Bug Finding Techniques

Tim Newsham and Alex Stamos

Stanford CS155

April 6, 2010

iSEC PARTNERS

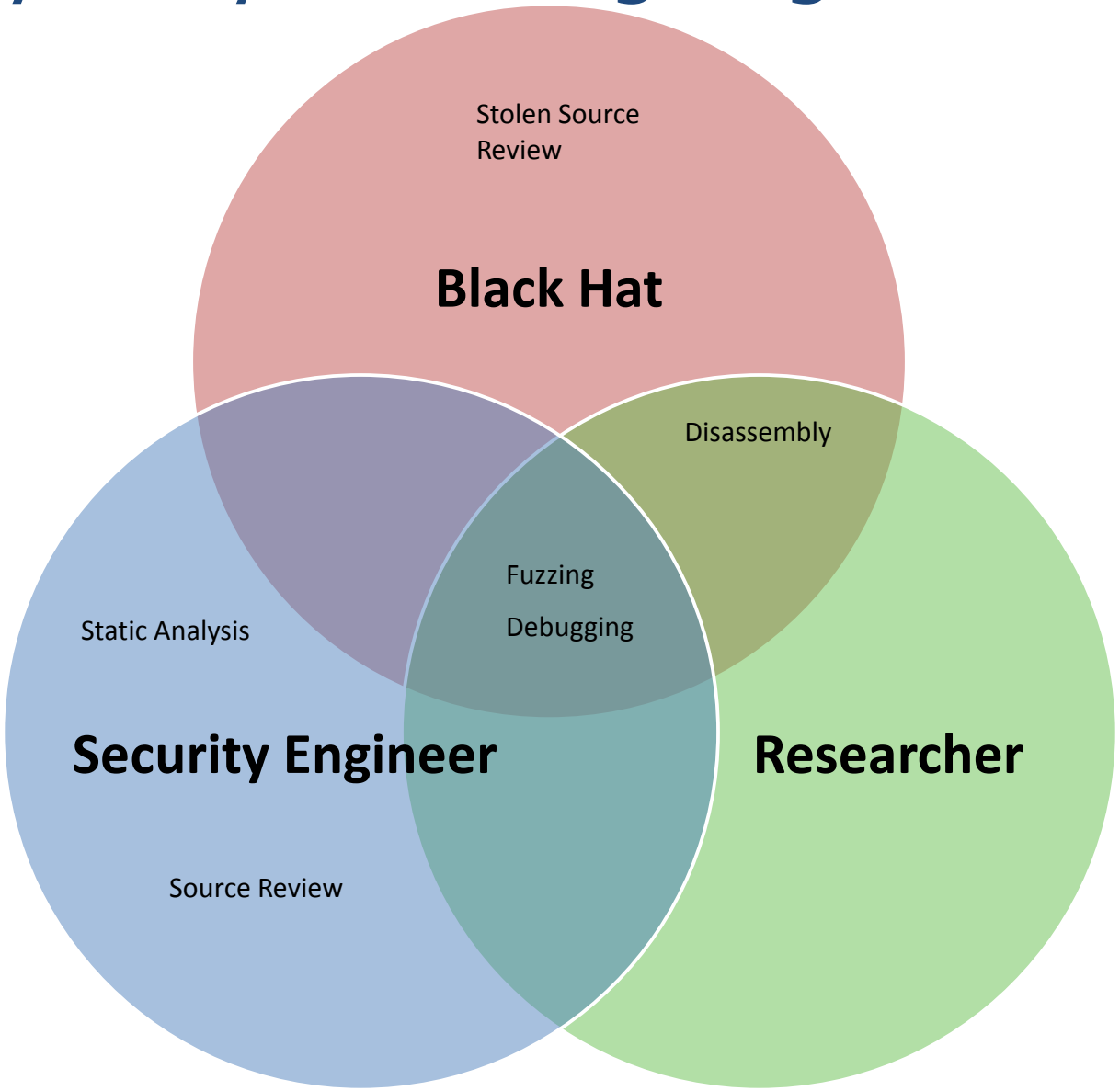
Your Humble Narrators

- Tim Newsham
  - Security Researcher
  - ISS, SNI, NAI, Guardent, @stake, iSEC
  - U of Hawaii BSEE, U of Arizona MSCS
- Alex Stamos
  - Co-Founder and Partner
  - LBNL, Loudcloud, @stake
  - UC Berkeley BS EECS

Agenda

- Why are you finding bugs?
- Overview of common techniques
  - Fuzzing
  - Debugging and Process Stalking
  - Reverse Engineering
- Demo
- Discussion

Why are you finding bugs?



iSEC PARTNERS

Bertha the Black Hat of Ill Repute

- Goal
  - Dependable Exploitation
  - Stealthy
- Thoroughness
  - Usually only need one bug
  - No need to document coverage
- Access
  - Often no source

iSEC PARTNERS

Marvin the Megalomaniacal Researcher

- Goal
  - Column inches from press, props from friends
  - Preferably in a trendy platform
  - Make money from ZDI/Pwn2Own
- Thoroughness
  - Don't need to be perfect, don't want to be embarrassed
- Access
  - Casual access to engineers
  - Source == Lawyers

iSEC PARTNERS

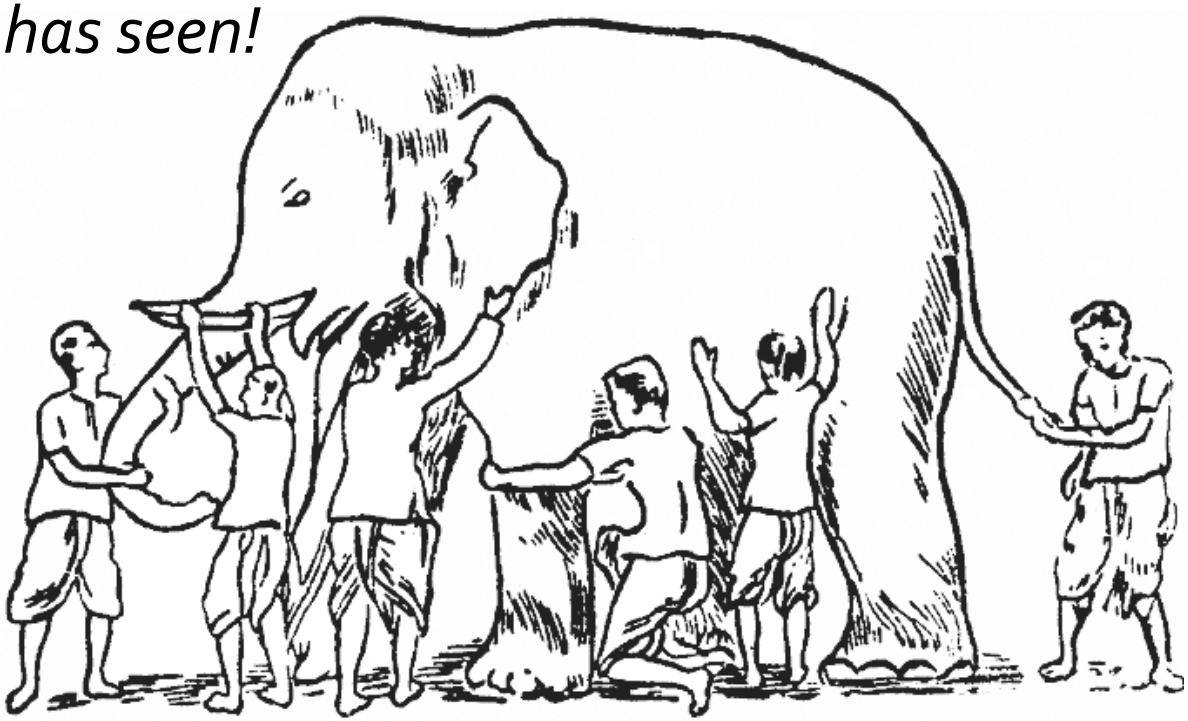
Sally the Stressed Security Engineer

- Goal
  - Find as many flaws as possible
  - Reduce incidence of exploitation*
- Thoroughness
  - Must have coverage metrics
  - Should at least find low-hanging fruit
- Access
  - Source code, debug symbols, engineers
  - Money for tools and staff

iSEC PARTNERS

The Difficulty of Defense

So, oft in theologic wars
The disputants, I ween,
Rail on in utter ignorance
Of what each other mean,
And prate about an Elephant
Not one of them has seen!



iSEC PARTNERS

The Difficulty of Defense

- Asymmetric Warfare
  - Defenders always have to be perfect
  - Attackers can be good and lucky

- Knowing this, is bug finding an efficient defense strategy?

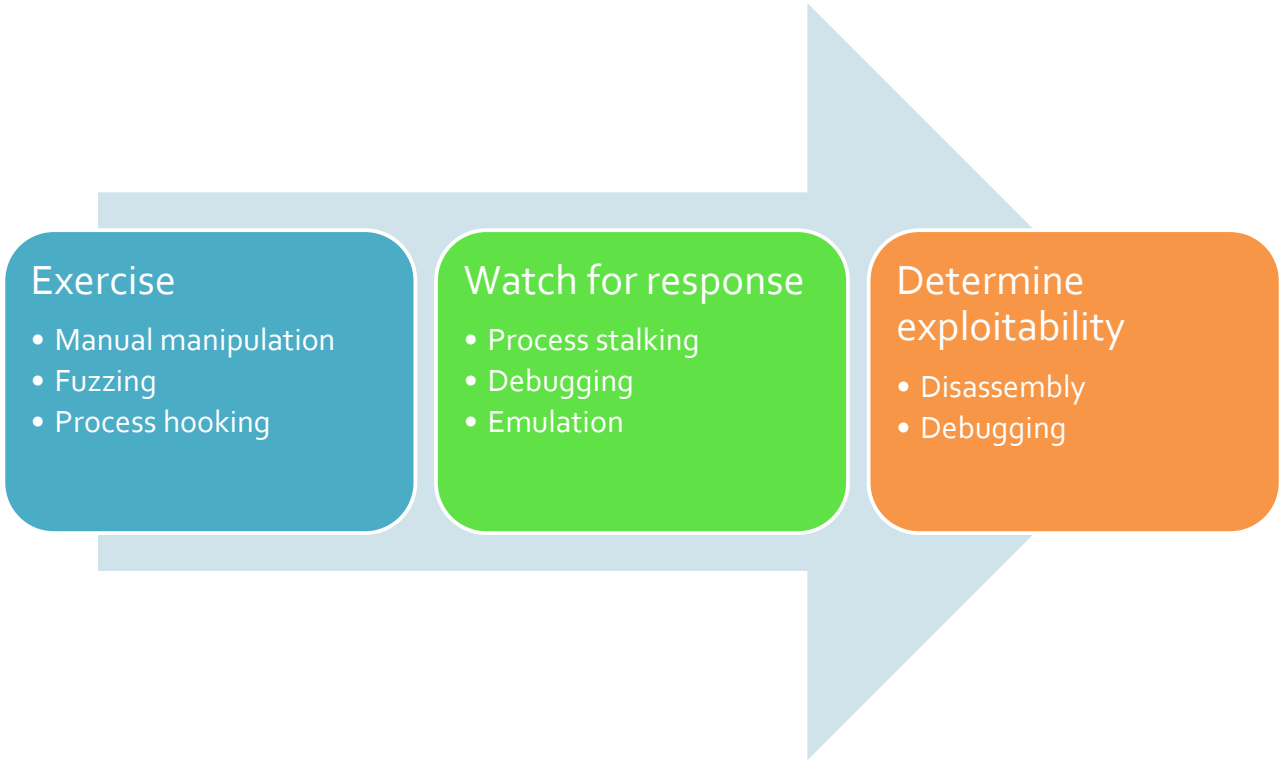
Limitations of Today's Lecture

- The most important flaws we find are NOT implementation flaws

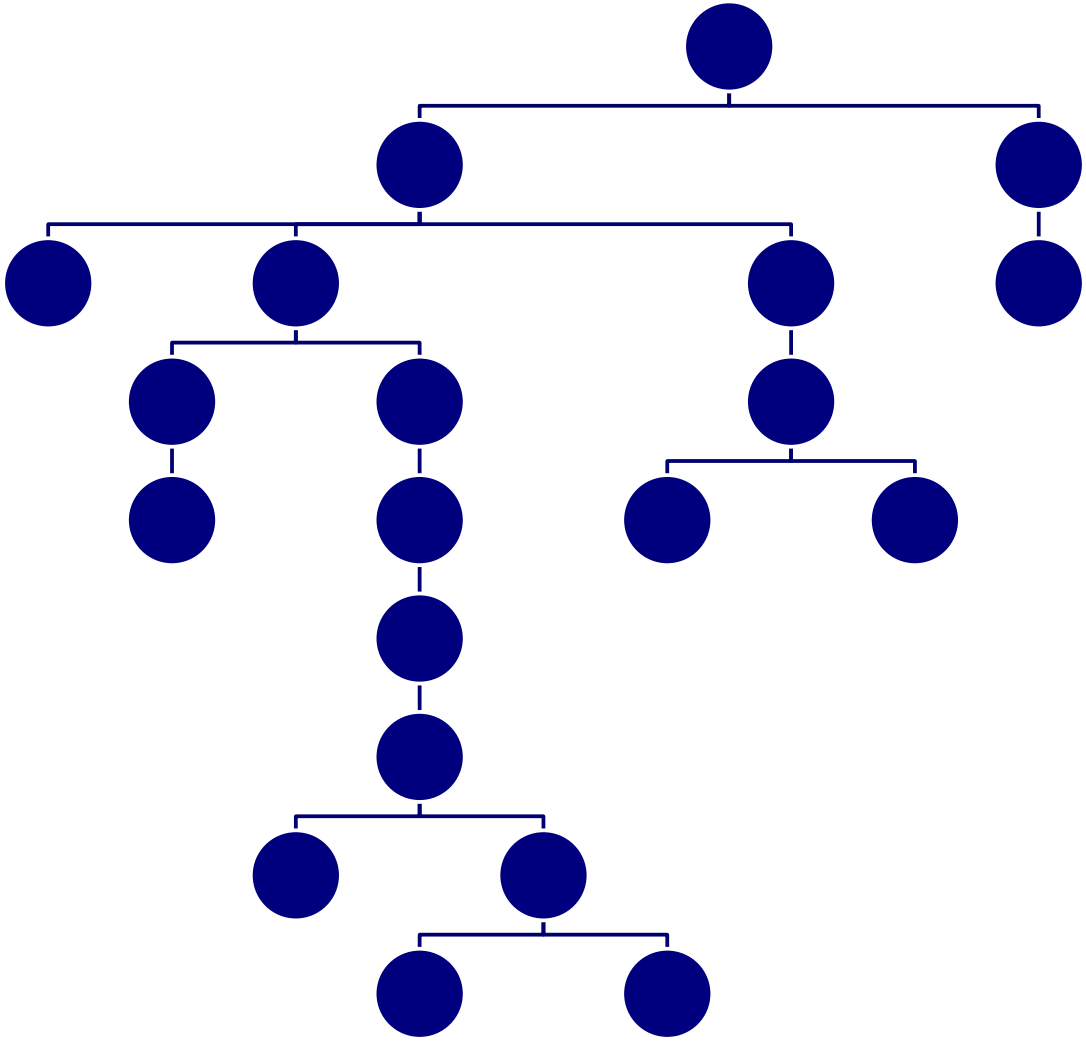
- Common problems:
  - Trusting untrusted components
  - Poor use of cryptography
  - Overreliance on DRM
  - Forgotten or cut security features

Black Box Bug Finding

- Basic goal is to exercise all states of software while watching for a response that indicates vulnerability



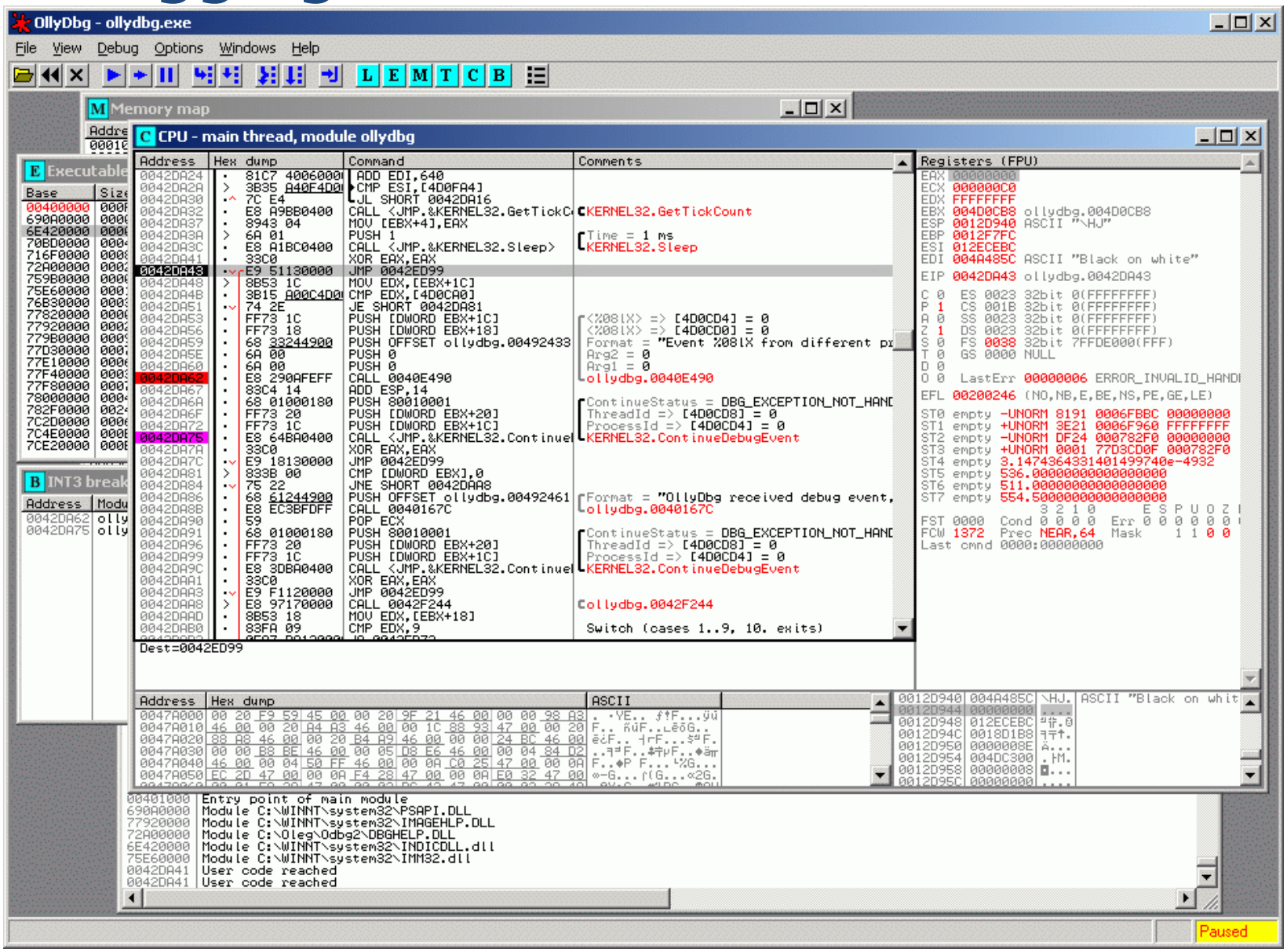
Fuzzing



"Smarter Fuzzing"

- Record or implement path through gating functions
- Utilize knowledge of protocol or file format
- Use process hooking

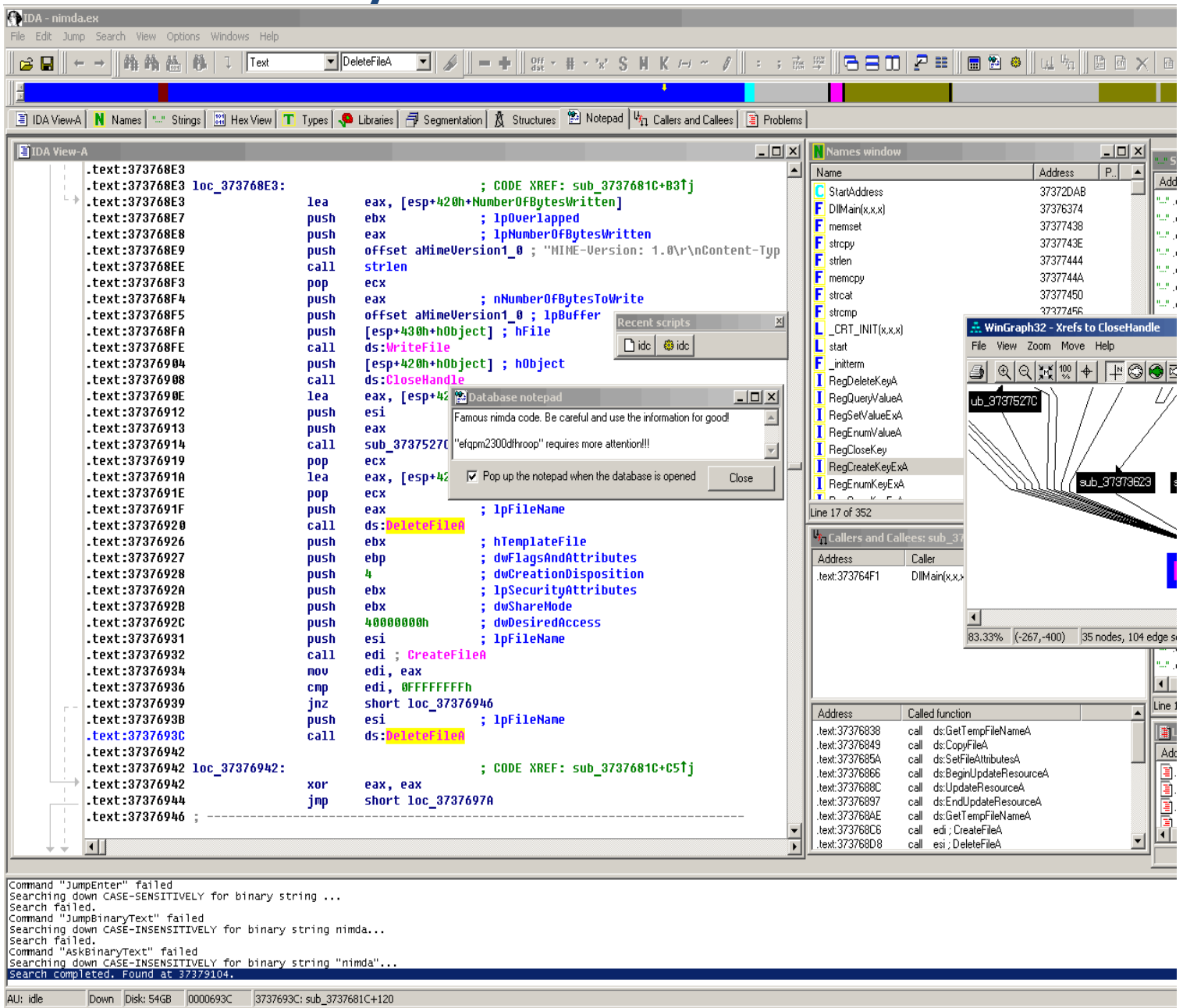
Debugging


OllyDbg - ollydbg.exe
File View Debug Options Windows Help
Memory map
CPU - main thread, module ollydbg
Address Hex dump Command Comments
Registers (FPU)
Executable
INT3 break
Paused

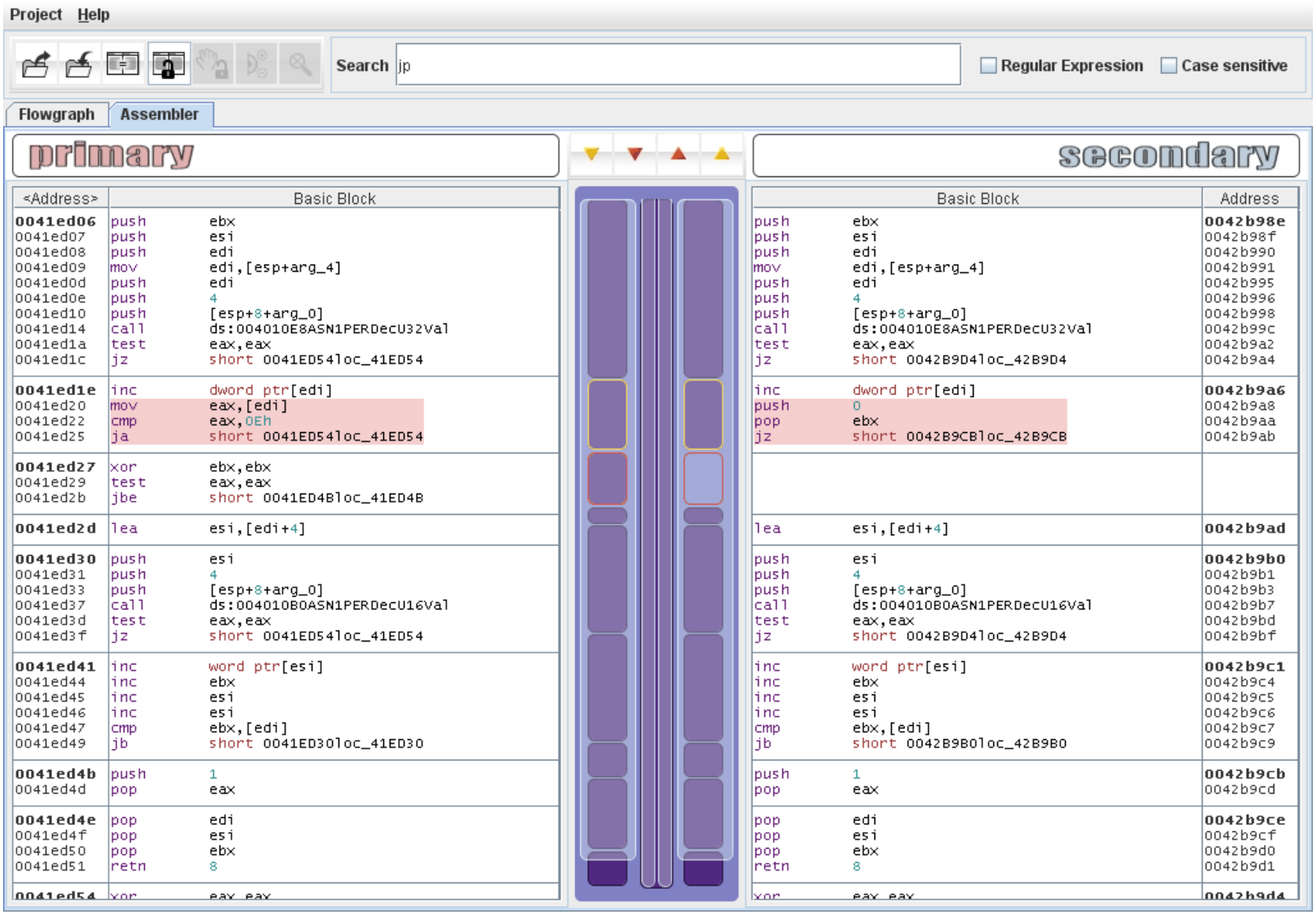
Reverse Engineering

- Decompilation
  - Often used for semi-compiled code
    - .Net CLR
    - Java
    - Flash
  - Can work with C++ w/ symbols
- Disassembly
  - 1:1 matching with machine code
  - Modern disassemblers allow for highly automated analysis process
- Protocol Reverse Engineering

Disassembly - IDA Pro



Reversing Patches - BinDiff

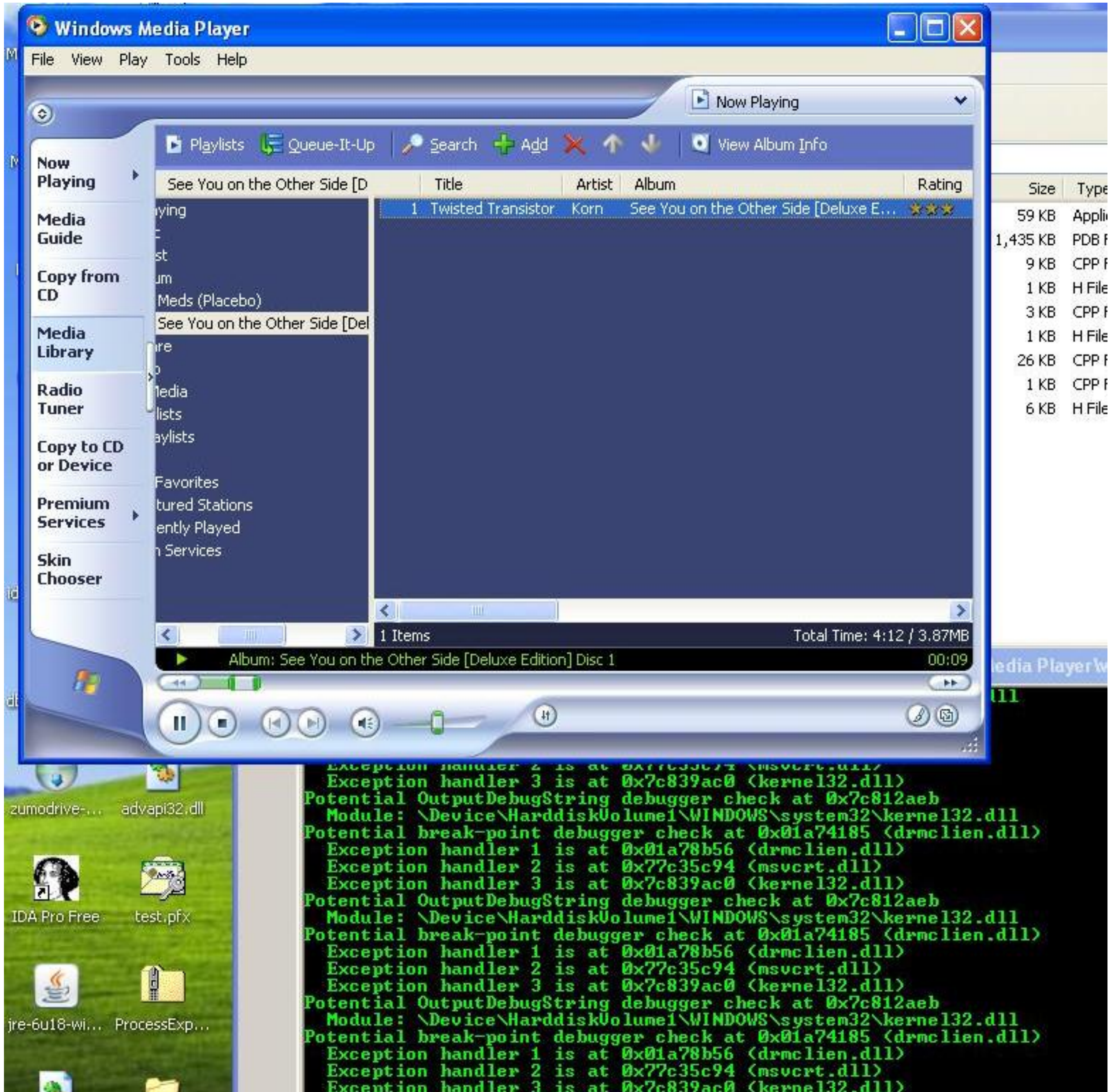


Defeating Black Box Bug Analysis

- Many programs include anti-debug functionality
  - Check PDB
  - System calls, monitor process space
  - Throw INTs, test for catch
  - Timing tests

- Anti-Reversing
  - Dynamic Unpacking
  - Pointer Arithmetic
  - Encrypted and obfuscated function calls

Anti-Anti-Debug - Snitch



iSEC PARTNERS

Snitch Output on WMP

```
Potential break-point debugger check at 0x4bf9f889 (blackbox.dll)
  Exception handler 1 is at 0x4bf9fe71 (blackbox.dll)
  Exception handler 2 is at 0x7c839ac0 (kernel32.dll)
Potential break-point debugger check at 0x4bf9f9fc (blackbox.dll)
  Exception handler 1 is at 0x4bf9fe71 (blackbox.dll)
  Exception handler 2 is at 0x7c839ac0 (kernel32.dll)
Potential break-point debugger check at 0x4bf9f889 (blackbox.dll)
  Exception handler 1 is at 0x4bf9fe71 (blackbox.dll)
  Exception handler 2 is at 0x7c839ac0 (kernel32.dll)
Potential break-point debugger check at 0x4bf9f889 (blackbox.dll)
  Exception handler 1 is at 0x4bf9fe71 (blackbox.dll)
  Exception handler 2 is at 0x7c839ac0 (kernel32.dll)
Potential break-point debugger check at 0x4bf9f889 (blackbox.dll)
  Exception handler 1 is at 0x4bf9fe71 (blackbox.dll)
  Exception handler 2 is at 0x7c839ac0 (kernel32.dll)
Potential OutputDebugString debugger check at 0x7c812aeb
  Module: \Device\HarddiskVolume1\WINDOWS\system32\kernel32.dll
Potential break-point debugger check at 0x4df75f36 (drmv2clt.dll)
  Exception handler 1 is at 0x4dfda68e (drmv2clt.dll)
  Exception handler 2 is at 0x7c839ac0 (kernel32.dll)
```

White Box Bug Finding

- Black Box techniques always work better with more context
  - More quickly triage flaws
  - Patch flaws much faster
- Analysis can start with source code
  - Look at sensitive areas
  - Use lexical analysis to give pointers
    - Flawfinder
    - RATS
  - Use semantic analysis
    - Coverity
    - Fortify
- Most White Box techniques also increase false positive count

Hard to Find Bugs

- MS10-002 – Remote Code Execution in IE 5-8

```
function window :: onload ()
{
    var SourceElement = document.createElement ("div");
    document.body.appendChild (SourceElement);
    var SavedEvent = null;
    SourceElement.onclick = function () {
        SavedEvent = document.createEventObject (event);
        document.body.removeChild (event.srcElement);
    }
    SourceElement.fireEvent ("onclick");
    SourceElement = SavedEvent.srcElement;
}
```

Hard to Find Bugs

- How does this become a reliable exploit?
  - Heap spraying allows for predictable control of memory space
  - IE Small Block Manager Reuses Pages
  - Asynchronous Garbage Collection can be synchronized by attacker: `CollectGarbage()`

- How about on more modern OSes?
  - ASLR and DEP defeated with Flash JIT
  - Return Oriented Programming

  http://cseweb.ucsd.edu/~hovav/talks/blackhat08.html

- Good analyses of Aurora Exploit:

http://www.geoffchappell.com/viewer.htm?doc=notes/security/aurora/index.htm

http://www.hbgary.com/wp-content/themes/blackhat/images/hbgthreatreport_aurora.pdf

Future of Bug Finding

- How could you find this bug?
  - Requires **understanding** of IE code
  - Difficult to triage
- Low-Hanging Fruit is Gone
  - This bug has existed since IE5
- Initial flaw can be found by smart fuzzing. How would you do that?
- Exploitation should require 2-3 flaws for reliability

More Reading

http://www.openrce.org/articles/

Shellcoder's Handbook

http://www.Rootkits.com

http://peachfuzzer.com/

Thank you for coming!

alex@isecpartners.com
newsham@lava.net