REVERSE ENGINEERING MACHINE CODE: PART 2

Advanced Breakpointing

- Software Breakpoints
  - INT 3
- Memory Breakpoints
  - Page guarding
- Hardware Breakpoints
  - CPU hardware

Software Breakpoints

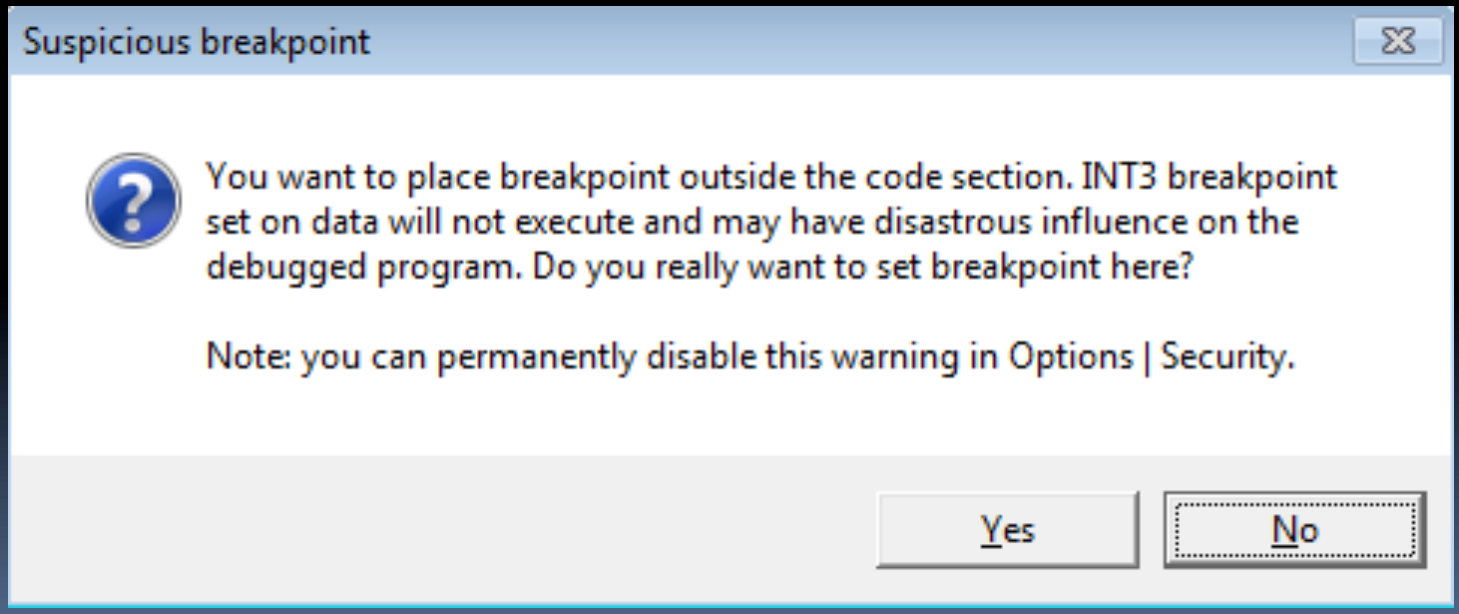
- Software Breakpoints
  - A software breakpoint is an interrupt (0x3)
  - Generates a debug event of `EXCEPTION_DEBUG_EVENT` for the debugger

Software Breakpoints

- Setting a Breakpoint
  - To set a breakpoint at address A_{bp} in process memory:
    - Record the byte value at A_{bp}
    - Set the byte value at A_{bp} to oxCC
    - What is oxCC again???
- Clearing a Breakpoint
  - To clear a breakpoint at address A_{bp} in process memory:
    - Restore the byte value at A_{bp}

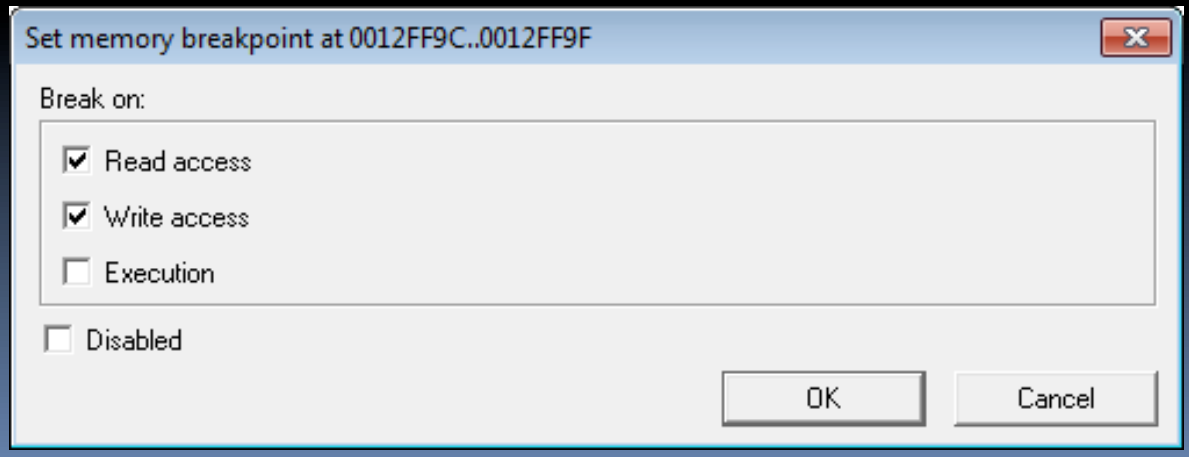
Memory Breakpoints

- Problems
  - Software interrupts change memory values



Memory Breakpoints

- Breaking on Read/Write/Execute
  - We can set a page of memory with `VirtualProtect()` and `PAGE_GUARD`
  - When guarded memory is accessed, a `STATUS_GUARD_PAGE_VIOLATION` exception is generated



Hardware Breakpoints

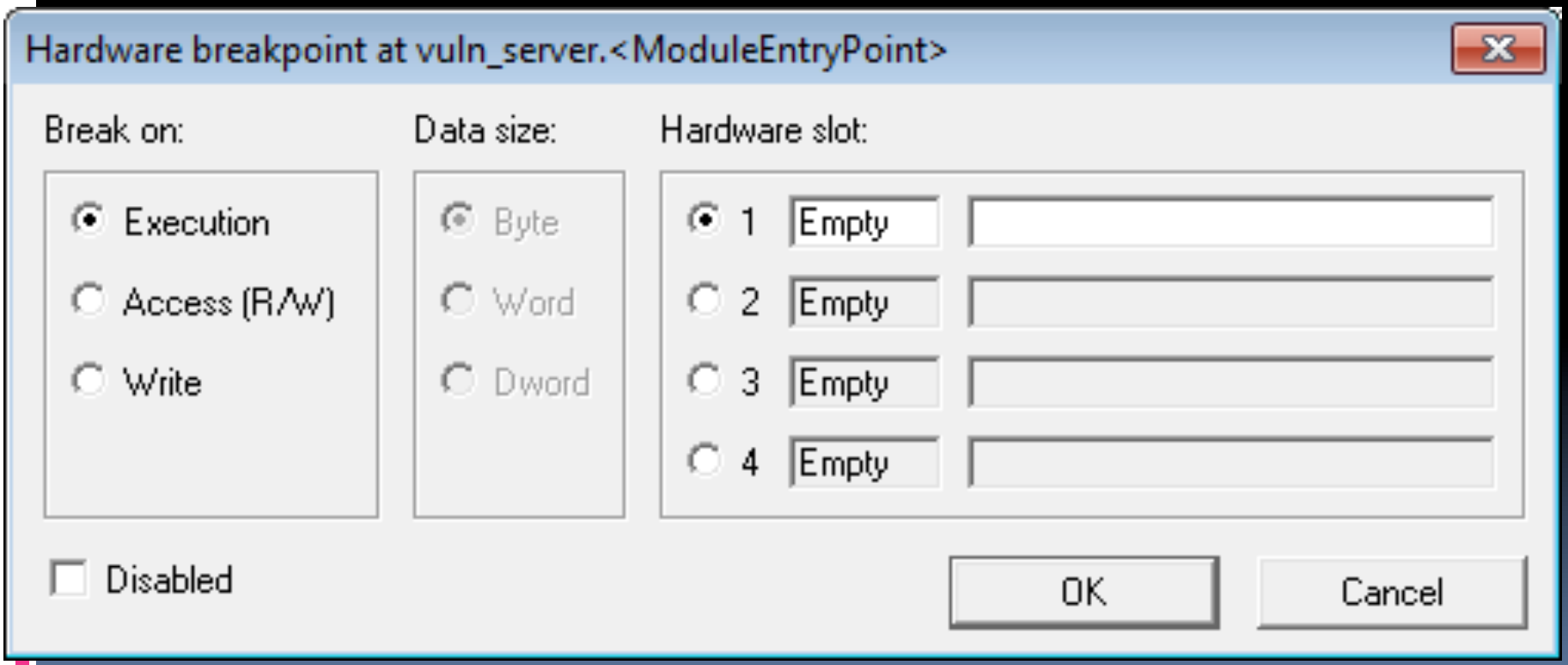
- Processor-handled Breakpointing
  - HW breakpoints are not interrupt instructions
  - Special hardware dedicated to breakpoints
  - Special "debug registers" interact with CPU

Hardware Breakpoints

- x86 Debug Registers
  - DR0-DR3: linear breakpoint addresses
    - Up to 4 hardware breakpoints max
  - DR4-DR5: reserved
  - DR6: debug status
    - 4 bits (0,1,2,3) which are set upon an interrupt
    - Bits must be unset by debug exception handler
  - DR7: debug control
    - Local/global breakpoint enables (0,2,4,6/1,3,5,7)
    - R/W/X breakpoint trigger option (16,17/20,21/24,25/28,29)
    - 1,2,8,4 byte breakpoint trigger area (18,19/22,23/26,27/30,31)

Hardware Breakpoints

- x86 Debug Registers



C++ Reverse Engineering

- Differences from C
  - Indirect calls
    - Via vtables
  - Virtual functions
    - Function that can be overridden by inheriting object
  - The `this` pointer
    - Object oriented programming

C++ Reverse Engineering

```
class Foo : Bar {
public:
    Foo();  // constructor
    ~Foo(): // destructor
    int doFoo();
private:
    int doMoreFoo();
    int x;
    int y;
}
```

C++ vtables

- Virtual Functions
  - Can be overriden in inheriting classes
  - What is printed?

```
class A {
public:
    virtual void func() {cout << "A";}
};
class B: public A {
public:
    void func() {cout << "B";}
};

B *b = new B();
b->func();
```

C++ vtables

- Virtual Method Table (vtable)
  - Array of pointers to the functions of an object
  - vpointer points to the vtable
    - First member of the object for MS compilers
    - After all user-declared members for Unix compilers
- Indirect call

```
mov eax, [edi]            // vtable
mov ecx, edi              // this ptr
call dword ptr[eax+4h]    // vtable method
```

C++ vtables

```
class B1 {
public:
    void f0() {}
    virtual void f1() {}
    int int_in_b1;
};

class B2 {
public:
    virtual void f2() {}
    int int_in_b2;
};
```

C++ vtables

```
B2 *b2 = new B2();
// b2:
//    +0: pointer to vtable of B2
//    +4: value of int_in_b2

// vtable of B2:
//    +0: B2::f2()
```

C++ vtables

```
class D : public B1, public B2 {
public:
    void d() {}
    void f2() {}  // override B2::f2()
    int int_in_d;
};
```

C++ vtables

```
D *d  = new D();
// d:
//   +0: pointer to vtable of D (for B1)
//   +4: value of int_in_b1
//   +8: pointer to vtable of D (for B2)
//  +12: value of int_in_b2
//  +16: value of int_in_d

// vtable of D (for B1):
//   +0: B1::f1()  // B1::f1() is not overridden

// vtable of D (for B2):
//   +0: D::f2()   // B2::f2() is overridden by D::f2()
```

C++ vtables

```
class B1 {
public:
  void f0() {cout << "B1.f0\n";}
  virtual void f1() {cout << "B1.f1\n";}
  int int_in_b1;
};

class B2 {
public:
  virtual void f2() {cout << "B2.f2\n";}
  int int_in_b2;
};

class D : public B1, public B2 {
public:
  void d() {cout << "D.d\n";}
  void f2() {cout << "D.f2\n";}
  int int_in_d;
};
```

```
int main(int argc, char **argv) {
    B1 *b1 = new B1();
    B2 *b2 = new B2();
    D *d   = new D();

    b1->f0();
    b1->f1();
    b1->int_in_b1 = 1;

    b2->f2();
    b2->int_in_b2 = 2;

    d->d();
    d->f2();
    d->int_in_d   = 3;

    return 0;
}
```

```
004011BF | vtables | Always | CALL vtables.operator new
004011DA | vtables | Always | CALL vtables.00401037
00401200 | vtables | Always | CALL vtables.operator new
0040121B | vtables | Always | CALL vtables.0040105A
00401241 | vtables | Always | CALL vtables.operator new
0040125C | vtables | Always | CALL vtables.00401073
00401283 | vtables | Always | CALL vtables.00401023
00401292 | vtables | Always | CALL DWORD PTR DS:[EDX]
0040129E | vtables | Always | MOV DWORD PTR DS:[EAX+4],1
004012AF | vtables | Always | CALL DWORD PTR DS:[EDX]
004012BB | vtables | Always | MOV DWORD PTR DS:[EAX+4],2
004012C5 | vtables | Always | CALL vtables.00401069
004012D8 | vtables | Always | CALL DWORD PTR DS:[EAX]
004012E4 | vtables | Always | MOV DWORD PTR DS:[ECX+10],3
```

C++ vtables

- B1 Constructor
  - Initializes vtable for this object

```
class B1 {
public:
  void f0() {cout << "B1.f0\n";}
  virtual void f1() {cout << "B1.f1\n";}
  int int_in_b1;
};
```

```
00401410  /> 55              PUSH EBP
00401411  |. 8BEC            MOV EBP,ESP
00401413  |. 83EC 44         SUB ESP,44
00401416  |. 53              PUSH EBX
00401417  |. 56              PUSH ESI
00401418  |. 57              PUSH EDI
00401419  |. 51              PUSH ECX
0040141A  |. 8D7D BC         LEA EDI,DWORD PTR SS:[EBP-44]
0040141D  |. B9 11000000     MOV ECX,11
00401422  |. B8 CCCCCCCC     MOV EAX,CCCCCCCC
00401427  |. F3:AB           REP STOS DWORD PTR ES:[EDI]
00401429  |. 59              POP ECX
0040142A  |. 894D FC         MOV DWORD PTR SS:[EBP-4],ECX
0040142D  |. 8B45 FC         MOV EAX,DWORD PTR SS:[EBP-4]
00401430  |. C700 2CF04200   MOV DWORD PTR DS:[EAX],OFFSET vtables.B1::'vftable'
00401436  |. 8B45 FC         MOV EAX,DWORD PTR SS:[EBP-4]
00401439  |. 5F              POP EDI
0040143A  |. 5E              POP ESI
0040143B  |. 5B              POP EBX
0040143C  |. 8BE5            MOV ESP,EBP
0040143E  |. 5D              POP EBP
0040143F  \. C3              RETN
```

C++ vtables

```
class B1 {
public:
  void f0() {cout << "B1.f0\n";}
  virtual void f1() {cout << "B1.f1\n";}
  int int_in_b1;
};

class B2 {
public:
  virtual void f2() {cout << "B2.f2\n";}
  int int_in_b2;
};

class D : public B1, public B2 {
public:
  void d() {cout << "D.d\n";}
  void f2() {cout << "D.f2\n";}
  int int_in_d;
};
```

```
b1->f0();
b1->f1();
b1->int_in_b1 = 1;

b2->f2();
b2->int_in_b2 = 2;

d->d();
d->f2();
d->int_in_d   = 3;
```

```
00401283  . E8 9BFDFFFF   CALL vtables.00401023
00401288  . 8B45 F0       MOV EAX,DWORD PTR SS:[EBP-10]
0040128B  . 8B10          MOV EDX,DWORD PTR DS:[EAX]

0040128F  . 8B4D F0       MOV ECX,DWORD PTR SS:[EBP-10]
00401292  . FF12          CALL DWORD PTR DS:[EDX]

0040129B  . 8B45 F0       MOV EAX,DWORD PTR SS:[EBP-10]
0040129E  . C740 04 01000 MOV DWORD PTR DS:[EAX+4],1
004012A5  . 8B4D EC       MOV ECX,DWORD PTR SS:[EBP-14]
004012A8  . 8B11          MOV EDX,DWORD PTR DS:[ECX]

004012AC  . 8B4D EC       MOV ECX,DWORD PTR SS:[EBP-14]
004012AF  . FF12          CALL DWORD PTR DS:[EDX]

004012B8  . 8B45 EC       MOV EAX,DWORD PTR SS:[EBP-14]
004012BB  . C740 04 02000 MOV DWORD PTR DS:[EAX+4],2
004012C2  . 8B4D E8       MOV ECX,DWORD PTR SS:[EBP-18]
004012C5  . E8 9FFDFFFF   CALL vtables.00401069
004012CA  . 8B4D E8       MOV ECX,DWORD PTR SS:[EBP-18]
004012CD  . 83C1 08       ADD ECX,8
004012D0  . 8B55 E8       MOV EDX,DWORD PTR SS:[EBP-18]
004012D3  . 8B42 08       MOV EAX,DWORD PTR DS:[EDX+8]

004012D8  . FF10          CALL DWORD PTR DS:[EAX]

004012E1  . 8B4D E8       MOV ECX,DWORD PTR SS:[EBP-18]
004012E4  . C741 10 03000 MOV DWORD PTR DS:[ECX+10],3
```

Symbols

- Debug Symbols
  - Attaches names to variables and methods
  - May be compiled into the binary
  - May be distributed in a separate file
  - May be destroyed upon compilation/linking
  - Most debuggers have support for symbols

Name Mangling

- Name Mangling
  - Compiler symbols for
    - Functions
    - Structures
    - Classes
    - …
  - Used to distinguish identifiers of the same name in different namespaces
  - Used by the compiler for function overloading

Name Mangling

- Name Mangling (in C)
  - Fairly standardized
    - Not useful in C since function overloading is not allowed

```
int _cdecl    f (int x) { return 0; }
int _stdcall  g (int y) { return 0; }
int _fastcall h (int z) { return 0; }
```

  - Mangled names
    - _f
    - _g@4
    - @h@4

Name Mangling

- Name Mangling (in C++)
  - Highly used, most non-standardized

```
int  f (void) { return 1; }
int  f (int)  { return 0; }
void g (void) { int i = f(), j = f(0); }
```

  - Mangled names
    - __f_v
    - __f_i
    - __g_v

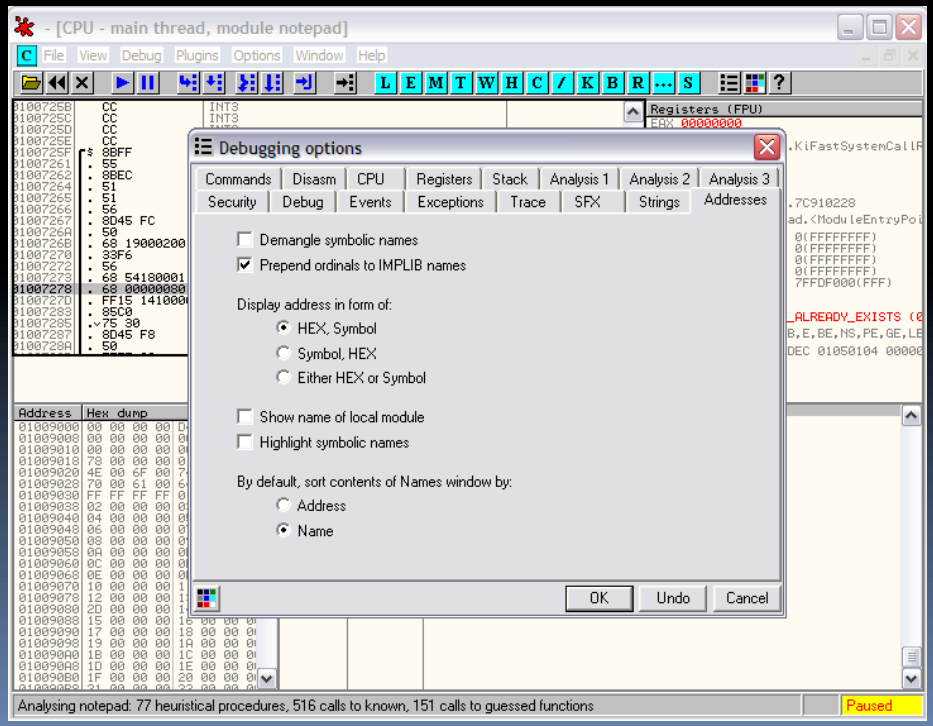
Name Mangling

- Name Mangling (in C++)

Compiler	void h(int)	void h (int, char)	void h(void)
Intel C++ 8.0 for Linux	_Z1hi	_Z1hic	_Z1hv
HP aC++ A.05.55 IA-64	_Z1hi	_Z1hic	_Z1hv
GNU GCC 3.*x* and 4.*x*	_Z1hi	_Z1hic	_Z1hv
HP aC++ A.03.45 PA-RISC	h__Fi	h__Fic	h__Fv
GNU GCC 2.9*x*	h__Fi	h__Fic	h__Fv
Microsoft VC++ v6/v7	?h@@YAXH@Z	?h@@YAXHD@Z	?h@@YAXXZ
Digital Mars C++	?h@@YAXH@Z	?h@@YAXHD@Z	?h@@YAXXZ
Borland C++ v3.1	@h$qi	@h$qizc	@h$qv
OpenVMS C++ V6.5 (ARM mode)	H__XI	H__XIC	H__XV
OpenVMS C++ V6.5 (ANSI mode)	CXX$__7H__FI0ARG51T	CXX$__7H__FIC26CDH77	CXX$__7H__FV2CB06E8
OpenVMS C++ X7.1 IA-64	CXX$_Z1HI2DSQ26A	CXX$_Z1HIC2NP3LI4	CXX$_Z1HV0BCA19V
SunPro CC	__1cBh6Fi_v_	__1cBh6Fic_v_	__1cBh6F_v_
Tru64 C++ V6.5 (ARM mode)	h__Xi	h__Xic	h__Xv
Tru64 C++ V6.5 (ANSI mode)	__7h__Fi	__7h__Fic	__7h__Fv

Name Demangling

- Name Demangling
  - Good disassemblers support name demangling
  - Allows you to guess function types



Windows Internals

- Process/Thread Information
  - PEB
  - TEB
- Exception Handling
  - VEH
  - SEH
  - UEF

Process Environment Block (PEB)

- PEB
  - Pointer to the PEB located at fs:[0x30]
  - Not entirely documented
  - Contains informative information
    - Initialized by kernel or PE header
  - Contains runtime information
    - BeingDebugged
    - Ldr
      - Loaded modules
    - ProcessParameters
      - Information like command line arguments

Thread Environment Block (TEB)

- TEB
  - Also called Thread Information Block (TIB)
  - Located at fs:[0x0]
  - Contains information about a single thread
    - Current SEH frame (fs:[0x0])
    - Linear address of the TIB (fs:[0x18])
    - Process ID (fs:[0x20])
    - Thread ID (fs:[0x24])
    - Pointer to the process PEB (fs:[0x30])

Thread Contexts

- Context
  - The state of a thread's execution
    - Essentially all registers
      - GP registers
      - Memory segment registers
      - EIP
      - …
  - One context per thread

Windows Exception Handling

1. Vectored Exception Handler (VEH)
2. Structured Exception Handler (SEH) Chain
3. Unhandled Exception Filter (UEF)

Vectored Exception Handler

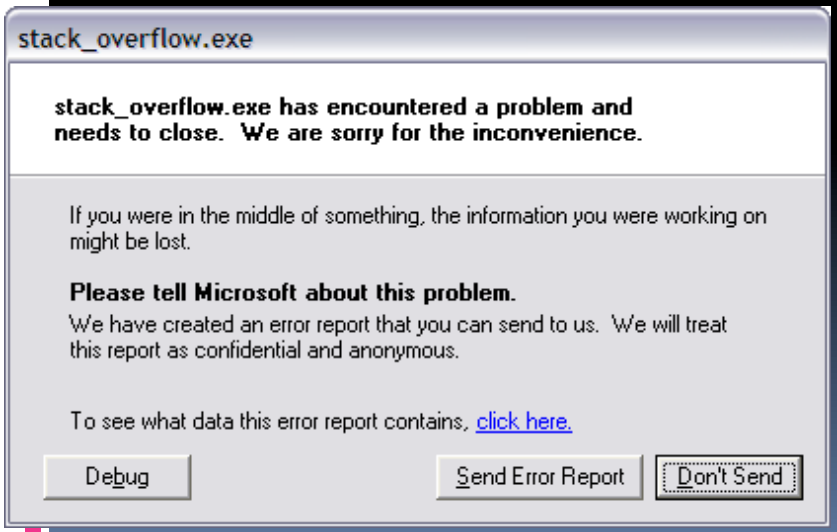
- VEH
  - New feature starting in Windows XP
  - Chain of pointers to exception handlers
  - Chain is located on the heap as a linked list
  - When an exception occurs, the VEH chain is traversed for an appropriate handler
  - VEHs are not frame based
    - Unlike SEHs, VEHs may be triggered from anywhere in the process
  - Vectored exception handling occurs before any frame-based handlers (like SEH)

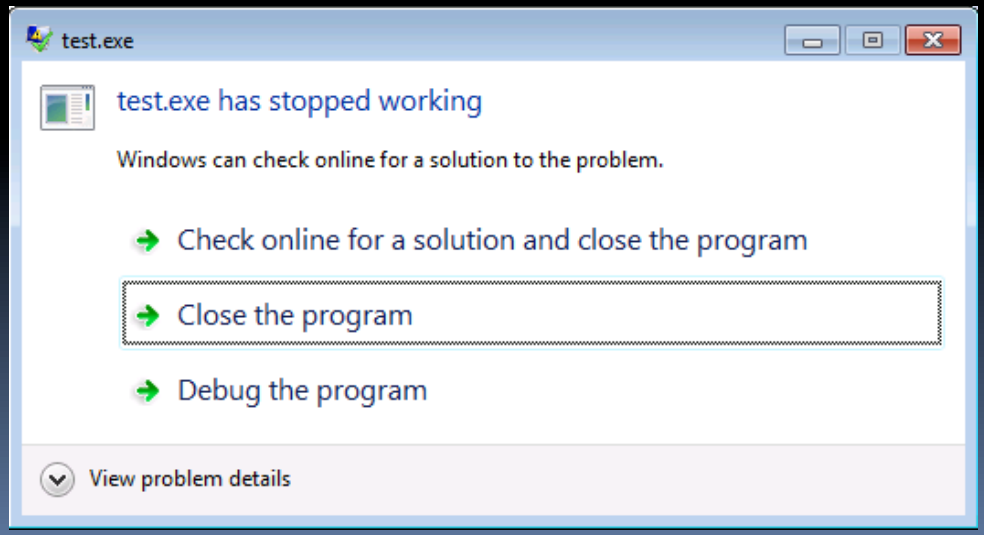
Structured Exception Handler

- SEH Chain
  - Chain of pointers to exception handlers
  - Chain is located on the stack as a linked list
  - `try`/`catch` (or `__try`/`__except`) blocks install these handlers in the chain
  - When an exception occurs, the SEH chain is traversed for an appropriate handler

Unhandled Exception Filter

- UEF
  - "Last ditch effort" exception handler
  - Can be altered externally (through Win32 API)
    - This is how just-in-time-debugging is implemented





Questions/Comments?