ReDbg: Exposition on the Design and Development of a Debugger for Reverse Engineering

Motivation

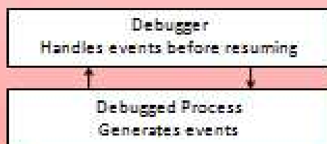
- Malware is increasing! (As much malware was produced in 2007 as in the previous 20 years combined), and this number has only gone up
- Unpacking /anti-debugging are 'black magic'
- Current tools are platform-specific, often closed source, and have steep learning curves

Approach

- Use existing libraries (libdasm)
- Modularity! Debugging events pass through callback chains, to allow discrete handling code
- Make sure actions like anti-debugging cause debugger events so they can be handled
- Provide easy access to a lot of information

Event Flow

- OS APIs are hooked to cause traps in order to control information
- Important areas of memory are marked as unreadable to trap reads and writes



Catching Debugger Checks

```
; check PEB.BeingDebugged directly
mov eax, dword [fs:0x30]
movzx eax, byte [eax+0x02] ;Trap
test eax, eax
jnz debugger_found

; call ntdll!NtQueryInformationProcess()
; to check the debugport in the kernel
call [NtQueryInformationProcess] ;Trap
test eax, eax
jnz debugger_found
```

Evaluation

- Has been used as a debugger to find bugs, even in itself!
- Has been used successfully for examining packed code, handling anti-debugging

Future Direction

- Handle more anti-debugging
- Improved API Identification
- Linux/ptrace integration
- GUI and Scripting support

CS 736 – Jonathan Spalletta , Rohit Koul