Obfuscation Techniques

REVERSE ENGINEERING

João Paulo Barraca

deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

Obfuscation Techniques

- Aims at hardening the process of reverse engineering
  - Increases level of experience required
  - Increases cost (time, money)
  - Imposes the need for specific tools, techniques and procedures

- Applications (some):
  - **License protected software**: to prevent the generation of arbitrary licenses or subversion of the program code
  - **Proprietary software**: prevent the recovery of a design pattern or algorithm (IP protection)
  - **Malware**: to prevent recovery of the actions, prevent detection, Social Engineer users

Obfuscation Techniques

Static vs Dynamic

- Static obfuscation frequently transforms code before execution
  - Maybe before compilation, or during compilation
  - Countering static analysis
  - An obfuscated program is complex to analyze

- Dynamic obfuscation transforms code during execution
  - Countering Dynamic Analysis
  - The obfuscated program may change its behavior, expand or include further code

Obfuscation Techniques

Main Categories (Balachandran, TIFS 2013)

- Layout Obfuscation
- Design Obfuscation
- Data Obfuscation
- Control Obfuscation

- Also: Content Type Obfuscation

Content Type Obfuscation

- Dissimulate one file type as another file type or as raw data
  - Exploring how the file is processed
  - Exploring how users interact with it
  - Exploring how researchers and automatic tools process a file
- Purposes (some):
  - Marketing, branding and usability
  - Exploit users through social engineering
  - Increase the cost required for a reverse engineering task
  - Carry a malicious payload while escaping manual analysis
  - Carry a malicious payload bypassing automatic filtering

Content Type Obfuscation

Marketing, Branding and Usability

- Aims to make a filetype more usable, or to make the brand present to the user
  - Benning and common usage

- **Approach**: file has one specific type, but uses another file extension
  - Environment has a configuration stating how to handle such file extension
  - Explores the fact that an Environment uses fixed string to know how to open file

- **Impact**: File explorers will present a content based on the file extension, not based on the content

Content Type Obfuscation

Marketing, Branding and Usability

- For a PPTX file
  - File reports a zip file and magic is PK
  - DOCX and XLSX are similar

```
 $ unzip -l 8\ -\ Obfuscation.pptx
Archive:  8 - Obfuscation.pptx
  Length      Date    Time    Name
---------  ---------- -----   ----
     5179  1980-01-01 00:00   ppt/presentation.xml
    12041  1980-01-01 00:00   customXml/item1.xml
     1203  1980-01-01 00:00   customXml/itemProps1.xml
      219  1980-01-01 00:00   customXml/item2.xml
      335  1980-01-01 00:00   customXml/itemProps2.xml
      394  1980-01-01 00:00   customXml/item3.xml
      606  1980-01-01 00:00   customXml/itemProps3.xml
    33895  1980-01-01 00:00   ppt/slideMasters/slideMaster1.xml
     2477  1980-01-01 00:00   ppt/slides/slide1.xml
     4665  1980-01-01 00:00   ppt/slides/slide2.xml
     4384  1980-01-01 00:00   ppt/slides/slide3.xml
     4003  1980-01-01 00:00   ppt/slides/slide4.xml
     4719  1980-01-01 00:00   ppt/slides/slide5.xml
```

```
PK.........!.x......;.......ppt/presentati
on.xml...n.8.....;..-............@P...Jt"T'.
t.t...$-.CS(z.w.........x.....W>..k.>..|..E
WV.......2....%.../.w..O......~....I5.SaZ.`K
.E.~...x...7].[....aR....r`?T...q.%a.......3
......w..Q.....6>.K..)Z.;.`..%.^...Mp..Z)...
...u.7......B^...r.cDS...*v.B.Q....m87..$..z
w...1.........[.kr4?O........)..l...\..! ..
....#'pv..).Q....r..pu..=..n...C{...u....R.u
..N0.]z....>k..~..x....]~i.u._..a._..a._....
......,........ ?...a~.....G\~..~d..5.f...Kf
.Y../....R....r..../....?....r.8u......?.A..
.G"k}..AV|... ...~.....G"..:.H...u...:..~$..
+........^.:....o..b..R....K?..L.1.Mi..M...#
JO.J.g.Z>.7...5_.2...q...<.^.t........C....j
```

Content Type Obfuscation

Explore users through social engineering

- Aims to confuse users about the purpose of a file
  - Malicious and common in phishing campaigns and malware
- Approach: file has a filename and presentation that confuses users
  - Mail client or explorer presents a safe file with known extension
  - But… icon is stored in the file metadata, and file has two extensions (file.txt.exe)
- Impact: User thinks that a file is not malicious (e.g, it's a word document), while in reality, it executes a malicious code

Content Type Obfuscation

Explore users through social engineering

- Windows hides extension of known file types
  - **`Sample.pptx`** becomes only **`Sample`**
- Executable files may have an embedded icon
  - Freely defined by the developer
  - Explorer will show that icon
- A file named **`Sample.pptx.exe`** will be shown as **`Sample.pptx`**
  - Users recognize the extension and may think the file is safe
- In a RE task, a file may have bogus extensions

Content Type Obfuscation

Increase the cost required for a reverse engineering task

- Aims to disguise/manipulate files so that a RE task skips the file, or processes the file incorrectly

- Approaches:
  - Hides content in file without extension, without headers or with modified headers
  - Mangles content to make it less human friendly
  - Polyglots

- Impact: Reversing or Forensics Analyst will not process the file, or will not process the file with the correct approach/tools
  - May prevent the researcher from recovering the original file

Content Type Obfuscation

Magic Headers

- Besides extensions, most files can be recognized by a magic value in the file start/end
  - Manipulating headers can lead to incorrect detection and maybe processing

- Some magic values:
  - Office Documents: `D0 CF 11 E0`
  - ELF: `7F E L F`
  - JPG: `FF D8`
  - PNG: `89 P N G 0D 0A 1A 0A`
  - Java class: `CA FE BA BE`

Content Type Obfuscation

Magic Headers

- Headers are important to maintain compatibility with third party software

- Headers may be irrelevant for custom software
  - Software has the filetype hard coded

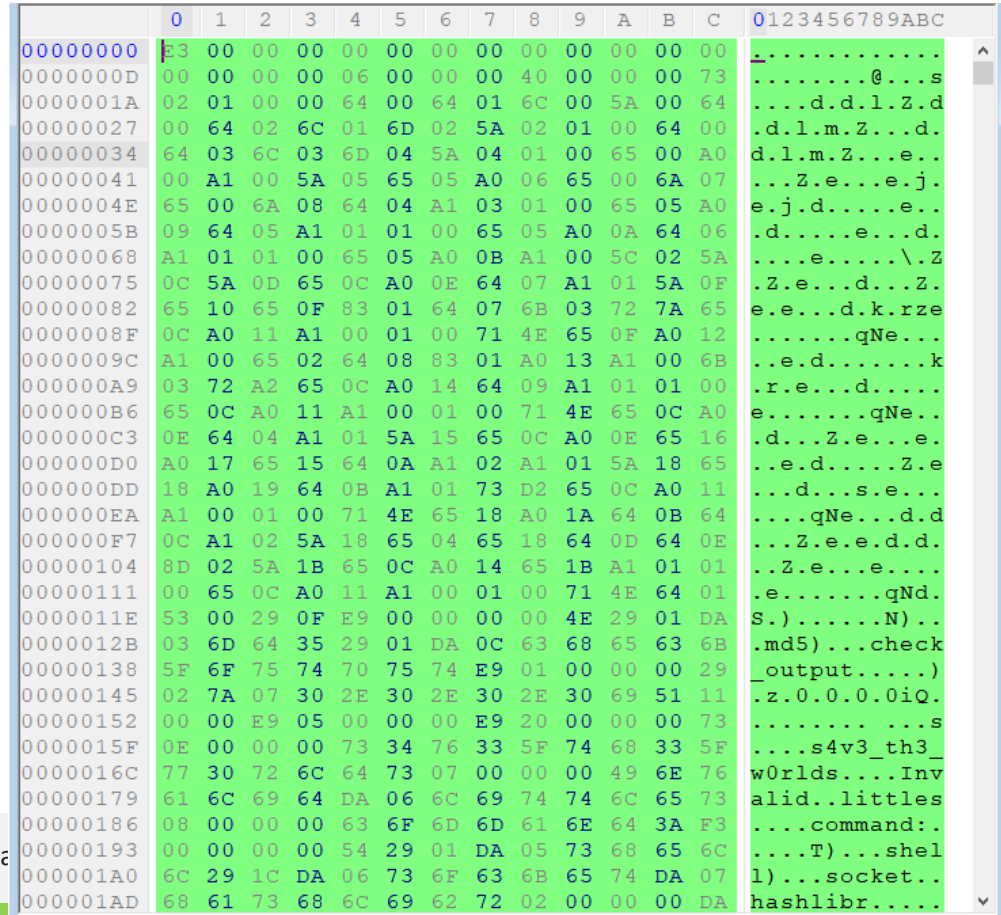
Content Type Obfuscation

Magic Headers

- `PyInstaller` allows converting Python code to an executable
  - It packs the `pyc` files into a container. Container is extracted on runtime and compiled python code is executed
  - Headers are omitted from `pyc` files. If header is added, extracted file executes as a standard `pyc` file

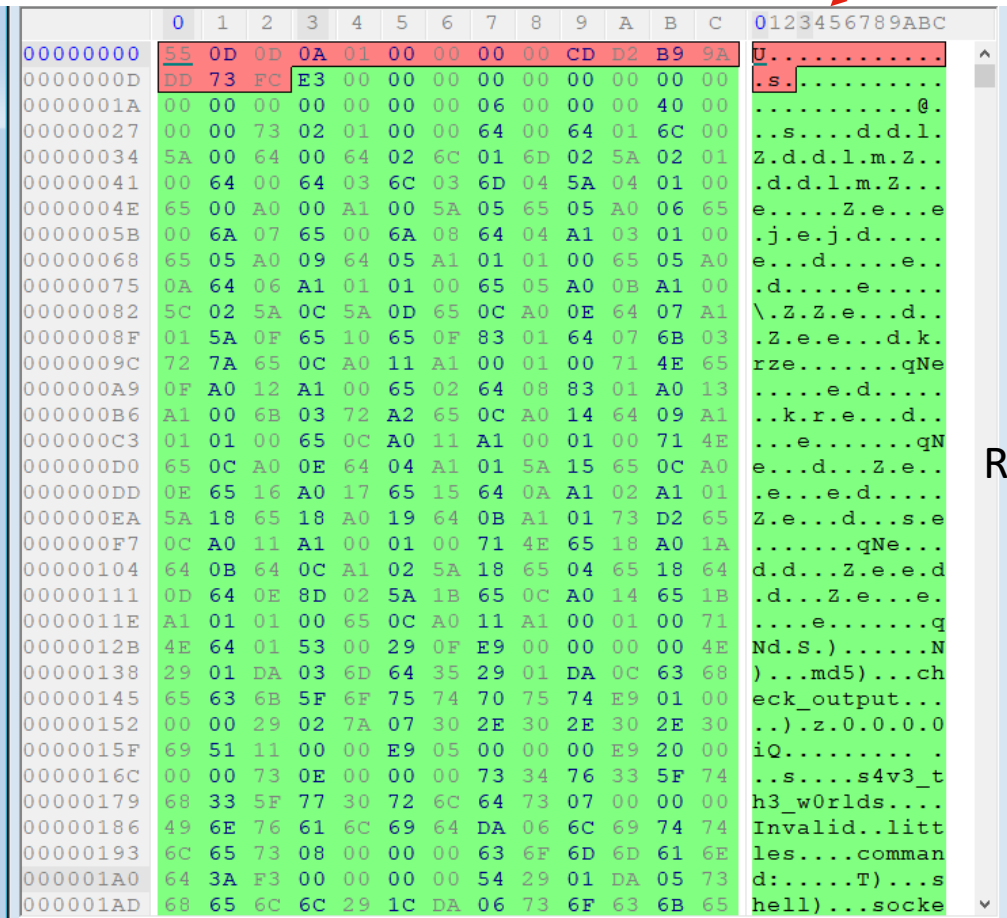
Added header

Extracted



	0	1	2	3	4	5	6	7	8	9	A	B	C	0123456789ABC
00000000	E3	00	00	00	00	00	00	00	00	00	00	00	00	.....
0000000D	00	00	00	00	06	00	00	00	40	00	00	00	73	........@...s
0000001A	02	01	00	00	64	00	64	01	6C	00	5A	00	64	....d.d.l.Z.d
00000027	00	64	02	6C	01	6D	02	5A	02	01	00	64	00	.d.l.m.Z...d.
00000034	64	03	6C	03	6D	04	5A	04	01	00	65	00	A0	d.l.m.Z...e..
00000041	00	A1	00	5A	05	65	05	A0	06	65	00	6A	07	...Z.e...e.j.
0000004E	65	00	6A	08	64	04	A1	03	01	00	65	05	A0	e.j.d.....e..
0000005B	09	64	05	A1	01	01	00	65	05	A0	0A	64	06	.d.....e...d.
00000068	A1	01	01	00	65	05	A0	0B	A1	00	5C	02	5A	....e.....\.Z
00000075	0C	5A	0D	65	0C	A0	0E	64	07	A1	01	5A	0F	.Z.e...d...Z.
00000082	65	10	65	0F	83	01	64	07	6B	03	72	7A	65	e.e...d.k.rze
0000008F	0C	A0	11	A1	00	01	00	71	4E	65	0F	A0	12	.......qNe...
0000009C	A1	00	65	02	64	08	83	01	A0	13	A1	00	6B	..e.d.......k
000000A9	03	72	A2	65	0C	A0	14	64	09	A1	01	01	00	.r.e...d.....
000000B6	65	0C	A0	11	A1	00	01	00	71	4E	65	0C	A0	e.......qNe..
000000C3	0E	64	04	A1	01	5A	15	65	0C	A0	0E	65	16	.d...Z.e...e.
000000D0	A0	17	65	15	64	0A	A1	02	A1	01	5A	18	65	..e.d.....Z.e
000000DD	18	A0	19	64	0B	A1	01	73	D2	65	0C	A0	11	...d...s.e...
000000EA	A1	00	01	00	71	4E	65	18	A0	1A	64	0B	64	....qNe...d.d
000000F7	0C	A1	02	5A	18	65	04	65	18	64	0D	64	0E	...Z.e.e.d.d.
00000104	8D	02	5A	1B	65	0C	A0	14	65	1B	A1	01	01	..Z.e...e....
00000111	00	65	0C	A0	11	A1	00	01	00	71	4E	64	01	.e.......qNd.
0000011E	53	00	29	0F	E9	00	00	00	00	4E	29	01	DA	S.).....N)..
0000012B	03	6D	64	35	29	01	DA	0C	63	68	65	63	6B	.md5)...check
00000138	5F	6F	75	74	70	75	74	E9	01	00	00	00	29	_output.....)
00000145	02	7A	07	30	2E	30	2E	30	2E	30	69	51	11	.z.0.0.0.0iQ.
00000152	00	00	E9	05	00	00	00	E9	20	00	00	00	73	....s
0000015F	0E	00	00	00	73	34	76	33	5F	74	68	33	5F	....s4v3_th3_
0000016C	77	30	72	6C	64	73	07	00	00	00	49	6E	76	w0rlds....Inv
00000179	61	6C	69	64	DA	06	6C	69	74	74	6C	65	73	alid..littles
00000186	08	00	00	00	63	6F	6D	61	6E	64	3A	F3		....command:.
00000193	00	00	00	00	54	29	01	DA	05	73	68	65	6C	....T)...shel
000001A0	6C	29	1C	DA	06	73	6F	63	6B	65	74	DA	07	l)...socket..
000001AD	68	61	73	68	6C	69	62	72	02	00	00	00	DA	hashlibr.....

Reconstructed



	0	1	2	3	4	5	6	7	8	9	A	B	C	0123456789ABC
00000000	55	0D	0D	0A	01	00	00	00	00	CD	D2	B9	9A	U.....
0000000D	DD	73	FC	E3	00	00	00	00	00	00	00	00	00	.s..........
0000001A	00	00	00	00	00	00	00	06	00	00	00	40	00	...........@.
00000027	00	00	73	02	01	00	00	64	00	64	01	6C	00	..s....d.d.l.
00000034	5A	00	64	00	64	02	6C	01	6D	02	5A	02	01	Z.d.d.l.m.Z..
00000041	00	64	00	64	03	6C	03	6D	04	5A	04	01	00	.d.d.l.m.Z...
0000004E	65	00	A0	00	A1	00	5A	05	65	05	A0	06	65	e.....Z.e...e
0000005B	00	6A	07	65	00	6A	08	64	04	A1	03	01	00	.j.e.j.d.....
00000068	65	05	A0	09	64	05	A1	01	01	00	65	05	A0	e...d.....e..
00000075	0A	64	06	A1	01	01	00	65	05	A0	0B	A1	00	.d.....e.....
00000082	5C	02	5A	0C	5A	0D	65	0C	A0	0E	64	07	A1	\.Z.Z.e...d..
0000008F	01	5A	0F	65	10	65	0F	83	01	64	07	6B	03	.Z.e.e...d.k.
0000009C	72	7A	65	0C	A0	11	A1	00	01	00	71	4E	65	rze.......qNe
000000A9	0F	A0	12	A1	00	65	02	64	08	83	01	A0	13	.....e.d.....
000000B6	A1	00	6B	03	72	A2	65	0C	A0	14	64	09	A1	..k.r.e...d..
000000C3	01	01	00	65	0C	A0	11	A1	00	01	00	71	4E	...e.......qN
000000D0	65	0C	A0	0E	64	04	A1	01	5A	15	65	0C	A0	e...d...Z.e..
000000DD	0E	65	16	A0	17	65	15	64	0A	A1	02	A1	01	.e...e.d.....
000000EA	5A	18	65	18	A0	19	64	0B	A1	01	73	D2	65	Z.e...d...s.e
000000F7	0C	A0	11	A1	00	01	00	71	4E	65	18	A0	1A	.......qNe...
00000104	64	0B	64	0C	A1	02	5A	18	65	04	65	18	64	d.d...Z.e.e.d
00000111	0D	64	0E	8D	02	5A	1B	65	0C	A0	14	65	1B	.d...Z.e...e.
0000011E	A1	01	01	00	65	0C	A0	11	A1	00	01	00	71	....e.......q
0000012B	4E	64	01	53	00	29	0F	E9	00	00	00	00	4E	Nd.S.).....N
00000138	29	01	DA	03	6D	64	35	29	01	DA	0C	63	68	)...md5)...ch
00000145	65	63	6B	5F	6F	75	74	70	75	74	E9	01	00	eck_output...
00000152	00	00	29	02	7A	07	30	2E	30	2E	30	2E	30	..).z.0.0.0.0
0000015F	69	51	11	00	00	E9	05	00	00	00	E9	20	00	iQ......... .
0000016C	00	00	73	0E	00	00	00	73	34	76	33	5F	74	..s....s4v3_t
00000179	68	33	5F	77	30	72	6C	64	73	07	00	00	00	h3_w0rlds....
00000186	49	6E	76	61	6C	69	64	DA	06	6C	69	74	74	Invalid..litt
00000193	6C	65	73	08	00	00	00	63	6F	6D	6D	61	6E	les....comman
000001A0	64	3A	F3	00	00	00	00	54	29	01	DA	05	73	d:.....T)...s
000001AD	68	65	6C	6C	29	1C	DA	06	73	6F	63	6B	65	hell)...socke

Code Obfuscation

Layout Obfuscation

- Aims at hiding how the **source code** is structured
  - As source code (or symbols) can present enough information to help reversing a program
- Applied to the source code, and focused on situations where source can be obtained
  - Javascript, HTML, CSS, Java
- Methods:
  - Deleting comments
  - Remove debugging information
  - Renaming classes, methods and variables
  - Removing spaces
  - Stripping a binary

Code Obfuscation

Design Obfuscation

- Aims at making the design nonobvious, more difficult to recover
  - Usually done by a tool before compilation or during compilation
  - GCC can do this automatically by inlining functions (`-O3 –finline -funroll-loops`)

- Methods:
  - Merging and splitting methods
  - Merging and splitting classes
  - Splitting binary code, while inserting dummy instructions
  - Splitting loops and conditions, maybe interleaved with dummy code
  - Inlining functions
  - Dead Code

Code Obfuscation

Design Obfuscation – Breaking Code

Code inserted, but never executed.
JMP before dummy code effectively only splits code

```c
#include <stdio.h>
#include <stdlib.h>

unsigned long long factorial(unsigned long long a) {

    unsigned long long r = 1;

    while(a > 0 ){
        unsigned long long v = r * a;
        if(v < r){
            printf("ERROR: Overflow\n");
            exit(-1);
        }
        r = v;
        a = a - 1;
    }
    return r;
}

int main(int argc, char** argv) {
    unsigned long long v = 0;
    if(argc != 2) {
        printf("Need a positive integer argument\n");
        return -1;
    }
    v = atol(argv[1]);

    if(v <= 0){
        printf("Need a positive integer argument\n");
        return -1;
    }

    printf("Result: %llu\n", factorial(v));

    return 0;
}
```

```c
int main(int argc, char** argv) {
    unsigned long long v = 0;
    if(argc != 2) {
        printf("Need a positive integer argument\n");
        return -1;
    }
    asm("jmp label");
    factorial(factorial(argc));
    asm("label:");
    v = atol(argv[1]);

    if(v <= 0){
        printf("Need a positive integer argument\n");
        return -1;
    }

    asm("jmp label_b");
    factorial(factorial(v * factorial(-v)));
    asm("label_b:");

    printf("Result: %llu\n", factorial(v));

    return 0;
}
```

Code Obfuscation

Design Obfuscation – Breaking Code

Code inserted, but never executed.
JMP before dummy code effectively only splits code

```c
#include <stdio.h>
#include <stdlib.h>

unsigned long long factorial(unsigned long long a) {

    unsigned long long r = 1;

    while(a > 0 ){
        unsigned long long v = r *
        if(v < r){
            printf("ERROR: Overflow
            exit(-1);
        }
        r = v;
        a = a - 1;
    }
    return r;
}

int main(int argc, char** argv) {
    unsigned long long v = 0;
    if(argc != 2) {
        printf("Need a positive int
        return -1;
    }
    v = atol(argv[1]);

    if(v <= 0){
        printf("Need a positive int
        return -1;
    }

    printf("Result: %llu\n", factor

    return 0;
}
```

```c
int main(int argc, char** argv) {
    unsigned long long v = 0;
    if(argc != 2) {
        printf("Need a positive integer argument\n");
        return -1;
```

What about the output binary?

Compile with gcc -O0 -o factorial-split factorial-split.c

Does it effect static or dynamic analysis?
Check with objdump -d and ghidra

What about if instead of jmp you use jz or jnz?

gcc may also inline functions (the opposite) when using –O3 or –finline-functions

Code Obfuscation

Design Obfuscation – Dead Code

- Aims at inserting dummy code to confuse the analysis
  - Code may follow some pattern (previous example), or be random
  - Code may lock the analysis tool if recursive disassembly is used
  - Decompilation to Pseudo C will surely be affected

- Dead code can be added after compilation
  - May contain fingerprinting information by making binaries unique

Code Obfuscation

Design Obfuscation – Dead Code

```
21  unsigned long long factorial(unsigned long long a) {
22
23      unsigned long long r = 1;
24
25      while(a > 0 ){
26          unsigned long long v = r * a;
27          if(v < r){
28              printf("ERROR: Overflow\n");
29              exit(-1);
30          }
31          r = v;
32          a = a - 1;
33
34          if(v != r) {
35              __asm__ (REP(3,3,3,"nop;"));
36          }
37      }
38      return r;
39  }
```

`r=v`, therefore, **`if(v!=r)`** will be always false. Compiler will not easily discard this code.

`__asm__....` Instruction will insert 333 NOPs (which will not be executed)

This is a placeholder that can be used later for post processing by editing the binary directly

Code Obfuscation

Design Obfuscation – Dead Code

```
2  undefined8 main(int param_1,long param_2)
3
4  {
5    undefined8 uVar1;
6    long lVar2;
7
8    if (param_1 == 0x2) {
9      lVar2 = atol(*(char **)(param_2 + 0x8));
10     if (lVar2 == 0x0) {
11       puts("Need a positive integer argument");
12       uVar1 = 0xffffffff;
13     }
14     else {
15       uVar1 = factorial(lVar2);
16       printf("Result: %llu\n",uVar1);
17       uVar1 = 0x0;
18     }
19   }
20   else {
21     puts("Need a positive integer argument");
22     uVar1 = 0xffffffff;
23   }
24   return uVar1;
25 }
26
```

Decompile: main - (factorial-dead-obf)

```
13   undefined4 *local_28;
14   int local_1c;
15   long local_10;
16
17   local_10 = 0x0;
18   local_28 = param_2;
19   local_1c = param_1;
20   if (param_1 == 0x2) {
21     puVar4 = *(undefined4 **)(param_2 + 0x2);
22     uStack48 = 0x10136a;
23     local_10 = atol((char *)puVar4);
24     if (local_10 == 0x0) {
25       uStack48 = 0x101381;
26       puts("Need a positive integer argument");
27       pcVar2 = (char *)0xffffffff;
28     }
29     else {
30       if (local_1c * local_10 == 0x0) {
31         *puVar4 = *param_2;
32         if ((POPCOUNT(local_1c * local_10 & 0xff) & 0x1U) != 0x0) {
33                     /* WARNING: Bad instruction - Truncating control flow here */
34           halt_baddata();
35         }
36         puVar4 = (undefined4 *)(ulong)((int)param_4 - 0x44);
37         puVar3 = &uStack48;
38         cVar1 = '\x12';
39         do {
40           puVar4 = puVar4 + -0x1;
41           puVar3 = (undefined8 *)((long)puVar3 + -0x4);
42           *(undefined4 *)puVar3 = *puVar4;
43           cVar1 = cVar1 + -0x1;
44         } while ('\0' < cVar1);
45                     /* WARNING: Bad instruction - Truncating control flow here */
46         halt_baddata();
47       }
48       uStack48 = 0x10172f;
49       factorial(local_10);
50       uStack48 = 0x101743;
51       printf("Result: %llu\n");
52       pcVar2 = (char *)(local_1c * local_10);
53       if (pcVar2 + -(local_10 + -0x3) != NULL) {
54         pcVar2 = NULL;
55       }
```

Code Obfuscation

Data Obfuscation

- Encrypts, or otherwise encodes data contents
  - Contents are decrypted in real time, as the program is executed
  - Static analysis, or fingerprint matching may fail to correctly recover useful information
  - Frequent tactic to evade filters

- Why?
  - Strings frequently carry semantic information, that may help analysis
  - E.g. Str="Please input your AES key": we will know that this a key, and know the algorithm

Code Obfuscation

Data Obfuscation - how

- Split the string in parts
  - May be combined with two conditions or loops to validate both parts individually
- Erase strings right after use
- Common XOR is frequently found as it requires no dependencies and is fast
  - More recent malware will use RC4 or even AES for this purpose
  - Decryption key can also be encrypted, and some key may be obtained dynamically
    - E.g. from a hardware token as a form of licensing enforcement
- Create a custom encoding based on a complex state machine
  - May use flow information, voiding the decoding of strings if the execution order it changed

Code Obfuscation

Control Obfuscation – Opaque Predicates

- Introduces dummy control structures, with little impact to execution
  - Impact is only from a performance point of view (additional branch)
  - However, analysis tools will interpret the control structures and create complex CFGs

- Makes use of Opaque Predicates: predicates for which the programmer already knows the result.
  - E.g. `if ( 1 > 0) or v=r; if(v==r)`

Code Obfuscation

Control Obfuscation – Opaque Predicates

- Opaque predicates can be more complex
- Manipulate pointers, linked lists, use computation processes
- Result of a predicate can be dynamic, and related to execution state
  - Dynamic analysis may change execution sequence, therefore the predicate result and invalidate the execution
  - Similar to TPMs, where keys are provided at a valid situation
  - Predicate can use dynamic data, received from external services
- Concurrency can be used to create predicates
  - If two threads are executing with some relation, one can update data, that the other uses to construct a predicate
  - Timing information can also be used, to further increase the complexity (information not available statically)

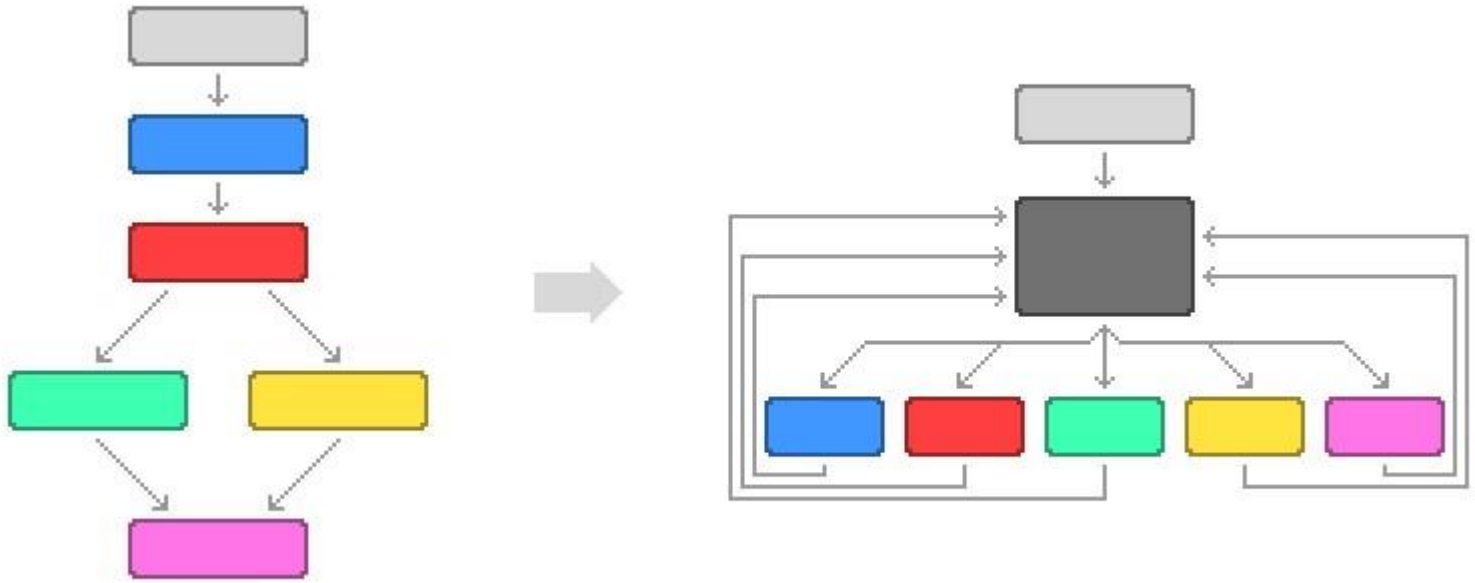
can be used with time if instruction takes a certain time it changes behaviour, can be used to detect if it's being analyzed

Code Obfuscation

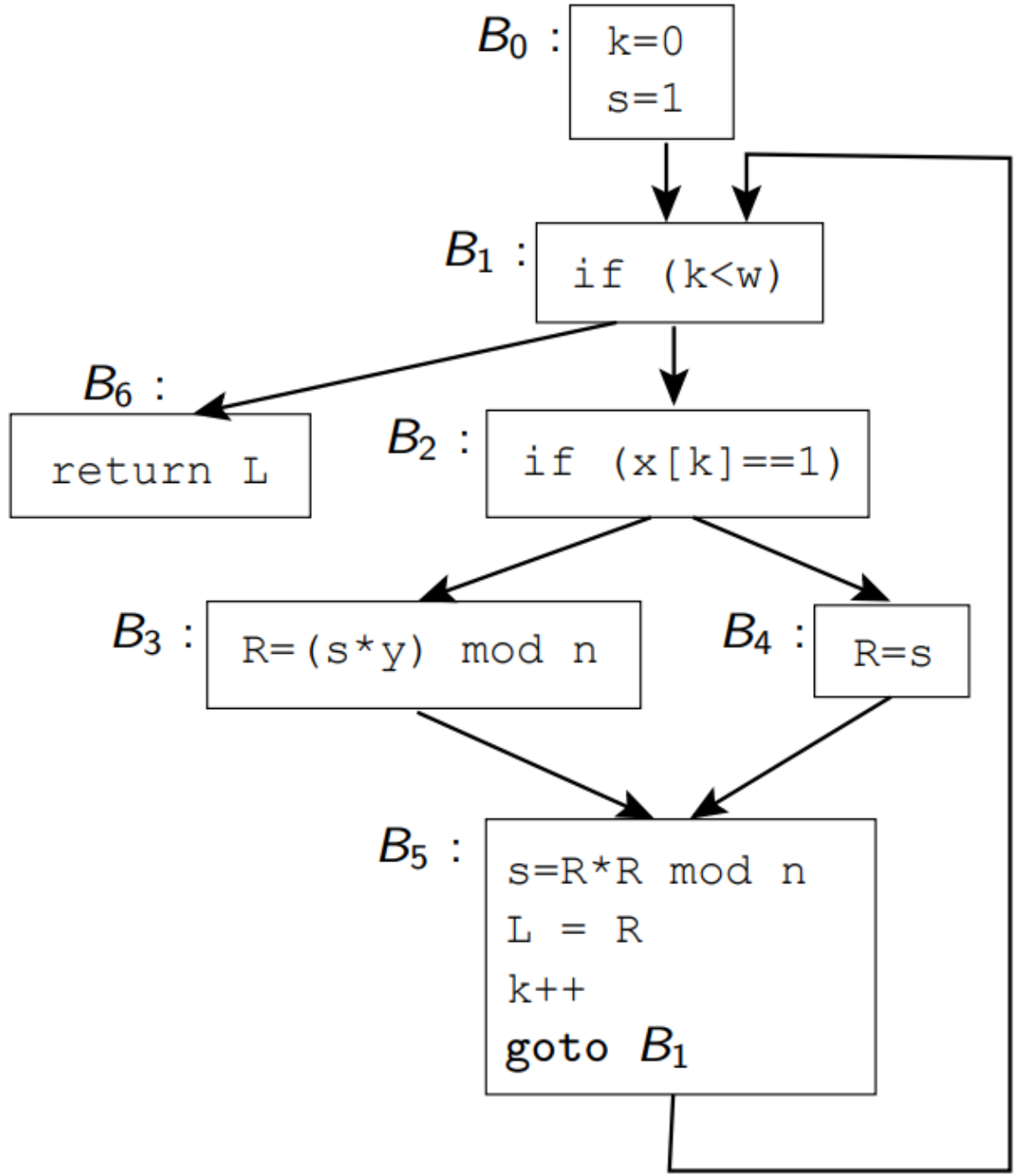
Control Obfuscation – Control Flow Flattening

- Removes control flow structures from program
  - Converts the program to a gigantic Switch, where each condition is a case
  - Program runs on an infinite loop around the switch

- Program becomes ~4 times slower, and 2 times larger



```
int modexp(int y,int x[],
           int w,int n) {
    int R, L;
    int k = 0;
    int s = 1;
    while (k < w) {
        if (x[k] == 1)
            R = (s*y) % n;
        else
            R = s;
        s = R*R % n;
        L = R;
        k++;
    }
    return L;
}
```



```
int modexp(int y, int x[], int w, int n) {
    int R, L, k, s;
    int next=0;
    for(;;)
        switch(next) {
            case 0 : k=0; s=1; next=1; break;
            case 1 : if (k<w) next=2; else next=6; break;
            case 2 : if (x[k]==1) next=3; else next=4; break;
            case 3 : R=(s*y)%n; next=5; break;
            case 4 : R=s; next=5; break;
            case 5 : s=R*R%n; L=R; k++; next=1; break;
            case 6 : return L;
        }
}
```

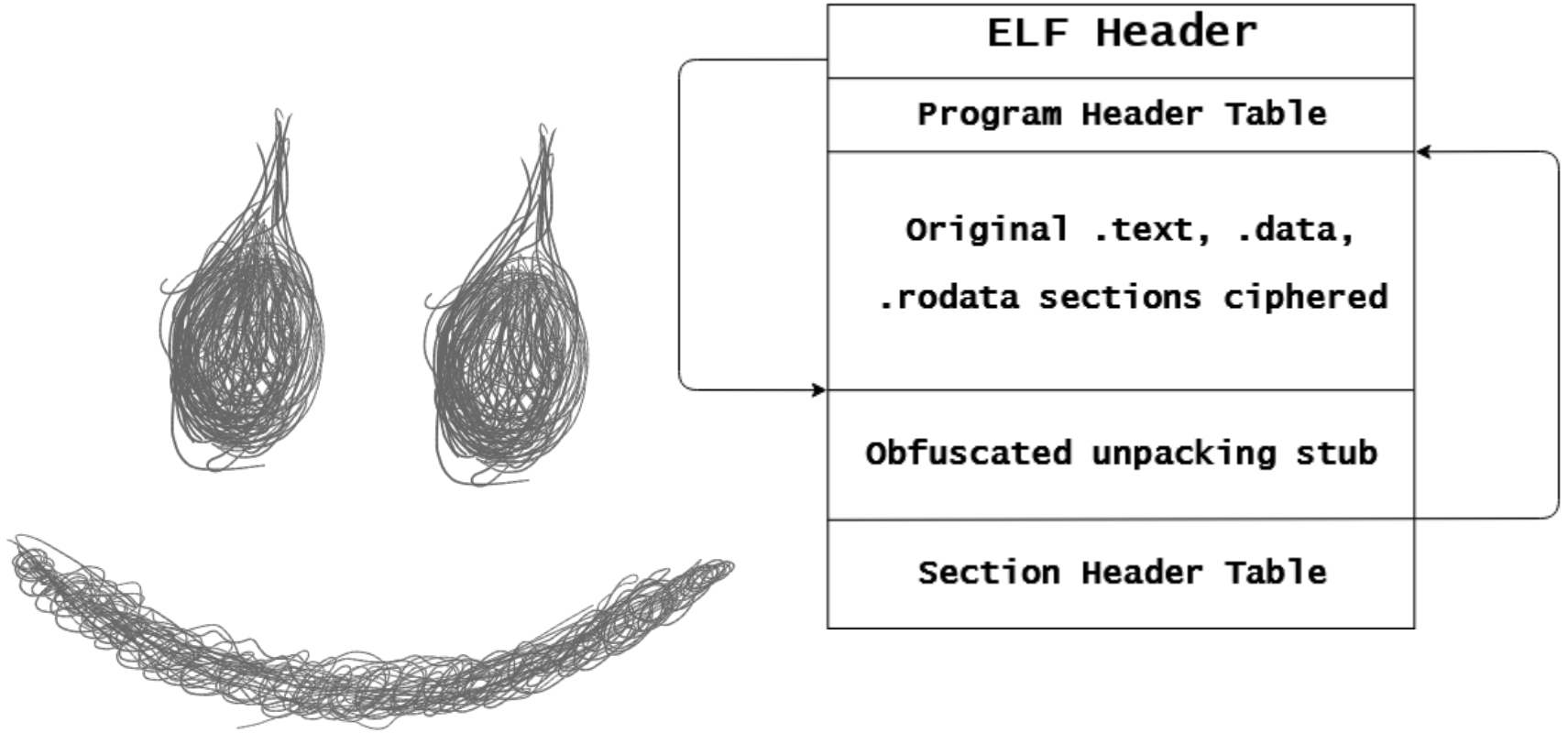
Code Obfuscation

Self Decompressing Binaries

- Binaries can be compressed into a blob (and even encrypted)
  - Stub will process the blob and jump into it
- Static analysis will be able to analyze the stub, which can be obfuscated
  - Stub provides a valid signature for scanners, but variations can exist
- Actual file is never available to analysis by static scanners
  - Is available at runtime, as file must be available for execution
  - Generic packers (upx) will pack the entire ELF, which is mapped at runtime
    - Easier to extract as file is recreated and mapped
  - Crafted packers will require more effort
- Generic approach uses a debugger or Qiling to dump the uncompressed file
  - For an overview, check: https://kernemporium.github.io/posts/unpacking/

Code Obfuscation

Self Decompressing Binaries



Code Obfuscation

Self Decompressing Binari