SEBASTIEN ANDRIVET

C++11 METAPROGRAMMING APPLIED TO SOFTWARE OBFUSCATION

About me

Since June 2014
Senior Security Engineer
at **SCRT** (Switzerland)

Sebastien ANDRIVET

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

PROBLEM

Problem

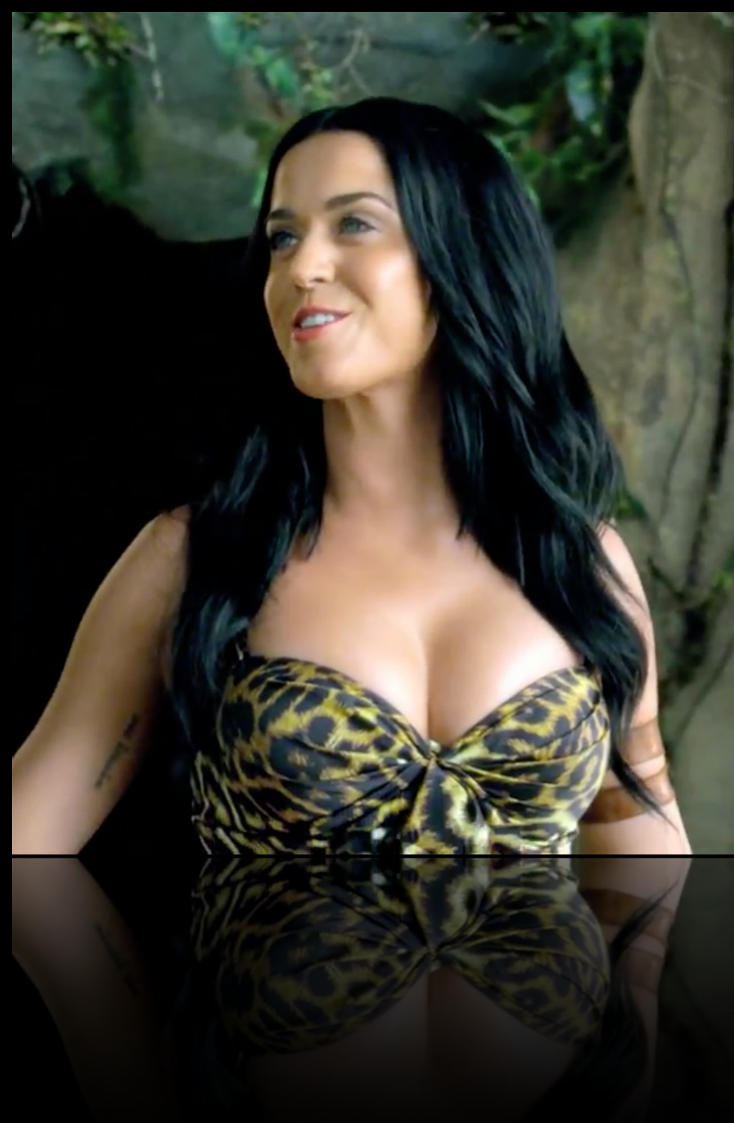
- Reverse engineering of an application if often like following the “white rabbit”
  - i.e. following string literals
- Live demo
  - Reverse engineering of an application using IDA

A SOLUTION

OBFUSCATION

What is Obfuscation?

Obfuscator $\mathcal{O}$

$\mathcal{O}($  $)$ $=$ 

YES! It is also Katy Perry!

- Same semantics
- Obfuscated

Obfuscation

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

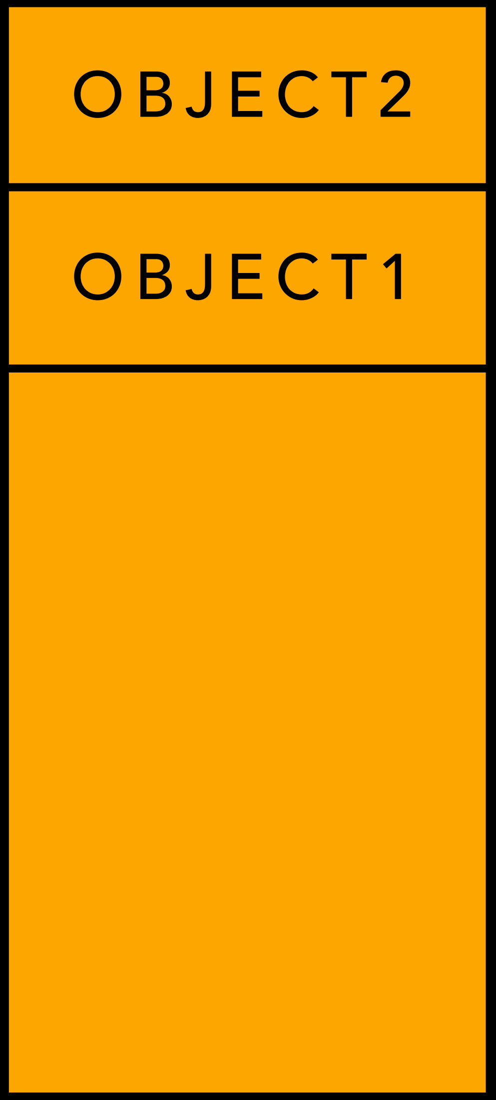
–WIKIPEDIA, APRIL 2014

Obfuscators classes

- Transformation of source code
  - Manual transformation by programmers
  - Pre-processors
  - **C++ template instantiation**
  - Abstract Syntax Tree (AST) transformation
- Transformation of binary code

C++ templates

- Example: Stack of objects
    - Push
    - Pop



Simple stack

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

- ```
  Stack stack;
  Singer* britney_spears = new Singer();
  stack.push(britney_spears);
  …
  ```

- ```
  Singer* singer = stack.pop();
  singer->sing();  // singer = britney_spears
  ```


  ```

Simple stack

- **Apple*** apple = new **Apple**();
  stack.push(apple);
  …
- **Singer*** justin = stack.pop();
  justin->sing();
- Crash at **runtime**!
  We pop up an **Apple**, not a **Singer**!

C++ template: type safety

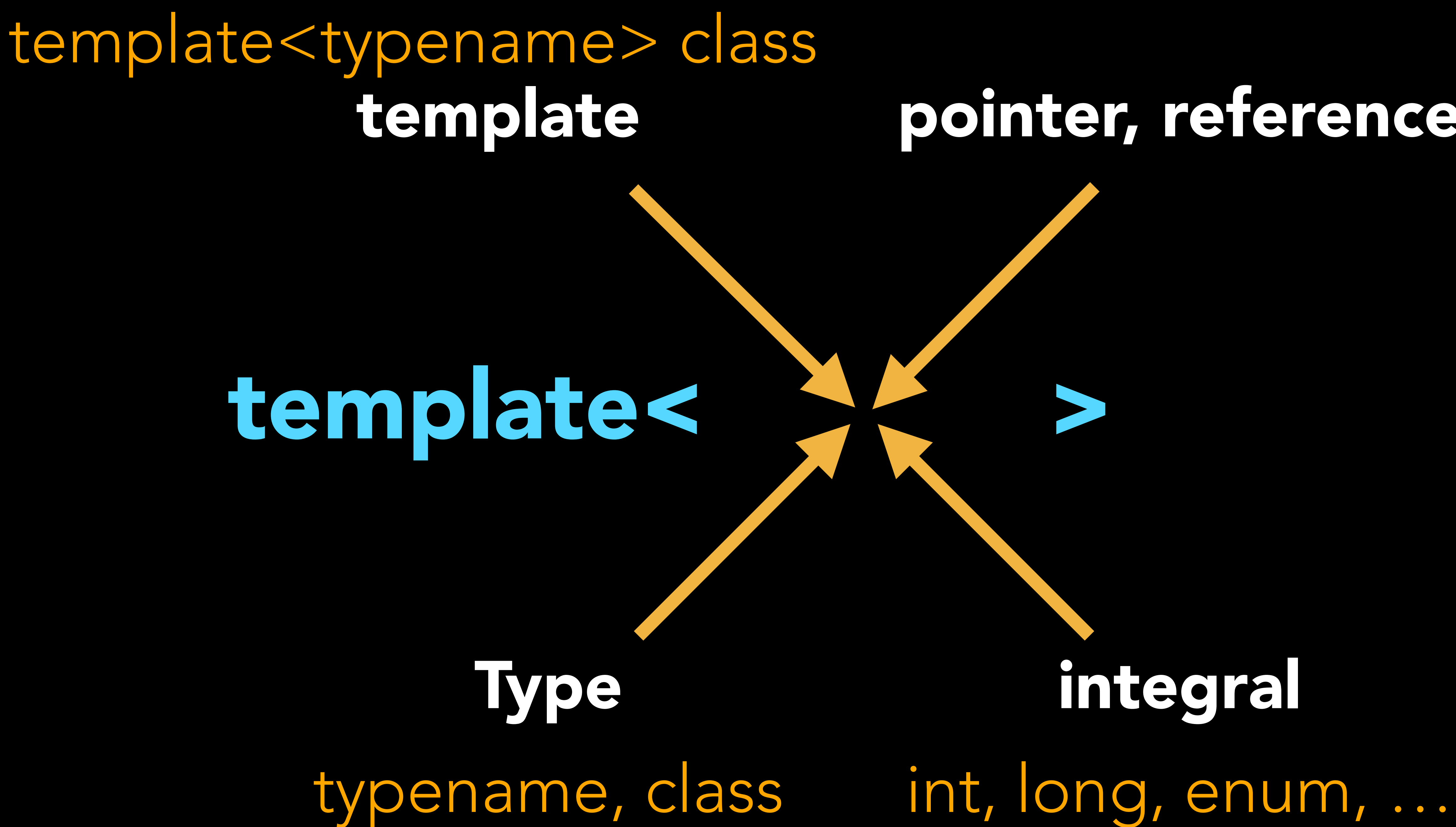
```
template<typename T>
struct Stack
{
    void push(T* object);
    T* pop();
};
```

```
Stack<Singer> stack;
stack.push(new Apple());    // compilation error
```

Templates kinds

- Functions templates
- **Class templates**
- (variables templates: C++14)

Template parameters



Integral parameters

- Example: Fibonacci sequence $F_n = F_{n-1} + F_{n-1}$ $F_0 = 0$ $F_1 = 1$
  0, 1, 1, 2, 3, 5, 8, …

- ```
  template<int N>
  struct Fibonacci {
      static constexpr int value =
          Fibonacci<N-1>::value + Fibonacci<N-2>::value;
  };
  ```

- ```
  template<> struct Fibonacci<1> { static constexpr int value = 1; };
  template<> struct Fibonacci<0> { static constexpr int value = 0; };
  ```

- ```
  cout << Fibonacci<20>::value << endl;
  ```
  ```

Usage

- Not the same than:
  ```
  int fibonacci(int n) {
      return n <= 1 ? n : fibonacci(n-1) + fibonacci(n-2); }
  ```
- **fibonacci(5)** is executed at runtime
- **Fibonacci<5>** is evaluated at compile time
  - no opcode generated, no CPU cycle at runtime

C++ metaprogramming

- Programs that manipulate or produce programs
- Subset of C++
- Turing-complete
- Close to Functional programming
- Part of C++ standards

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();

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

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

    const char* decrypt();

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

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

    const char* decrypt();

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

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

    const char* decrypt();

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

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

    const char* decrypt();

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

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

buffer_{encrypt(str[0]), encrypt(str[1]), encrypt(str[2])}

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

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

    const char* decrypt();

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

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

    const char* decrypt();

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

```
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
  - **MakeIndex<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
- But not possible to use custom suffixes
  - Problem with sizeof

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)
  ```

3^{rd} implementation

- Different keys for each compilation
  - thanks to **__TIME__**
- Different key for each string
  - thanks to **__COUNTER__**

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 Key, typename Indexes>
  struct MetaString4;
- template<int K, int... I>
  struct MetaString4<**0**, K, Indexes<I...>> {};
- template<int K, int... I>
  struct MetaString4<**1**, K, Indexes<I...>> {};
- #define DEF_OBFUSCATED4(str)
  MetaString4<**MetaRandom<__COUNTER__, 2>::value**, …

Result

- **Without obfuscation**

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

- **With obfuscation**

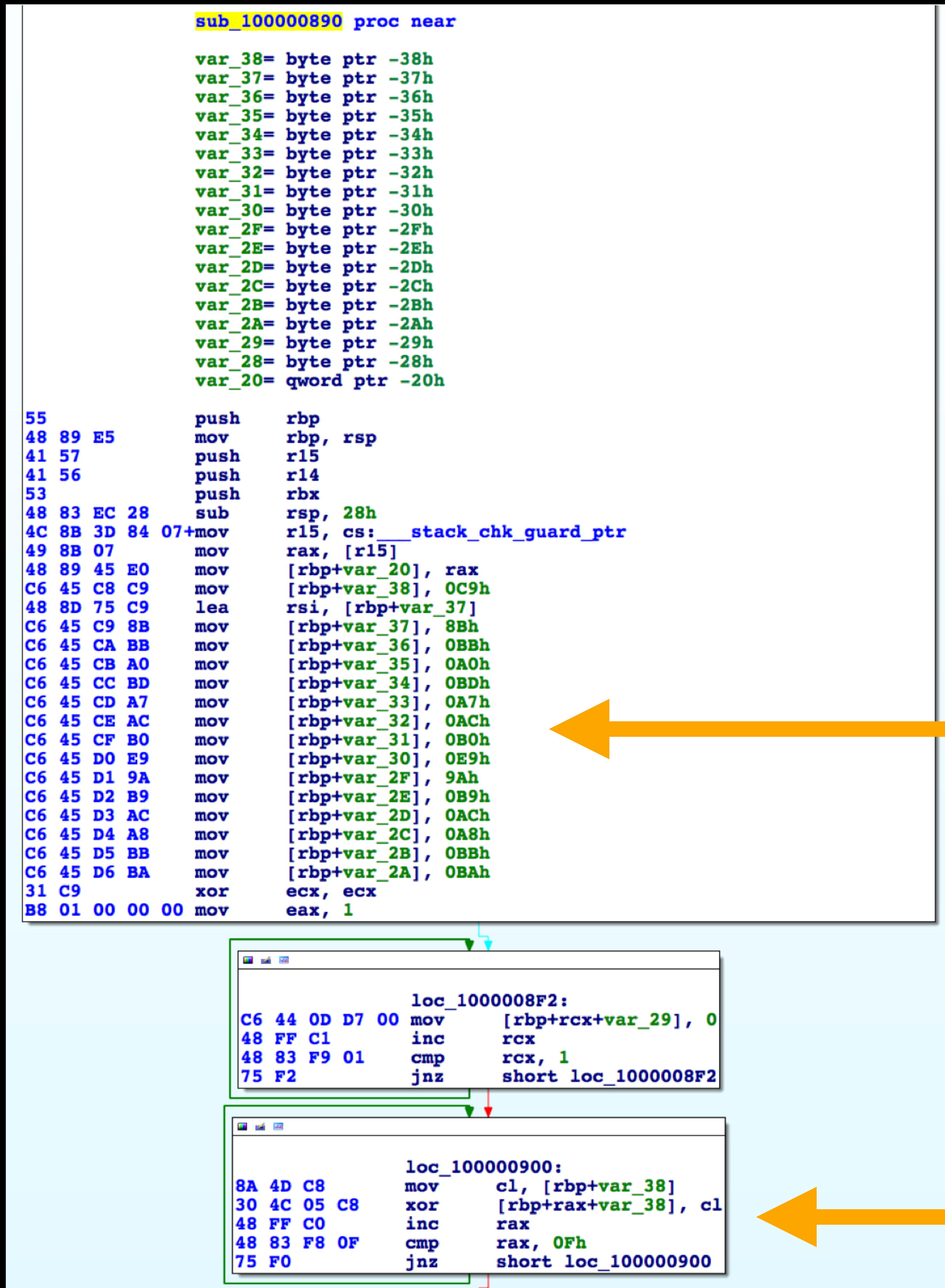
```
cout << OBFUSCATED4("Britney Spears") << endl;
```

Without obfuscation

```
_main               proc near
                    push    rbp
                    mov     rbp, rsp
                    mov     rdi, cs:__ZNSt3__14coutE_ptr
                    lea     rsi, aBritneySpears ; "Britney Spears"
                    call    __ZNSt3__11sINS_11char_traitsIcEEEERNS_13
                    xor     eax, eax
                    pop     rbp
                    retn
_main               endp
```

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

With obfuscation



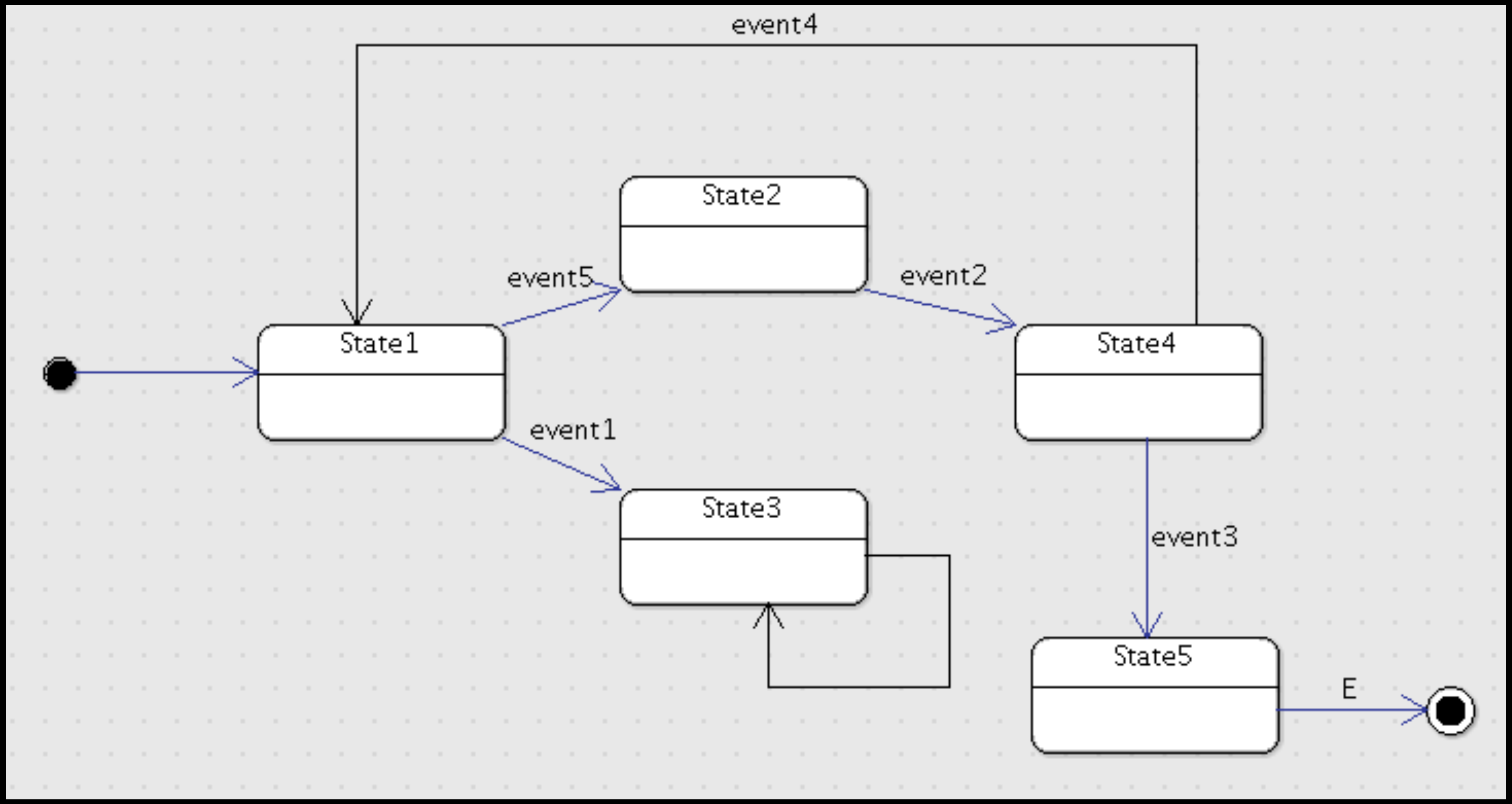
Encrypted characters (mixed with MOV)

Decryption

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

Compile time entity

- struct transition_table : mpl::vector<

```
//        Start     Event        Next      Action                Guard
//      +---------+------------+---------+---------------------+----------------------+
Row < State1 , event5      , State2                                                >,
Row < State1 , event1      , State3                                                >,
//      +---------+------------+---------+---------------------+----------------------+
Row < State2 , event2      , State4                                                >,
//      +---------+------------+---------+---------------------+----------------------+
Row < State3 , none        , State3                                                >,
//      +---------+------------+---------+---------------------+----------------------+
Row < State4 , event4      , State1                                                >,
Row < State4 , event3      , State5                                                >,
//      +---------+------------+---------+---------------------+----------------------+
Row < State5 , E           , Final    , State5ToFinal                              >
//      +---------+------------+---------+---------------------+----------------------+> { } ;
```

types

template parameter

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                          dec     ebx
75 DB                          jnz     short loc_100001718
48 8D BD E8 FE+                lea     rdi, [rbp+var_118]
48 8D 75 B0                    lea     rsi, [rbp+var_50]
E8 05 0C 00 00                 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                          xor     ecx, ecx
B8 01 00 00 00                 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                       inc     rcx
48 83 F9 01                    cmp     rcx, 1
75 EF                          jnz     short loc_1000017BD
```

Etc, etc, …

Go even further?

- Obfuscate function address
  - Otherwise, IDA is smart enough to get it
- Call target (function_to_protect) in the middle of FSM
- Select a FSM randomly from a set
- Combine transitions with debugger or VM detection
- Etc…

Compilers support

Compiler	Compatible	Remark
Apple LLVM 5.1	Yes	Previous versions not tested
Apple LLVM 6.0	Yes	Beta 2 (based on LLVM 3.5)
LLVM 3.4	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 2014	No	TP1 tested

Compilers options

- Use appropriate compiler options to generate a RELEASE build
  - Xcode: Deployment Postprocessing = Yes
  - GCC: -std=c++11 s -O3
- Otherwise, you will get a binary with original string literals inside
- Disabling RTTI (Runtime Type Information) generates an even more silent binary
  - But not compatible with MSM

Drawbacks

- Compiler has to be C++11 compliant
- Some parts of app has to be in C++ or Objective-C
- Increase executable size
- Increase compilation time (especially Intel compiler)
- Obfuscator complex to write and debug
- Some obfuscation techniques seems difficult to apply
    - control flow graph flattening

Benefits

- Does not rely on external tools, or modified compiler
- Independent of target platform
- High-level (source code). Allows complex obfuscation
- Applicable to environment such as iOS (AppStore)

My current researches

- More obfuscation areas and techniques
- Apply to Objective-C
  - selectors
- Apply to Android

Code

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

White paper

- On GitHub

C++11 metaprogramming applied to software obfuscation

Sebastien Andrivet - sebastien@andrivet.com, @AndrivetSeb
Senior Security Engineer, SCRT Information Security, www.scrt.ch

Abstract. The C++ language and its siblings like C and Objective-C are ones of the most used languages[1]. 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 ISO[2]. 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 mobiles phones or tablets are easier to steal or to loose 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 HMTL5).

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.

Contact

- @AndrivetSeb
- sebastien@andrivet.com

Thank you

Questions?