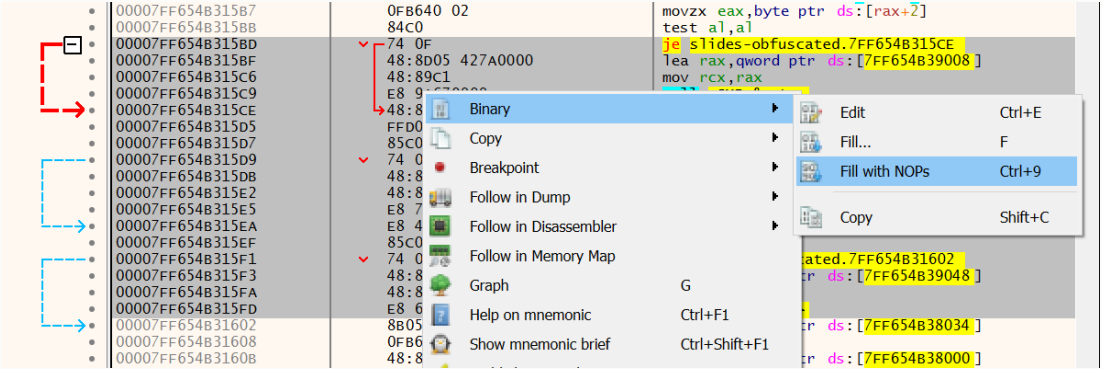
UNIVERSITÄT BONN

# Anti-Debug Circumvention

- Patching
- Modify Return Values
- ScyllaHide for x64DBG
- Hook API calls

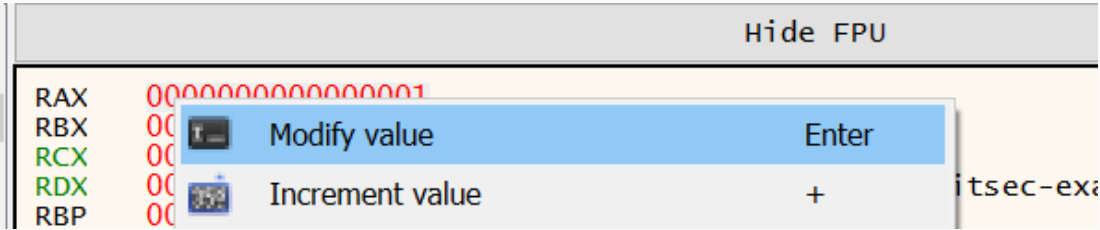
UNIVERSITÄT BONN

# Anti-Debug Circumvention — Patching



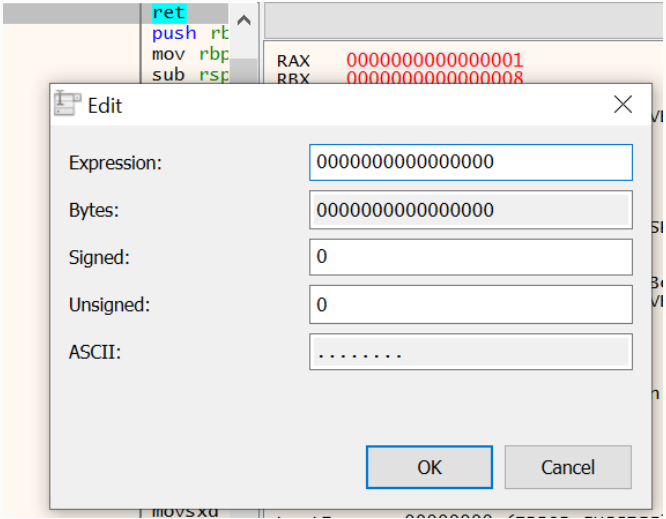
UNIVERSITÄT BONN

# Anti-Debug Circumvention — Modify Return Values

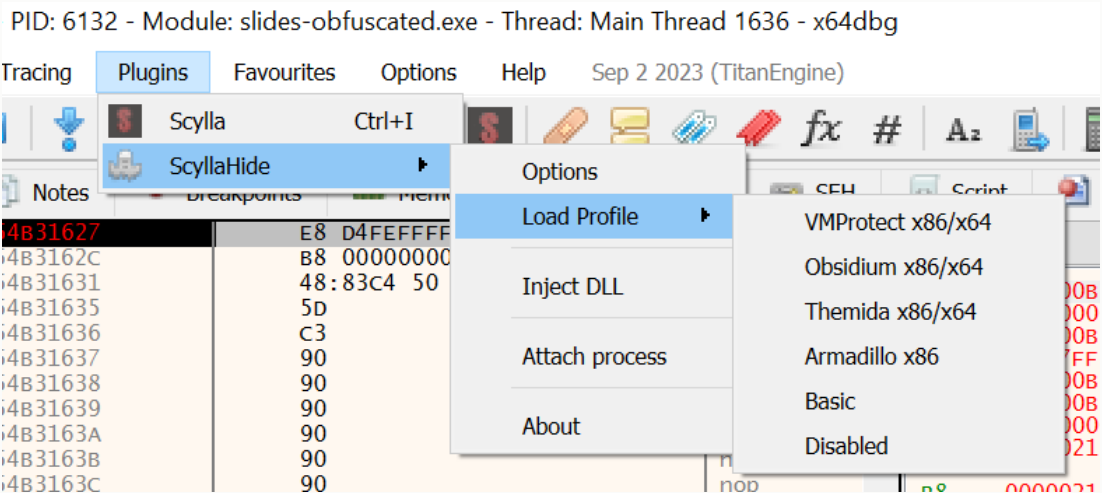


UNIVERSITÄT BONN

# Anti-Debug Circumvention — Modify Return Values



# Anti-Debug Circumvention — ScyllaHide



UNIVERSITÄT BONN

# Summary

- Anti-Debug techniques
  - Use / abuse various mechanisms to detect debugger presence
  - Lead to alternative code paths
- Anti-Debug circumvention
  - Jump over anti-debug routines
  - Change return values of detection routines
  - Use existing tools

# Questions?

UNIVERSITÄT BONN

# Anti-Sandboxing

- 💪 Motivation
- 🦠 Getting Infected
  - PDF Analysis
- 🔎 PE Analysis
  - Obfuscation
  - Anti-Debugging
  - Anti-Sandboxing
- 🧑‍🎓 Outlook

UNIVERSITÄT BONN

# Anti-Sandbox Techniques

- File system
- Processor instructions
- Hardware detection
- Network
- Behaviour
- Guest Additions
- … (`https://evasions.checkpoint.com`)

UNIVERSITÄT BONN

# Anti-Sandbox — File system artifacts

- Files specific to Virtual Machine
  - `System32\drivers\VBoxMouse.sys`
  - `System32\drivers\VBoxVideo.sys`
  - …
- Binary path contains keywords (“sample”, “malware”, …)
- …

UNIVERSITÄT BONN

# Anti-Sandbox — CPU instructions

- CPUID instruction
  - Returns CPU information like the vendor name
  - In sandboxed environments usually a sandbox-specific string
- Exotic instructions
  - Sandboxes often don’t implement all CPU instructions
- Undocumented instructions
- …

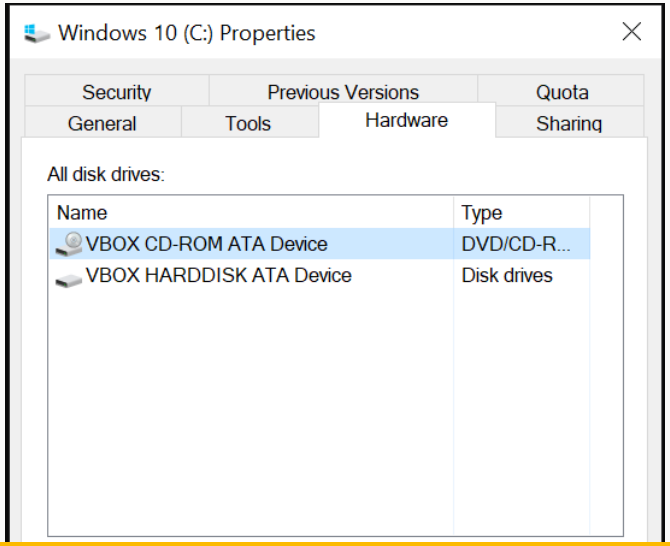
UNIVERSITÄT BONN

# Anti-Sandbox — Hardware Detection

- Device Names
  - In sanboxed environments usually sandbox-specific
- CPU Temperature Information
  - Sensors often not available to sandbox
- Audio device availability
- …

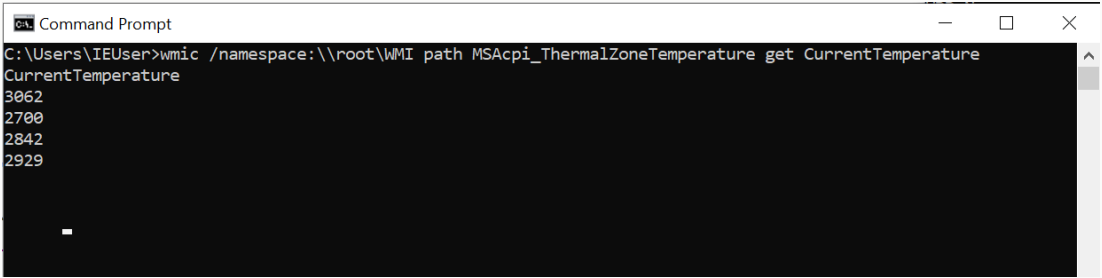
UNIVERSITÄT BONN

# Anti-Sandbox — Device Names



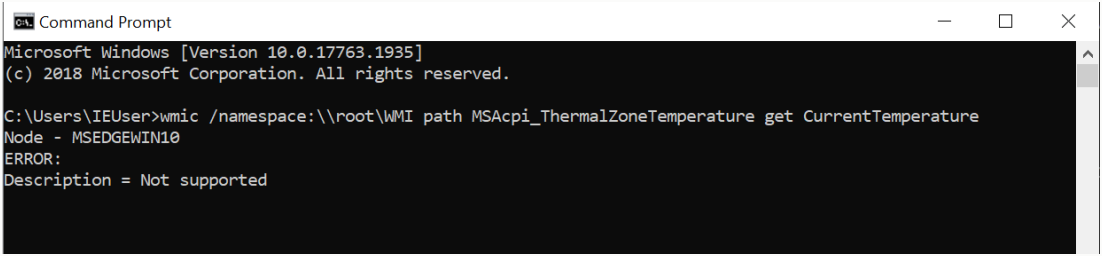
UNIVERSITÄT BONN

# CPU Temperature Information on Bare Metal



UNIVERSITÄT BONN

# CPU Temperature Information in a Sandbox



UNIVERSITÄT BONN

# Anti-Sandbox — Mitigations

- Existing tools
  - https://github.com/nsmfoo/antivmdetection
  - https://github.com/hfiref0x/VBoxHardenedLoader
- Painfully do it yourself
- Test your hardening with other tools
  - https://github.com/LordNoteworthy/al-khaser
  - https://github.com/a0rtega/pafish

UNIVERSITÄT BONN

# Summary

- There are *many* things giving away a sandboxed environment
- Some are of technical nature due to the environment
- Some are due to the absence of a real user
- Some are due to the habits of researchers
- Only mitigation is to know them all (or use tools from authors that do)

# Questions?

UNIVERSITÄT BONN

# Outlook

- 💪 Motivation
- 🦠 Getting Infected
  - PDF Analysis
- 🔎 PE Analysis
  - Obfuscation
  - Anti-Debugging
  - Anti-Sandboxing
- 👩‍🎓 Outlook

UNIVERSITÄT BONN

# Outlook

- Infection chains
- Payloads
- Process Injections
- Persistence
- Much more detail
- Real malware
- Practical Network Analysis

# Timo Pohl

University of Bonn | Institute of Computer Science 4

pohl@cs.uni-bonn.de


Lightning Surveys ⚡