Reversing C++ - How not the get an heart attack

ERNW
providing security.

Agenda

- Motivation
- Reversing C++ at a Glance
- Dynamic Binary Instrumentation to the Rescue
- Solution

ERNW
providing security.

Reverse Engineering

- Find out how software works
  - Find vulnerabilities
  - Understand the software
  - ...
- Static and/or dynamic

ERNW
providing security.

C and Assembly side by side

```c
#include<stdio.h>

int main() {
 printf("Hello World\n");
 return 0;
}
```

```
; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near
; __unwind {
push    rbp
mov     rbp, rsp
lea     rdi, s          ; "Hello World"
call    _puts
mov     eax, 0
pop     rbp
retn
; } // starts at 1139
main endp
```

ERNW
providing security.

C++ and Assembly side by side

```
class Hello {
   public:
   Hello() {}
   virtual void printWorld() {cout << "Hello"; }
};

class HelloWorld : public Hello {
   public: HelloWorld() {}
   virtual void printWorld() {
      cout << "Hello World";
   }
};

int main() {
Hello *my_hello;
my_hello = new HelloWorld();
my_hello->printWorld();
return 0; }
```

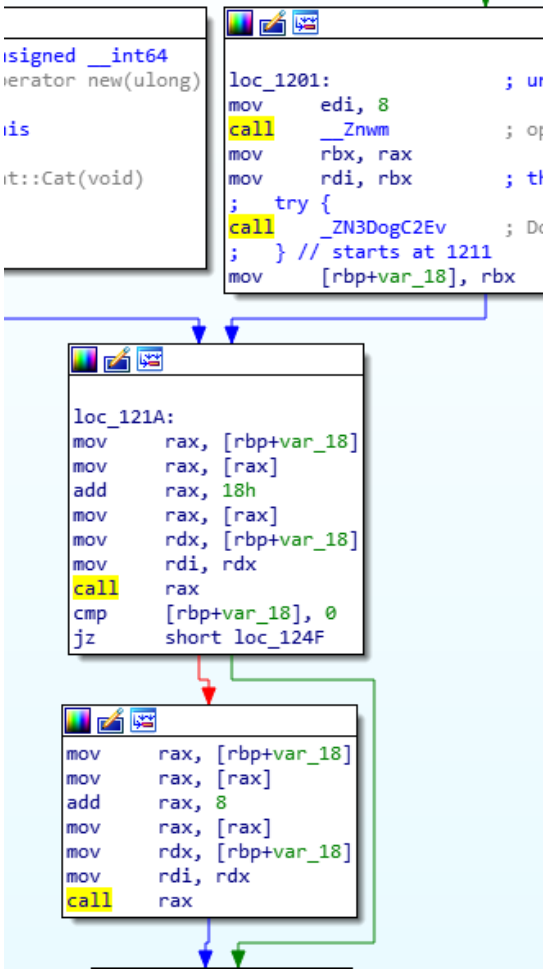
```
; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

var_18= qword ptr -18h

; __unwind {
push    rbp
mov     rbp, rsp
push    rbx
sub     rsp, 18h
mov     edi, 8          ; unsigned __int64
call    __Znwm          ; operator new(ulong)
mov     rbx, rax
mov     rdi, rbx        ; this
call    _ZN10HelloWorldC2Ev ; HelloWorld::HelloWorld(void)
mov     [rbp+var_18], rbx
mov     rax, [rbp+var_18]
mov     rax, [rax]
mov     rax, [rax]
mov     rdx, [rbp+var_18]
mov     rdi, rdx
call    rax
mov     eax, 0
add     rsp, 18h
pop     rbx
pop     rbp
retn
; } // starts at 1169
main endp
```

Reversing C++ at a Glance

ERNW
providing security.

Reversing C++ can be tedious

- Objects created dynamically
- (Multiple) inheritance support
- Information only available at runtime



Virtual Functions

- Functions can be overwritten by derived classes
- Without virtual keyword:
  - Early binding
  - Type is determined at compile time
- With virtual keyword
  - Late binding
  - Type is determined at runtime

Virtual functions

```
class Hello {
   public:
   Hello() {}
   virtual void printWorld() {cout << "Hello"; }
};

class HelloWorld : public Hello {
   public: HelloWorld() {}
   virtual void printWorld() {
      cout << "Hello World";
   }
};

int main() {
Hello *my_hello;
my_hello = new HelloWorld();
my_hello->printWorld();
return 0; }
```

- $ g++ test.cpp -o testcpp
- $ ./testcpp
  - Hello World

Virtual functions

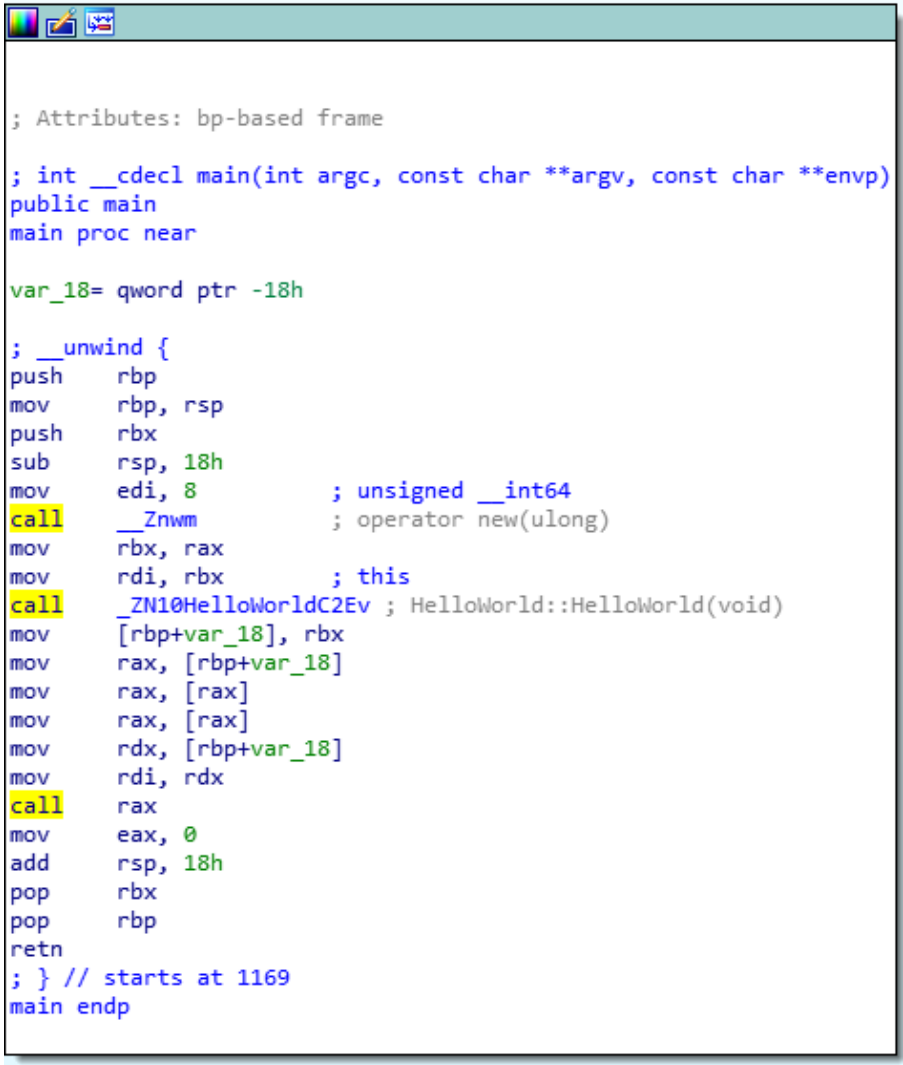
```
class Hello {
   public:
   Hello() {}
           void printWorld() {cout << "Hello"; }
};

class HelloWorld : public Hello {
   public: HelloWorld() {}
           void printWorld() {
      cout << "Hello World";
   }
};

int main() {
Hello *my_hello;
my_hello = new HelloWorld();
my_hello->printWorld();
return 0; }
```

- $ g++ test_no_virt.cpp -o test_no_virt
- $ ./test_no_virt
  - Hello

ERNW
providing security.

How virtual functions work

- Details depend on compiler
- Relies on Virtual function table (vtable)
  - Structure with pointers to virtual functions
  - Offset in table is accessed when function is called



```
; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

var_18= qword ptr -18h

; __unwind {
push    rbp
mov     rbp, rsp
push    rbx
sub     rsp, 18h
mov     edi, 8          ; unsigned __int64
call    __Znwm          ; operator new(ulong)
mov     rbx, rax
mov     rdi, rbx        ; this
call    _ZN10HelloWorldC2Ev ; HelloWorld::HelloWorld(void)
mov     [rbp+var_18], rbx
mov     rax, [rbp+var_18]
mov     rax, [rax]
mov     rax, [rax]
mov     rdx, [rbp+var_18]
mov     rdi, rdx
call    rax
mov     eax, 0
add     rsp, 18h
pop     rbx
pop     rbp
retn
; } // starts at 1169
main endp
```

Cross References

- Fast method to reconstruct call graph
  - What does a function call
  - From where is this function called
- Eases navigation in disassembled binary

Vtables I

```
class Hello {
   public:
   Hello() {}
   virtual void printWorld()
     {cout << "Hello"; }
};

class HelloWorld : public Hello {
   public: HelloWorld() {}
   virtual void printWorld() {
      cout << "Hello World";
   }
};

int main() {
Hello *my_hello;
my_hello = new HelloWorld();
my_hello->printWorld();
return 0; }
```

```
vtable for HelloWorld:
        .quad 0
        .quad typeinfo for HelloWorld
        .quad HelloWorld::printWorld()
vtable for Hello:
        .quad 0
        .quad typeinfo for Hello
        .quad Hello::printWorld()
```

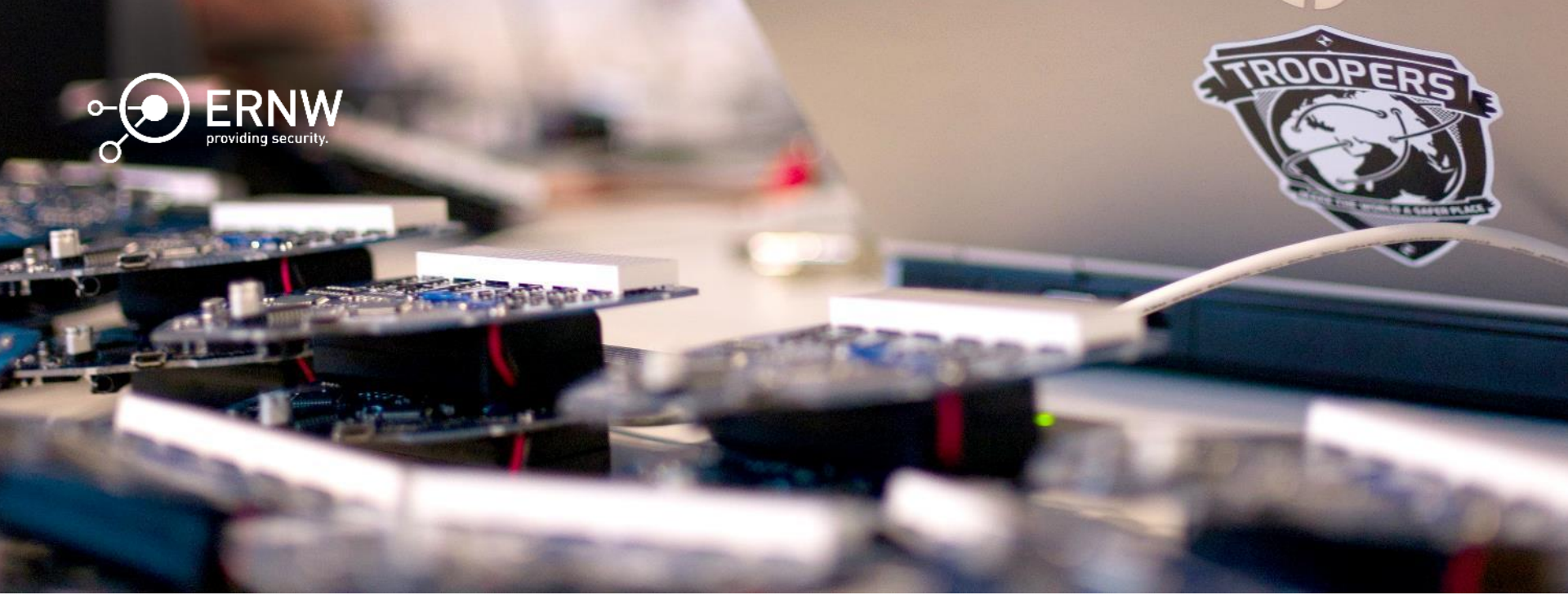
Vtables II

```
class Hello {
  public:
  Hello() {}
  virtual void fHello()
    {cout << "frmHello";}
  virtual void printWorld()
    {cout << "Hello"; }
};
class HelloWorld : public Hello {
  public:
  HelloWorld() {}
  virtual void printWorld()
    { cout << "Hello World"; }
  virtual void vfWorld()
    {cout << "frmHelloWorld";}
  void fWorld()
    {cout << "frmHelloWorld";}
};
```

```
vtable for HelloWorld:
        .quad 0
        .quad typeinfo for HelloWorld
        .quad Hello::fHello()
        .quad HelloWorld::printWorld()
        .quad HelloWorld::vfWorld()
vtable for Hello:
        .quad 0
        .quad typeinfo for Hello
        .quad Hello::fHello()
        .quad Hello::printWorld()
```

Hybrid Approach

- Static analysis only is not sufficient
- We need runtime information
  - Use dynamic acquired information to help static analysis



Dynamic Binary Instrumentation (DBI) to the Rescue

What is DBI

- Instrument binaries dynamically
- Aka dynamic binary rewriting
- Think JIT-compiler, but for machine code
- Inject custom instrumentation at interesting points
  - On end/beginning of basic block
  - On every instruction
  - On function entry/exit
  - On call instruction

Debugger vs DBI

Debugger	Dynamic Binary Instrumentation
Slow – Lots of context switches	Faster – Fewer switches
Build for debugging applications	Build with custom extendibility in mind
Coarse granularity	Fine granularity
Simpler to use	Harder to use

DBI Framework Overview

	Frida	Intel Pin	DynamoRio
Licence	OSS Friendly	Proprietary Software Free of charge	BSD Licence
Language(s)	JavaScript Python	C/C++	C/C++
Maturity	Immature	Mature	Mature
Documentation	Not so good	Okay, lots of examples	Okay, lots of examples
Granularity	Suffcient	Good	Good
Architectures	X86, ARM, MIPS	X86	X86, ARM

Solution – Introducing Devi

ERNW
providing security.

Devi's Architecture

- Trace execution of binary with Frida
  - Instrument every call instruction
  - Check if call is in the specified module
  - Disassemble call and check if indirect
  - Save indirect calls as JSON
- IDA Plugin
  - Load JSON
  - Create Cross Reference for indirect calls

ERNW
providing security.

Before Devi

```
; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

var_18= qword ptr -18h

; __unwind {
push    rbp
mov     rbp, rsp
push    rbx
sub     rsp, 18h
mov     edi, 8          ; unsigned __int64
call    __Znwm          ; operator new(ulong)
mov     rbx, rax
mov     rdi, rbx        ; this
call    _ZN10HelloWorldC2Ev ; HelloWorld::HelloWorld(void)
mov     [rbp+var_18], rbx
mov     rax, [rbp+var_18]
mov     rax, [rax]
mov     rax, [rax]
mov     rdx, [rbp+var_18]
mov     rdi, rdx
call    rax
mov     eax, 0
add     rsp, 18h
pop     rbx
pop     rbp
retn
; } // starts at 1169
main endp
```

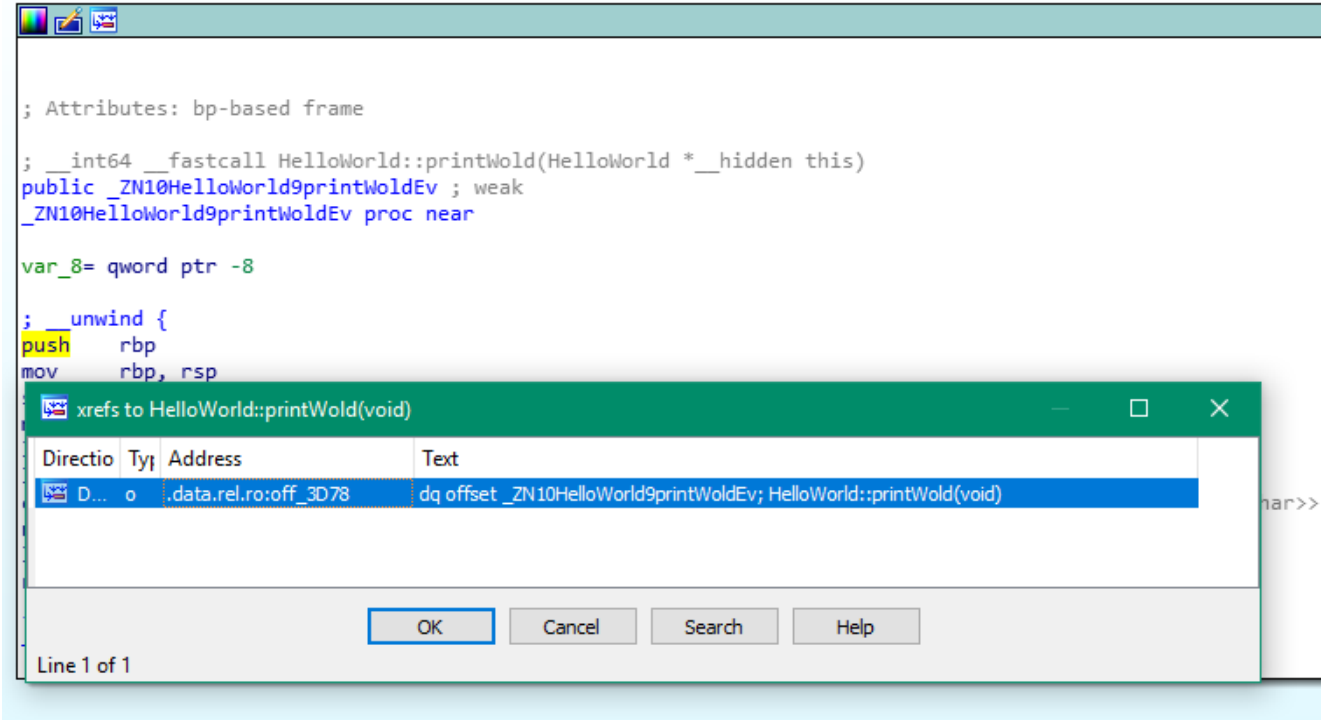
ERNW
providing security.

After Devi

```
; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

var_18= qword ptr -18h

; __unwind {
push    rbp
mov     rbp, rsp
push    rbx
sub     rsp, 18h
mov     edi, 8          ; unsigned __int64
call    __Znwm          ; operator new(ulong)
mov     rbx, rax
mov     rdi, rbx        ; this
call    _ZN10HelloWorldC2Ev ; HelloWorld::HelloWorld(void)
mov     [rbp+var_18], rbx
mov     rax, [rbp+var_18]
mov     rax, [rax]
mov     rax, [rax]
mov     rdx, [rbp+var_18]
mov     rdi, rdx
call    rax             ; HelloWorld::printWold(void)
mov     eax, 0
add     rsp, 18h
pop     rbx
pop     rbp
retn
; } // starts at 1169
main endp
```

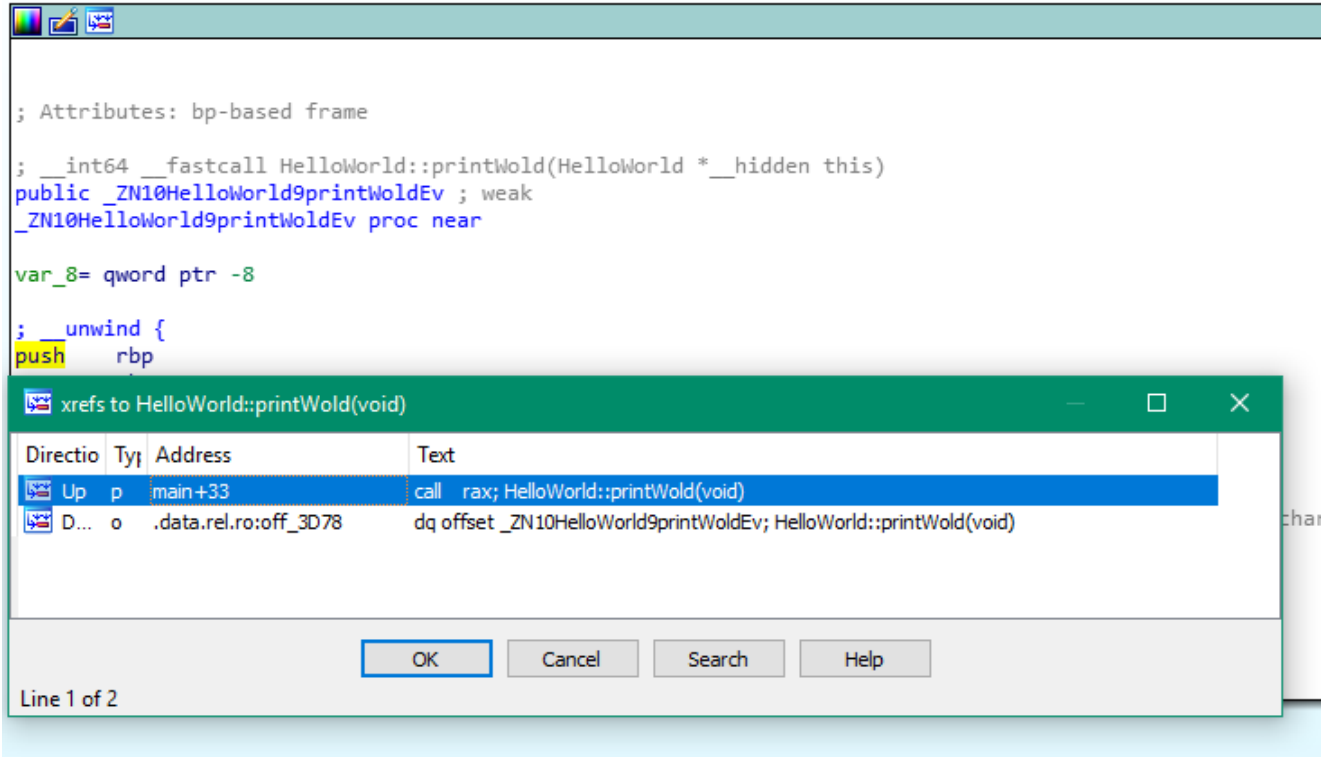
ERNW
providing security.

Before Devi



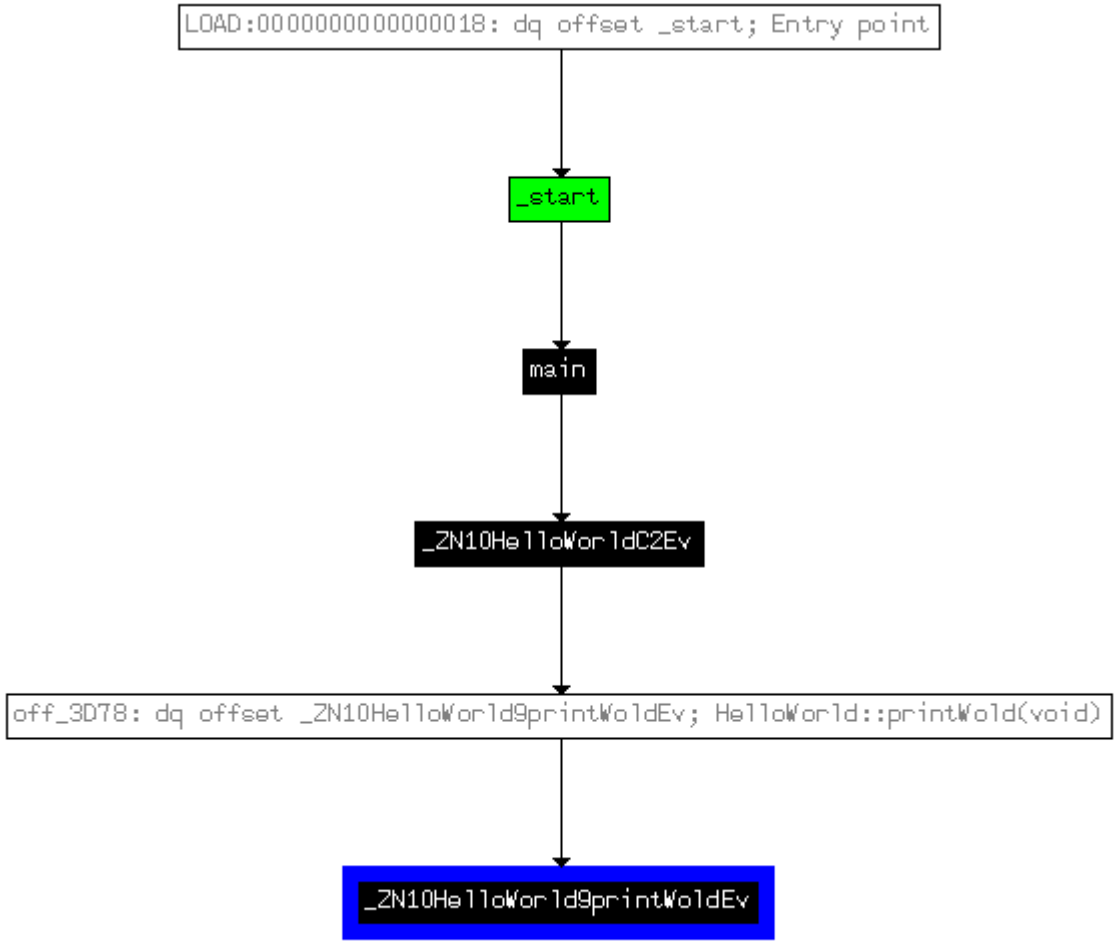
ERNW
providing security.

After Devi



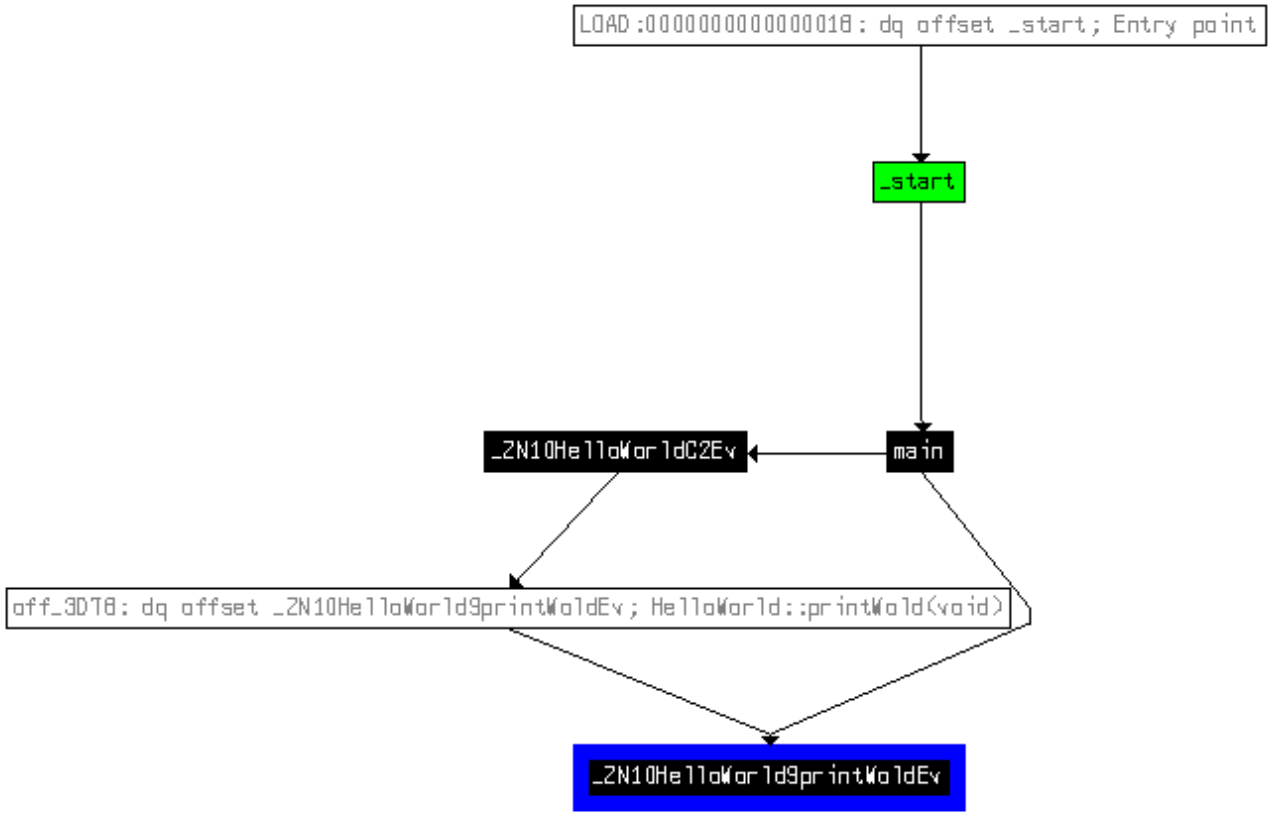
ERNW
providing security.

Before Devi



ERNW
providing security.

After Devi



Plans for the future

- Support more disassemblers
  - Ghidra
  - Binary Ninja
- Support more DBI frameworks
  - Intel PIN
  - DynamoRio
- Write some tests and publish

Thank you for you attention!

Questions?

@_murks

www.ernw.de

www.insinuator.net