

C++11 METAPROGRAMMING APPLIED TO SOFTWARE OBFUSCATION

SEBASTIEN ANDRIVET

Application Security Forum - 2014

Western Switzerland

05-06 November 2014

Y-Parc / Yverdon-les-Bains

<http://www.appsec-forum.ch>



About me



Senior Security Engineer
at **SCRT** (Swiss)

Developer
at **ADVTOOLS** (Swiss)

Sebastien ANDRIVET
Cyberfeminist & hacktivist
Reverse engineer Intel & ARM
C++, C, Obj-C, C# developer
Trainer (iOS & Android appsec)

PROBLEM

Reverse engineering

- Reverse engineering of an application is often like following the “white rabbit”
 - i.e. following string literals
- Live demo
 - Reverse engineering of an application using IDA
 - Well-known MDM (Mobile Device Management) for iOS

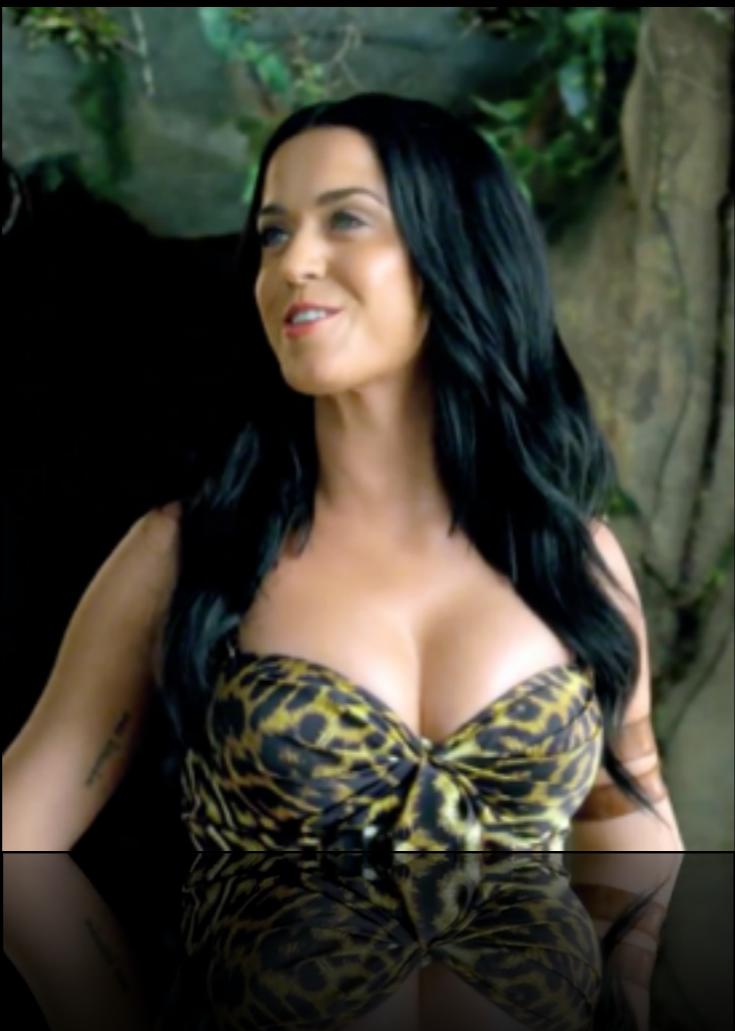
A SOLUTION OBFUSCATION

What is Obfuscation?

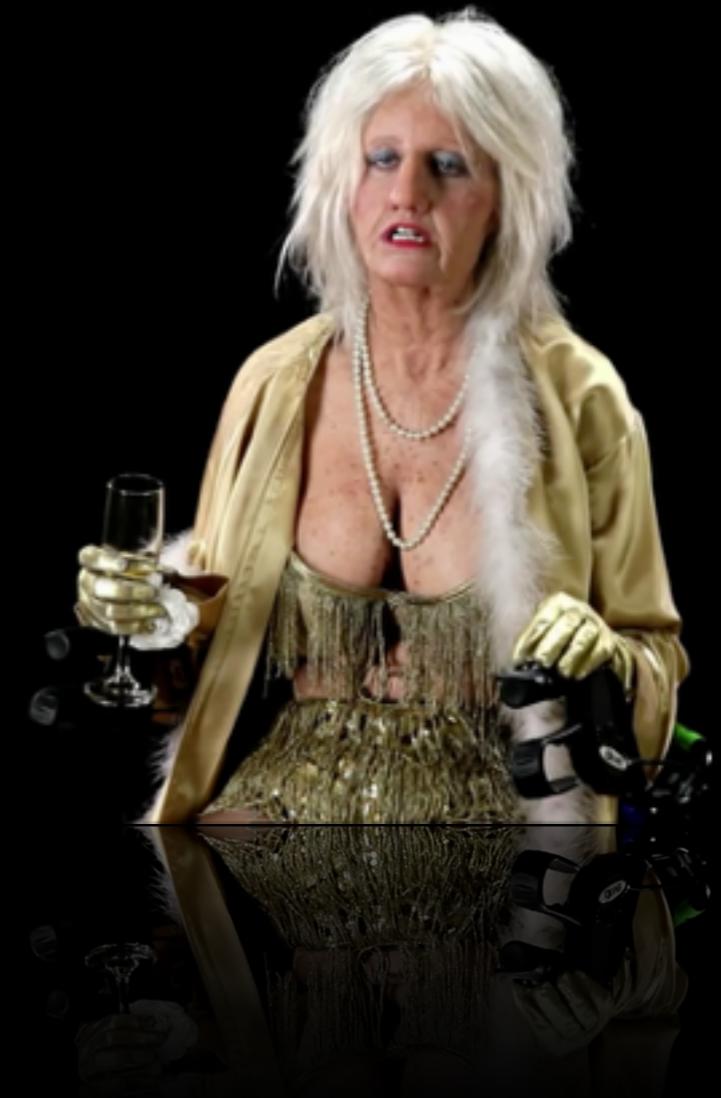


Obfuscator \mathcal{O}

$\mathcal{O}($



) =



YES! It is also Katy Perry!

- (almost) same semantics
- obfuscated



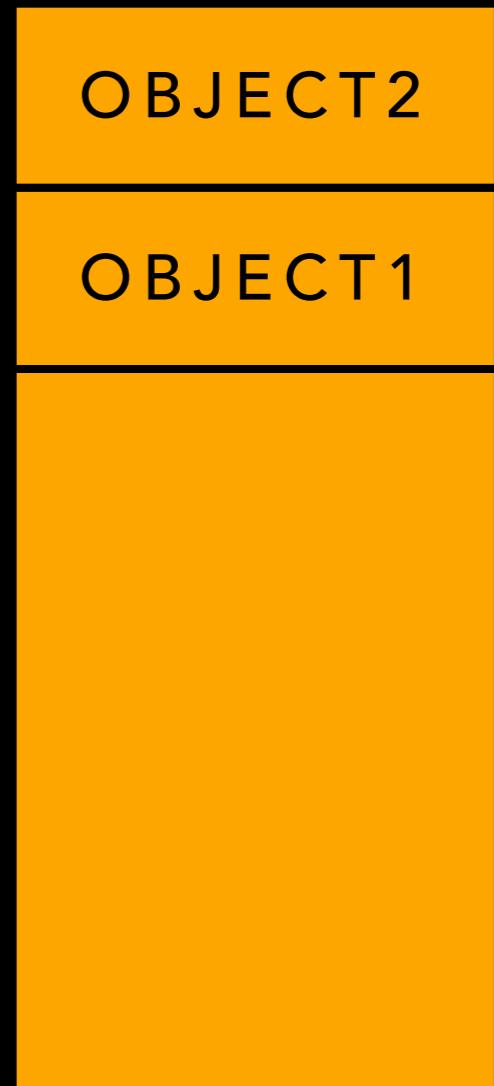
Obfuscation

“Deliberate act of creating source or machine code difficult for humans to understand”

–WIKIPEDIA, APRIL 2014

C++ templates

- Example: Stack of objects
 - Push
 - Pop

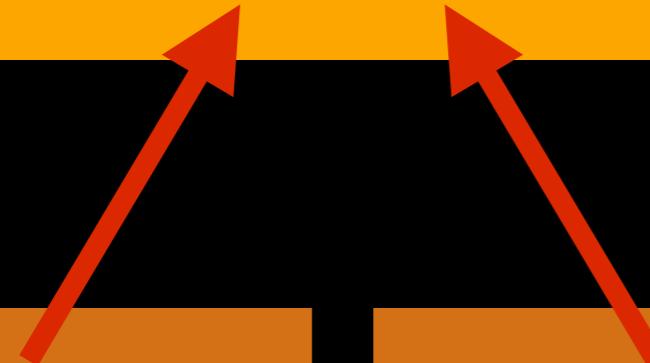


Without templates

```
class Stack
{
    void push(void* object);
    void* pop();
};
```

singers

apples



```
Stack singers;
singers.push(britney);
```

```
Stack apples;
apples.push(macintosh);
```

- Reuse the same code (binary)
- Only 1 instance of Stack class

With C++ templates

```
template<typename T>
class Stack
{
    void push(T object);
    T pop();
};
```

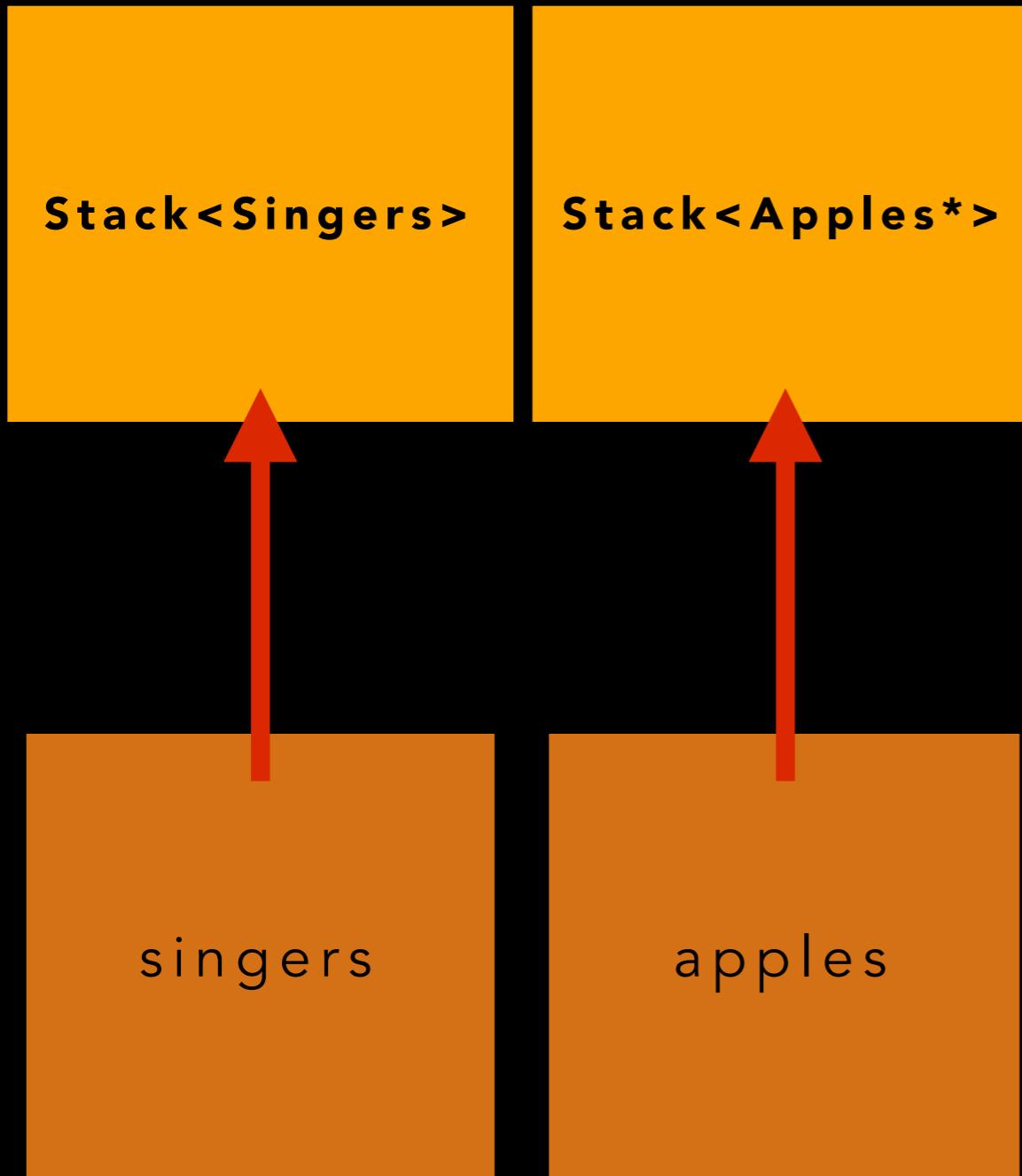
singers

apples

```
Stack<Singer> singers;
singers.push(britney);
```

```
Stack<Apple> apples;
apples.push(macintosh);
```

With C++ templates



```
Stack<Singer> singers;  
singers.push(britney);
```

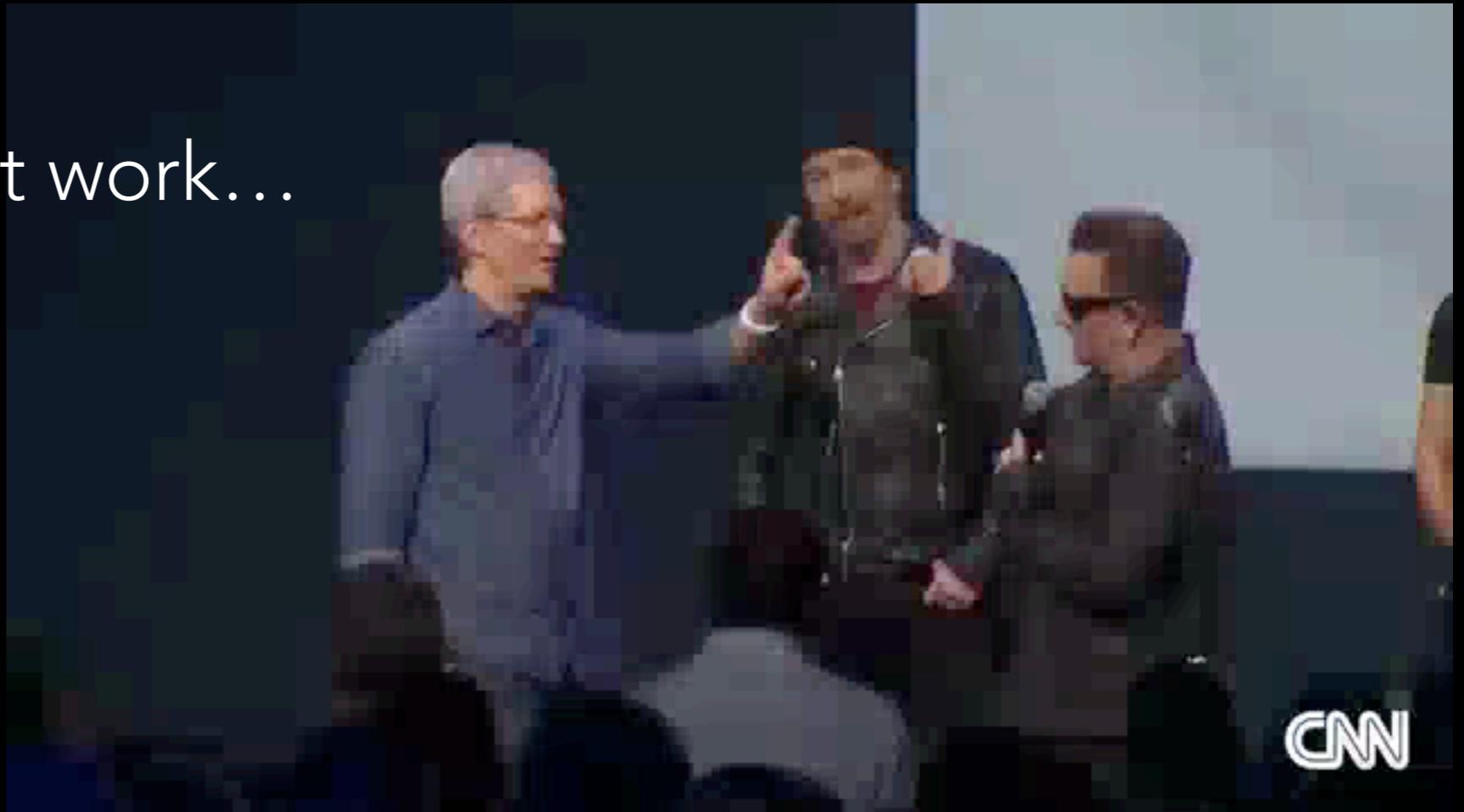
```
Stack<Apple> apples;  
apples.push(macintosh);
```

C++ templates

- Two instances of Stack class
 - One per type
- Does not reuse code
 - By default
- Permit optimisations based on types
 - For ex. reuse code for all pointers to objects
- Type safety, verified at compile time

Type safety

- `singers.push(apple);` **// compilation error**
- `apples.push(U2);`
 - may or may not work...



Optimisation based on types

- Generate different code based on types (template parameters)

```
template<typename T>
class MyClass
{
    ...
    enable_if_t<is_pointer<T>::value, T>
    member_function(T t) { ... };

    ...
};
```

- `member_function` is only defined if `T` is a pointer type
- (warning: C++14 code, not C++11)

C++ metaprogramming

- Programs that manipulate or produce programs
- Subset of C++
- Turing-complete (~ full programming language)
- Close to Functional programming
- Part of C++ standards
 - Major enhancements in C++11 et C++14

Application 1 - Strings literals obfuscation

- original string is source code
- original string in DEBUG builds
- developer-friendly syntax
- no trace of original string in compiled code in RELEASE builds

1st implementation

```
template<int... Indexes>
struct MetaString1 {
    constexpr MetaString1(const char* str)
        : buffer_{encrypt(str[Indexes])...} {}
```

const char* decrypt();

RUNTIME

private:

```
constexpr char encrypt(char c) const { return c ^ 0x55; }
constexpr char decrypt(char c) const { return encrypt(c); }
```

private:

```
char buffer_[sizeof...(Indexes) + 1];
};
```

1st implementation - Usage

```
#define OBFUSCATED1(str) (MetaString1<0, 1, 2, 3, 4, 5>(str).decrypt())  
  
cout << OBFUSCATED1("Britney Spears") << endl;
```

1st implementation - Problem

- List of indexes is hard-coded
 - 0, 1, 2, 3, 4, 5
- As a consequence, strings are truncated!

2nd implementation

- Generate a list of indexes with metaprogramming
- C++14 introduces **std:index_sequence**
- With C++11, we have to implement our own version
 - Very simplified
 - **MakelIndex<N>::type** generates:
 - **Indexes<0, 1, 2, 3, ..., N>**

2nd implementation

- Instead of:

MetaString1<0, 1, 2, 3, 4, 5>(str)

- we have:

MetaString2<**Make_Indexes<sizeof(str)-1>::type**>(str)

2nd implementation - Usage

```
cout << OBFUSCATED2("Katy Perry") << endl;
```

- No more truncation

3rd implementation

- In previous implementations, key is hard-coded

```
constexpr char encrypt(char c) const { return c ^ 0x55; }
```

- New template parameter for Key

```
template<int... I, int K>
struct MetaString3<Indexes<I...>, K>
```

Generating (pseudo-) random numbers

- C++11 includes `<random>`, but for runtime, not compile time
- **MetaRandom<N, M>**
 - N**: Nth generated number
 - M**: Maximum value (excluded)
- Linear congruential engine
 - Park-Miller (1988), “minimal standard”
 - Not exactly a uniform distribution (modulo operation)
 - Recursive

Seed

- **template<>**
struct MetaRandomGenerator<0> {
 static const int value = seed;
};
- How to choose an acceptable compile-time seed?
- Macros (C & C++):
 - **_TIME_**: compilation time (standard)
 - **_COUNTER_**: incremented each time it is used
(non-standard but well supported by compilers)

3rd implementation

- Different key for each string
 - thanks to **_COUNTER_**
- Different keys for each compilation
 - thanks to **_TIME_**
 - Sometimes not desirable: can give hints to attackers (differences between versions)

4th implementation

- Different and random keys, great!
- Why not go even further?
- Choose a different encryption algorithm, randomly!

4th implementation

- Template partial specialization
- ```
template<int A, int K, typename Indexes>
struct MetaString4;
```
- ```
template<int K, int... I>
struct MetaString4<0, K, Indexes<I...>> { ... c ^ K ... };
```
- ```
template<int K, int... I>
struct MetaString4<1, K, Indexes<I...>> { ... c + K ... };
```
- ```
#define DEF_OBFUSCATED4(str)
MetaString4<MetaRandom<__COUNTER__, 2>::value, ...
```

Result

- **Without obfuscation**

```
cout << "Britney Spears" << endl;
```

- **With obfuscation**

```
cout << OBFUSCATED4("Britney Spears") << endl;
```

Without obfuscation

```
_main proc near
    push    rbp
    mov     rbp, rsp
    mov     rdi, cs:_ZNSt3__14coutE_ptr
    lea     rsi, aBritneySpears ; "Britney Spears"
    call    __ZNSt3__1lsINS_1lchar_traitsIcEEEERNS_1:
    xor     eax, eax
    pop     rbp
    retn
_main    endp
```

Address	Length	Type	String
's' HEADER:0000000100000504	0000000E	C	/usr/lib/dyld
's' HEADER:0000000100000580	00000018	C	/usr/lib/libc++.1.dylib
's' HEADER:00000001000005B0	0000001B	C	/usr/lib/libSystem.B.dylib
's' __cstring:0000000100000F4C	0000000F	C	Britney Spears
's' __eh_frame:0000000100000FE9	00000005	C	zPLR

With obfuscation

Encrypted
characters
(mixed with MOV)

Decryption

```
sub_100000890 proc near
var_38= byte ptr -38h
var_37= byte ptr -37h
var_36= byte ptr -36h
var_35= byte ptr -35h
var_34= byte ptr -34h
var_33= byte ptr -33h
var_32= byte ptr -32h
var_31= byte ptr -31h
var_30= byte ptr -30h
var_2F= byte ptr -2Fh
var_2E= byte ptr -2Eh
var_2D= byte ptr -2Dh
var_2C= byte ptr -2Ch
var_2B= byte ptr -2Bh
var_2A= byte ptr -2Ah
var_29= byte ptr -29h
var_28= byte ptr -28h
var_20= qword ptr -20h

55      push    rbp
48 89 E5  mov     rbp,  rsp
41 57    push    r15
41 56    push    r14
53      push    rbx
48 83 EC 28  sub    rsp,  28h
4C 8B 3D 84 07+mov  r15,  cs:_stack_chk_guard_ptr
49 8B 07    mov    rax,  [r15]
48 89 45 E0  mov    [rbp+var_20],  rax
C6 45 C8 C9  mov    [rbp+var_38],  0C9h
48 8D 75 C9    lea    rsi,  [rbp+var_37]
C6 45 C9 8B  mov    [rbp+var_37],  8Bh
C6 45 CA BB  mov    [rbp+var_36],  0BBh
C6 45 CB AO  mov    [rbp+var_35],  0AOh
C6 45 CC BD  mov    [rbp+var_34],  0BDh
C6 45 CD A7  mov    [rbp+var_33],  0A7h
C6 45 CE AC  mov    [rbp+var_32],  0ACh
C6 45 CF B0  mov    [rbp+var_31],  0B0h
C6 45 D0 E9  mov    [rbp+var_30],  0E9h
C6 45 D1 9A  mov    [rbp+var_2F],  9Ah
C6 45 D2 B9  mov    [rbp+var_2E],  0B9h
C6 45 D3 AC  mov    [rbp+var_2D],  0ACh
C6 45 D4 A8  mov    [rbp+var_2C],  0A8h
C6 45 D5 BB  mov    [rbp+var_2B],  0BBh
C6 45 D6 BA  mov    [rbp+var_2A],  0BAh
31 C9    xor    ecx,  ecx
B8 01 00 00 00  mov    eax,  1

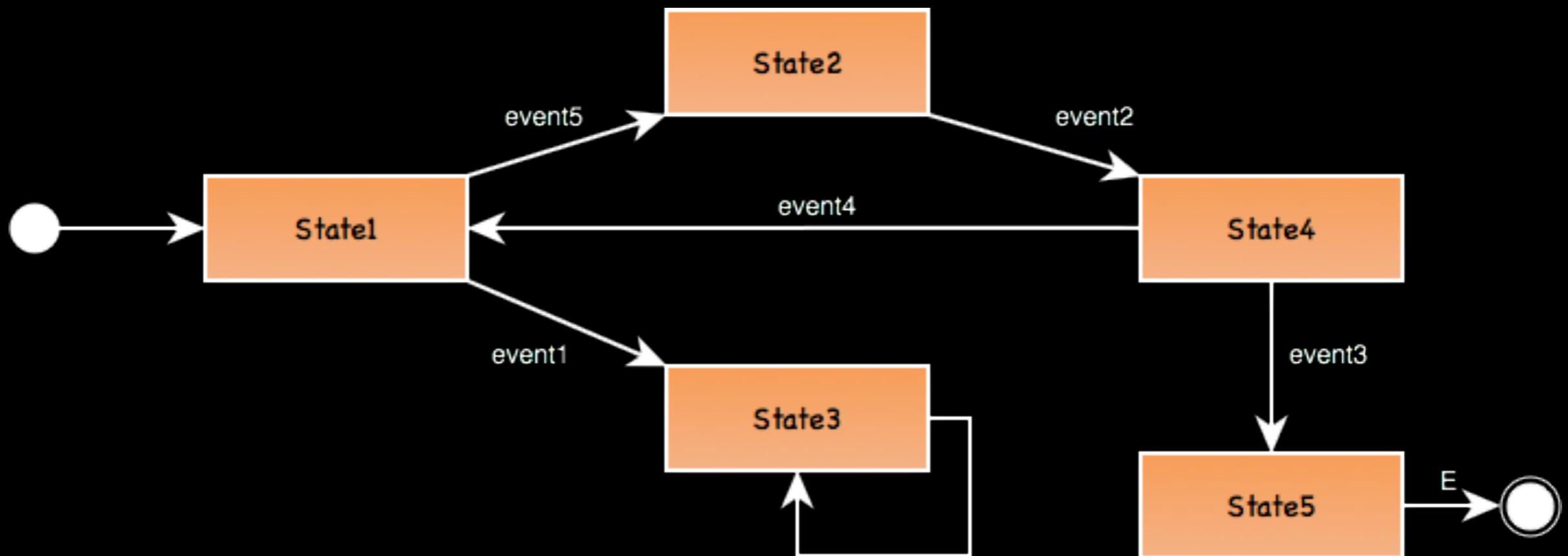
loc_1000008F2:
C6 44 0D D7 00  mov    [rbp+rcx+var_29],  0
48 FF C1    inc    rcx
48 83 F9 01  cmp    rcx,  1
75 F2    jnz    short loc_1000008F2

loc_100000900:
8A 4D C8    mov    cl,  [rbp+var_38]
30 4C 05 C8  xor    [rbp+rax+var_38],  cl
48 FF C0    inc    rax
48 83 F8 0F  cmp    rax,  0Fh
75 F0    jnz    short loc_100000900
```

Application 2 - Obfuscate calls

- How to obfuscate call such as:
 - **function_to_protect();**
- against static analysis (or even dynamic analysis)?

Finite State Machine (simple example)



Boost Meta State Machine (MSM) library

```
// --- Transition table
struct transition_table : mpl::vector<
    // Start   Event      Next     Action           Guard
    // +-----+-----+-----+-----+-----+
    Row < State1 , event5   , State2   , Action1  , Guard1
    Row < State1 , event1   , State3   , Action2  , Guard2
    // +-----+-----+-----+-----+-----+
    Row < State2 , event2   , State4   , Action3  , Guard3
    // +-----+-----+-----+-----+-----+
    Row < State3 , none     , State3   , Action4  , Guard4
    // +-----+-----+-----+-----+-----+
    Row < State4 , event4   , State1   , Action5  , Guard5
    Row < State4 , event3   , State5   , Action6  , Guard6
    // +-----+-----+-----+-----+-----+
    Row < State5 , E        , Final    , CallTarget, Guard7
    // +-----+-----+-----+-----+-----+
> {};
```

types

template parameter

Compile time entity
Generates code (FSM) at compile-time

Result

- **Without obfuscation**

```
function_to_protect("did", "again");
```

- **With obfuscation**

```
OBFUSCATED_CALL(function_to_protect, "did", "again");
```

- **Even better**

```
OBFUSCATED_CALL(function_to_protect,  
OBFUSCATED("did"), OBFUSCATED("again"));
```

Without obfuscation

```
sub_10000160E proc near
55          push    rbp
48 89 E5    mov     rbp, rsp
E8 E9 FD FF FF    call    sub_100001400
48 8D 3D 49 66+   lea     rdi, aDid      ; "did"
48 8D 35 46 66+   lea     rsi, aAgain    ; "again"
5D          pop     rbp
E9 C7 FE FF FF    jmp     sub_1000014F2
sub_10000160E endp
```

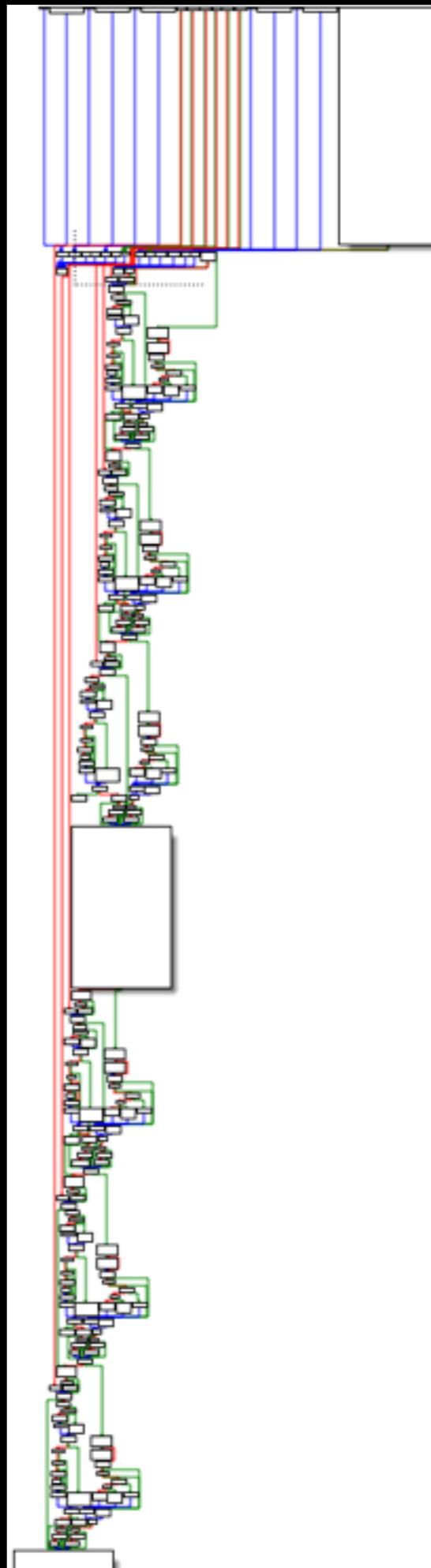
With obfuscation

```
48 C7 85 44 FF+          mov    [rbp+var_BC], 0
48 C7 85 3C FF+          mov    [rbp+var_C4], 0
48 8D BD E8 FE+          lea    rdi, [rbp+var_118]
48 8D B5 38 FF+          lea    rsi, [rbp+var_C8]
E8 49 43 00 00           call   sub_100005A2A
C7 85 EC FE FF+          mov    [rbp+var_114], 0
48 8D BD E8 FE+          lea    rdi, [rbp+var_118]
48 8D 75 D0               lea    rsi, [rbp+var_30]
BA 01 00 00 00           mov    edx, 1
E8 2E 46 00 00           call   sub_100005D2E
BB 45 00 00 00           mov    ebx, 45h
4C 8D AD E8 FE+          lea    r13, [rbp+var_118]
4C 8D 75 C8               lea    r14, [rbp+var_38]
4C 8D 7D C0               lea    r15, [rbp+var_40]
4C 8D 65 B8               lea    r12, [rbp+var_48]

loc_100001718:          ; CODE XREF: sub_10000163A+101:j
4C 89 EF                 mov    rdi, r13
4C 89 F6                 mov    rsi, r14
E8 2F 0C 00 00           call   sub_100002352
4C 89 EF                 mov    rdi, r13
4C 89 FE                 mov    rsi, r15
E8 40 0D 00 00           call   sub_10000246E
4C 89 EF                 mov    rdi, r13
4C 89 E6                 mov    rsi, r12
E8 51 0E 00 00           call   sub_10000258A
FF CB
75 DB
48 8D BD E8 FE+          jnz   short loc_100001718
48 8D 75 B0               lea    rdi, [rbp+var_118]
E8 05 0C 00 00           lea    rsi, [rbp+var_50]
call   sub_100002352
48 8D BD E8 FE+          lea    rdi, [rbp+var_118]
48 8D 75 A8               lea    rsi, [rbp+var_58]
E8 11 0D 00 00           call   sub_10000246E
48 8D BD E8 FE+          lea    rdi, [rbp+var_118]
48 8D 75 A0               lea    rsi, [rbp+var_60]
E8 39 0F 00 00           call   sub_1000026A6
48 8D 05 64 FE+          lea    rax, loc_1000015D7+1
48 89 45 88               mov    [rbp+var_78], rax

loc_100001778:          ; DATA XREF: sub_10000163A+332:o
C7 45 90 B8 01+           mov    [rbp+var_70], 1B8h
48 8D BD E8 FE+          lea    rdi, [rbp+var_118]
48 8D 75 88               lea    rsi, [rbp+var_78]
BA 01 00 00 00           mov    edx, 1
E8 42 43 00 00           call   sub_100005AD6
48 8D BD F0 FE+          lea    rdi, [rbp+var_110]
E8 D4 15 00 00           call   sub_100002D74
48 8D BD 40 FF+          lea    rdi, [rbp+var_C4+4]
E8 C8 15 00 00           call   sub_100002D74
C7 85 D8 FD FF+          mov    [rbp+var_228], 0F020F6Bh
31 C9
B8 01 00 00 00           xor    ecx, ecx
mov    eax, 1

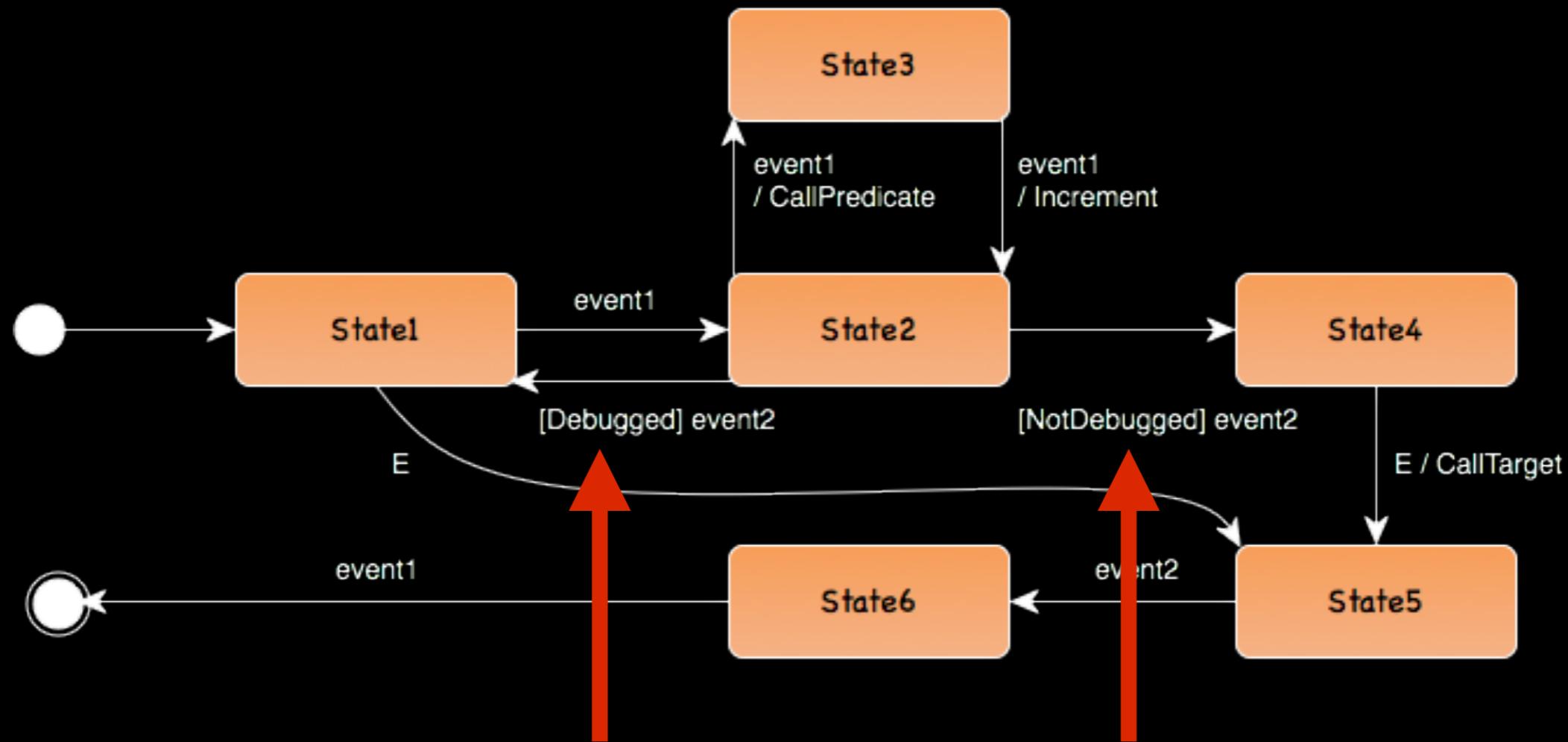
loc_1000017BD:          ; CODE XREF: sub_10000163A+192:j
C6 84 0D DC FD+           mov    [rbp+rcx+var_224], 0
48 FF C1
48 83 F9 01               inc    rcx
75 EF                     jnz   short loc_1000017BD
```



Application 3: FSM + Debugger Detection

- FSM
 - To fight against static analysis
- Debugger detection
 - To fight against dynamic analysis

Finite State Machine



- Follows a different path depending of a predicate (Debugged or not Debugged, that is the question)

More obfuscation

- Obfuscate predicate result
 - Avoid simple “if”, too simple for reverse engineers
 - Make computation instead
 - If the example, counter is odd if predicate is false

More obfuscation

- Obfuscate function address
 - Otherwise, IDA is smart enough to get it
 - Simply make some computation on address
 - Using MetaRandom (like for strings obfuscation)

Predicate

- Debugger detection is only an example
 - In the example, implemented only for Mac OS X / iOS
- Virtual environment detection
- Jailbreak detection
- Etc

Compilers support

Compiler	Compatible	Remark
Apple LLVM 5.1	Yes	Previous versions not tested
Apple LLVM 6.0	Yes	Xcode 6, 6.1 beta
LLVM 3.4, 3.5	Yes	Previous versions not tested
GCC 4.8.2 or higher	Yes	Previous versions not tested Compile with -std=c++11
Intel C++ 2013	Yes	Version 14.0.3 (2013 SP1 Update 3)
Visual Studio 2013	No	Lack of constexpr support
Visual Studio 14	Almost	Not far, lack init of arrays CTP3 tested

White paper

C++11 metaprogramming applied to software obfuscation

Black Hat Europe 2014 - Amsterdam

Sebastien Andrivet - [sebastien@andrivet.com](mailto:sbastien@andrivet.com), @AndrivetSeb
Senior Security Engineer, SCRT Information Security, www.scrt.ch
CTO, ADVTOOLS SARL, www.advtools.com

Abstract. The C++ language and its siblings like C and Objective-C are ones of the most used languages¹. Significant portions of operating systems like Windows, Linux, Mac OS X, iOS and Android are written in C and C++. There is however a fact that is little known about C++: it contains a Turing-complete sub-language executed at compile time. It is called C++ template metaprogramming (not to be confounded with the C preprocessor and macros) and is close to functional programming.

This white paper will show how to use this language to generate, at compile time, obfuscated code without using any external tool and without modifying the compiler. The technics presented rely only on C++11, as standardized by ISO². It will also show how to introduce some form of randomness to generate polymorphic code and it will give some concrete examples like the encryption of strings literals.

Keywords: software obfuscation, security, encryption, C++11, metaprogramming, templates.

Introduction

In the past few years, we have seen the comeback of heavy clients and of client-server model. This is in particular true for mobile applications. It is also the return of off-line modes of operation with Internet access that is not always reliable and fast. On the other hand, we are far more concerned about privacy and security than in the old times and mobile phones or tablets are easier to steal or to lose than desktops or laptops. We have to protect secrets locally. In some cases, we also need to protect intellectual property (for example when using DRM systems) knowing that we are giving a lot of information to the attacker, in particular a lot of binary code. This is different from the web application model where critical portions of code are executed exclusively on the server, behind firewalls and IDS/IPS (at least until HTML5).

We have thus to protect software in a hostile environment and obfuscation is one of the tools available to achieve this goal, even if it is far from a bullet-proof solution. Popular software such as Skype is using obfuscation like the majority of DRM (Digital Rights Management) systems and several viruses (to slow down their study).

Obfuscation

Obfuscation is "the deliberate act of creating [...] code that is difficult for humans to understand"³. Obfuscated code has the same or almost the same semantics than the original and obfuscation is transparent for the system executing the application and for the users of this application.

Credits

- I take inspiration from many sources
- In particular:
 - Samuel Neves & Filipe Araujo
 - LeFF (virus maker)

Binary code obfuscation through C++ template metaprogramming

Samuel Neves and Filipe Araujo

CISUC, Department of Informatics Engineering
University of Coimbra, Portugal
{sneves,filipt}@dei.uc.pt

Abstract. Defending programs against illegitimate use and tampering has become both a field of study and a large industry. Code obfuscation is one of several strategies to stop, or slow down, malicious attackers from gaining knowledge about the internal workings of a program.

Binary code obfuscation tools often come in two (sometimes overlapping) flavors. On the one hand there are “binary protectors”, tools outside of the development chain that translate a compiled binary into another, less intelligible one. On the other hand there are software development kits that require a significant effort from the developer to ensure the program is adequately obfuscated.

In this paper, we present obfuscation methods that are easily integrated into the development chain of C++ programs, by using the compiler itself to perform the obfuscated code generation. This is accomplished by using advanced C++ techniques, such as operator overloading, template metaprogramming, expression templates, and more. We achieve obfuscated code featuring randomization, opaque predicates and data masking. We evaluate our obfuscating transformations in terms of potency, resilience, stealth, and cost.

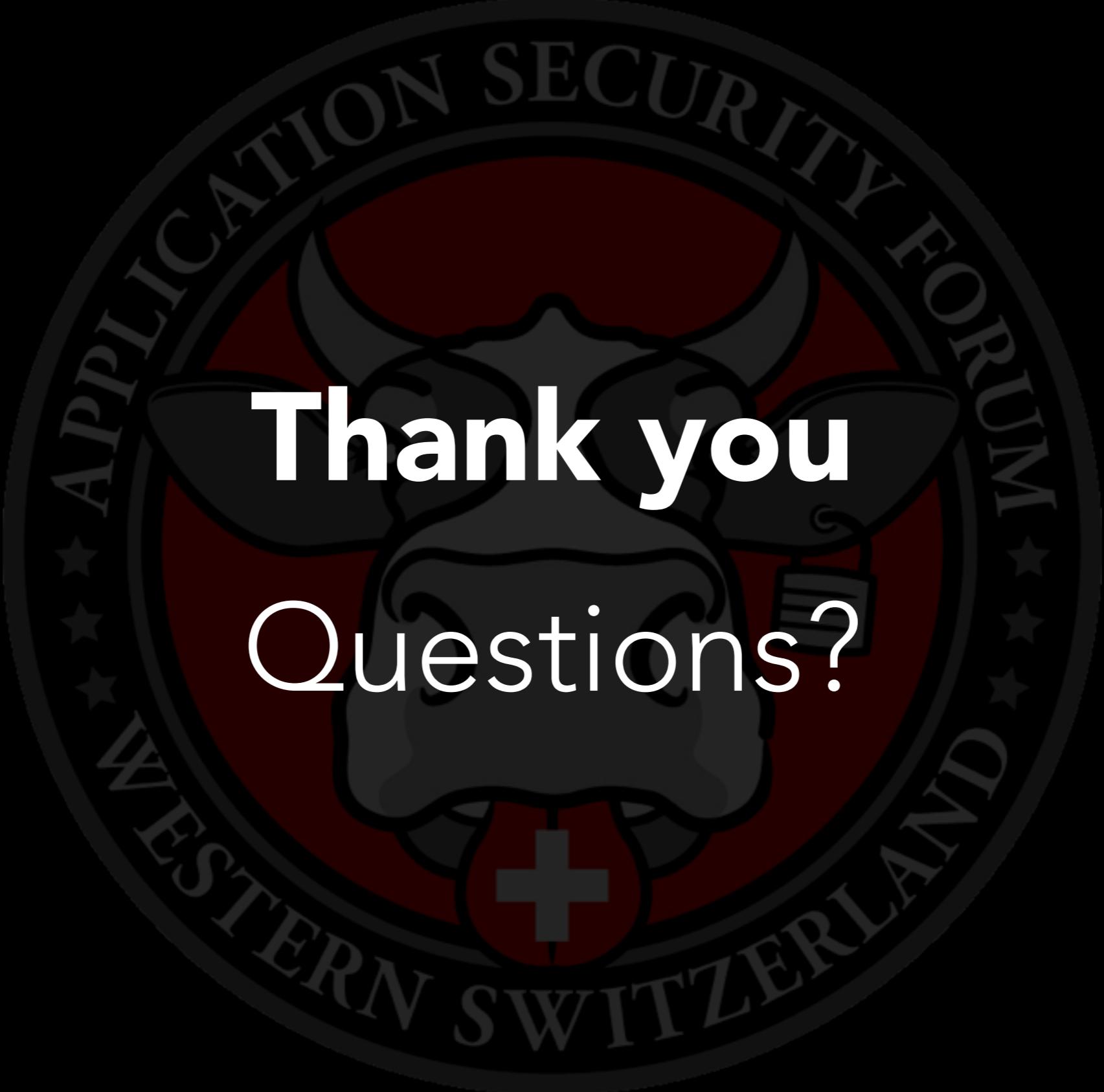
1 Introduction

Today, the Internet is the major channel for software distribution. However, software available online may not only attract legitimate users, but malicious agents as well (e.g., pirates, competitors, etc). To fight these agents vendors may resort to diverse mechanisms including legal suites, and technical measures. Technical measures consist of transformations that render the code harder or (ideally) impossible to reverse engineer. One particularly effective measure is to move critical areas of code out of the reach of the attacker, by e.g. offloading execution to a remote server. It is not, however, always possible to offload critical bits of an application elsewhere, leading to alternative “white-box” techniques to thwart analysis.

Obfuscation has been often used as such an anti-reverse engineering measure [10]. Formally, an obfuscating transform \mathcal{O} is a function that takes in a program P and outputs a program $\mathcal{O}(P)$ with the same semantics, but somehow “harder” to understand. Although general obfuscators cannot exist [5], secure

Infos

- @AndrivetSeb sebastien@andrivet.com
- All code presented here is available on GitHub
 - <https://github.com/andrivet/ADVobfuscator>
 - Contains
 - obfuscator (in source)
 - examples
 - Whitepaper
 - BSD 3-clauses license



The logo is circular with a dark red background. The top half features a stylized illustration of a human head profile facing left, with a red cross on the forehead. The bottom half shows a red shield with a white cross. The text "APPLICATION SECURITY FORUM" is at the top and "WESTERN SWITZERLAND" is at the bottom, both in a serif font. A ring of small white stars surrounds the central figure.

Thank you
Questions?