

Bug Finding Techniques

Tim Newsham and Alex Stamos

Stanford CS155

April 6, 2010



Your Humble Narrators

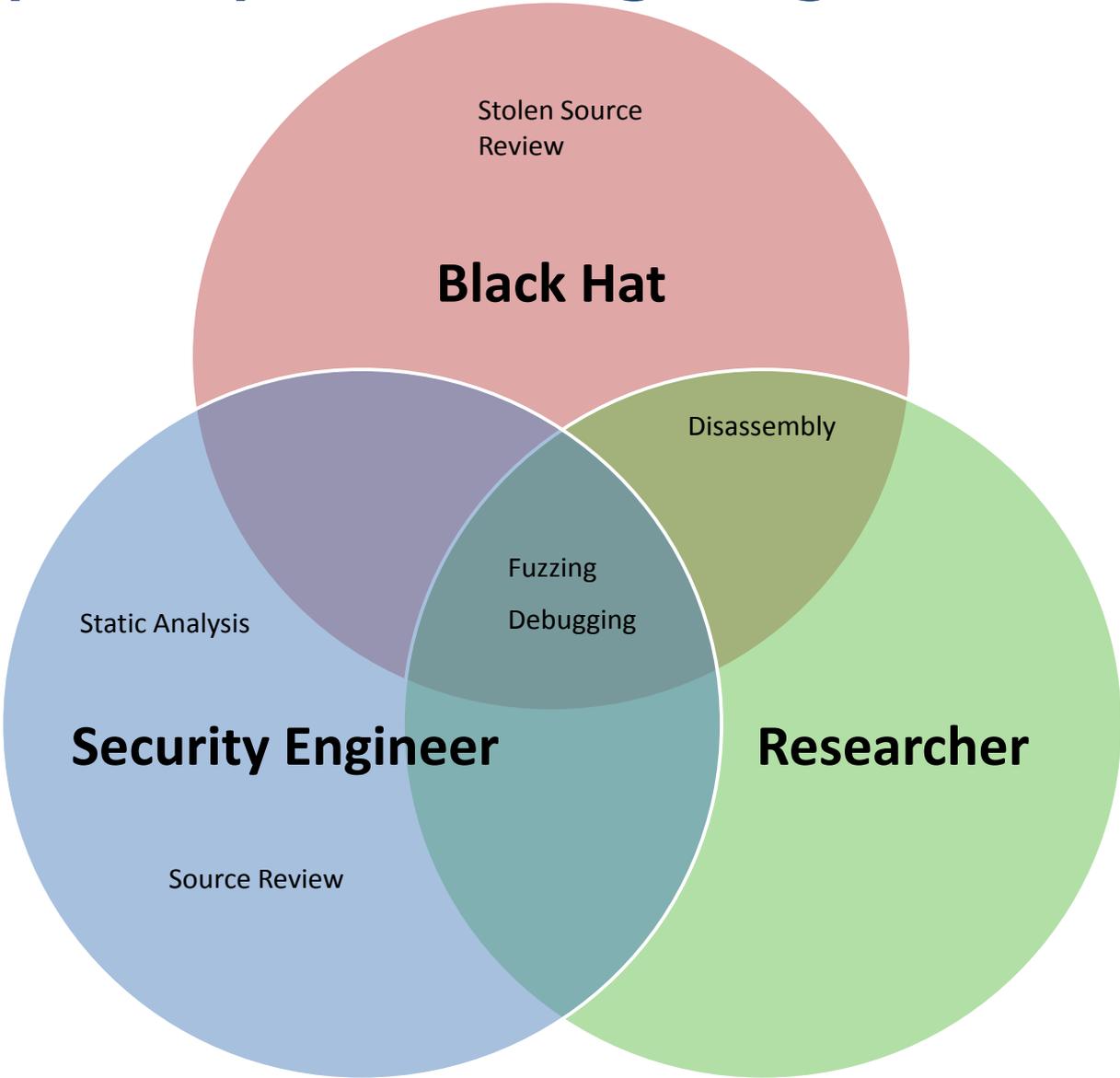
- Tim Newsham
 - Security Researcher
 - ISS, SNI, NAI, Guardent, @stake, iSEC
 - U of Hawaii BSEE, U of Arizona MSCS

- 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?



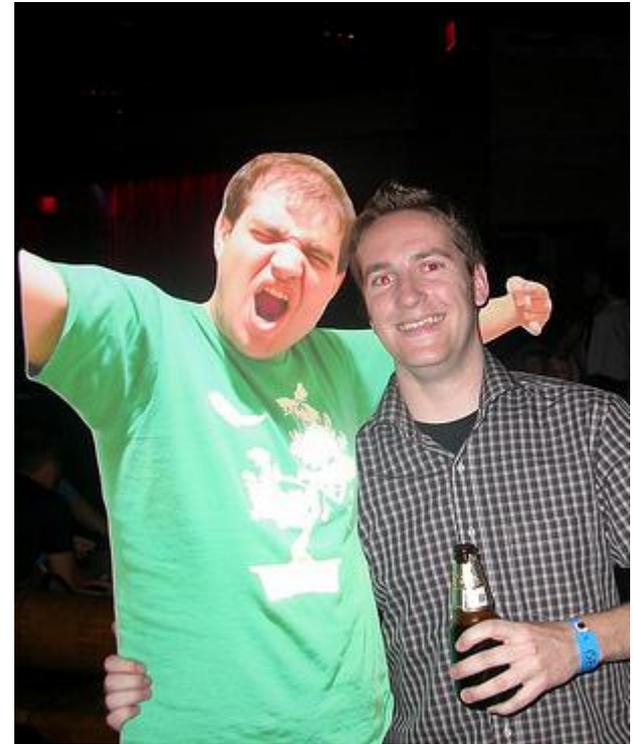
Bertha the Black Hat of Ill Repute



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

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



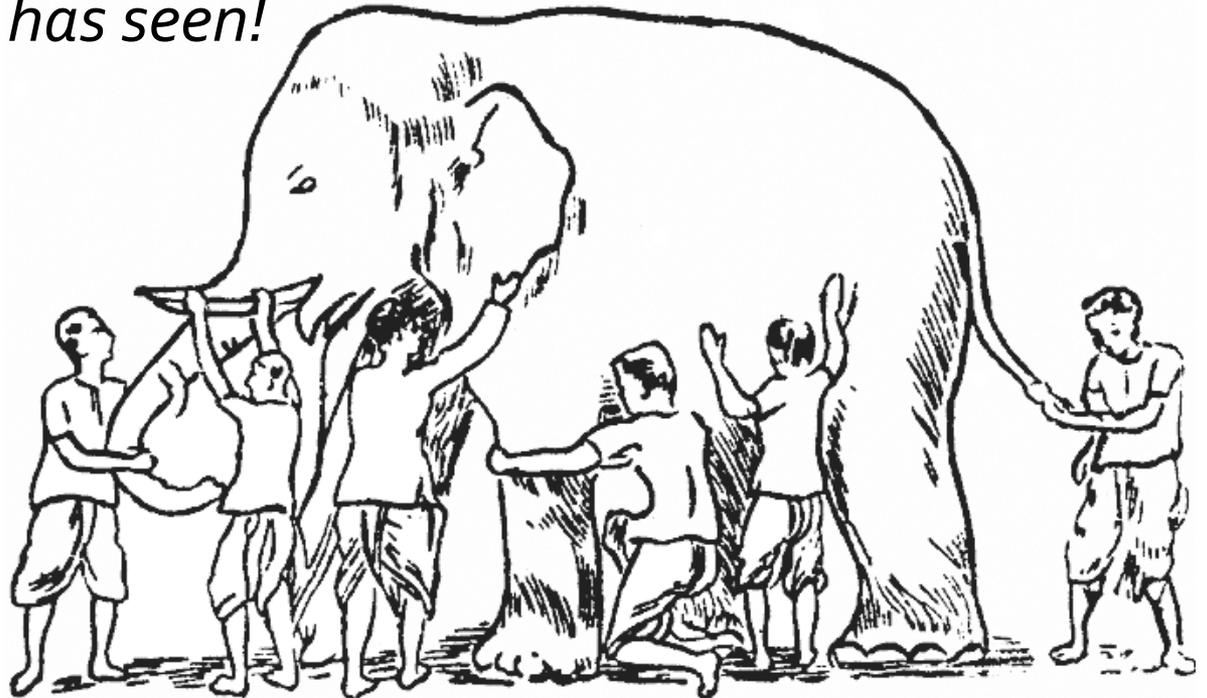
Sally the Stressed Security Engineer



- 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

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!*



The Difficulty of Defense

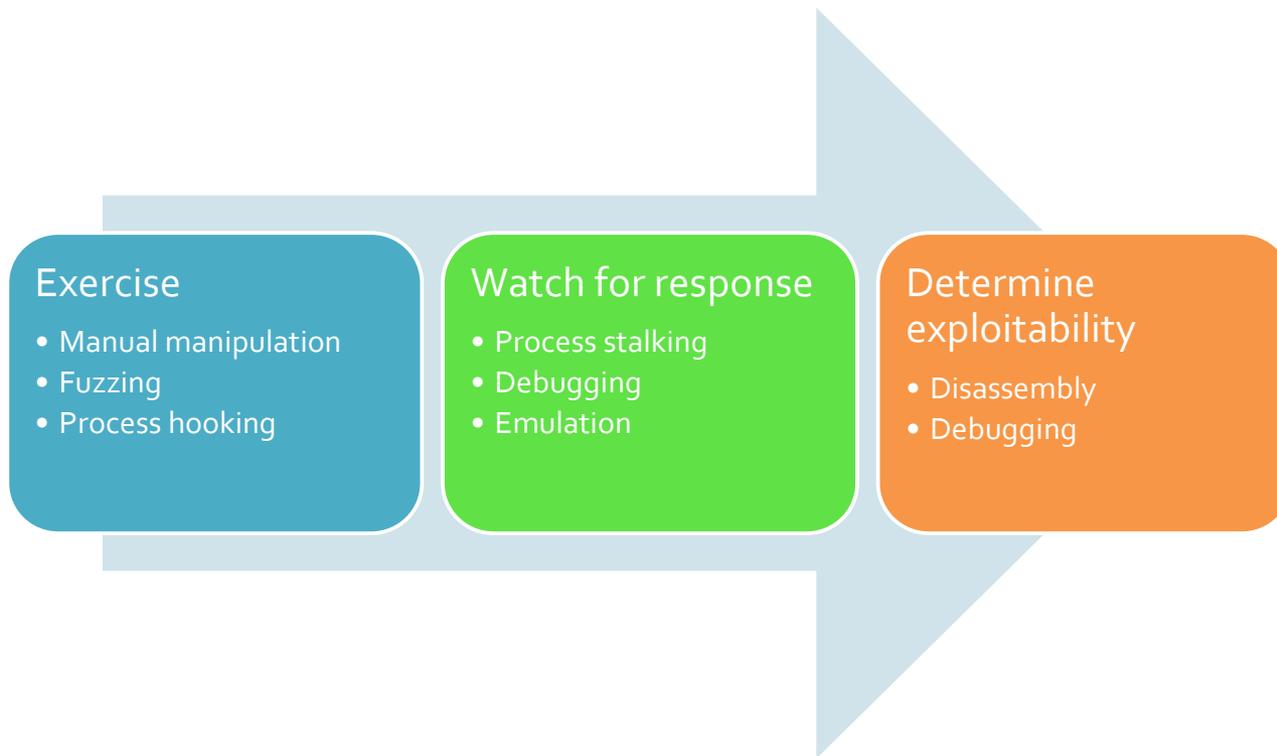
- Asymmetric Warfare
 - Defenders always have to be perfect
 - Attackers can be good and lucky
- Knowing this, is bug finding an efficient defense strategy?

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



“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
0042DA24	81C7 40060000	ADD EDI,640	
0042DA2A	3B35 840F4D00	CMP ESI,[400FA4]	
0042DA30	7C E4	JL SHORT 0042DA16	
0042DA32	E8 A9BB0400	CALL <JMP.&KERNEL32.GetTickCount>	KERNEL32.GetTickCount
0042DA37	8943 04	MOV [EBX+4],EAX	
0042DA3A	6A 01	PUSH 1	
0042DA3C	E8 A1BC0400	CALL <JMP.&KERNEL32.Sleep>	Time = 1 ms KERNEL32.Sleep
0042DA41	33C0	XOR EAX,EAX	
0042DA43	E9 51130000	JMP 0042ED99	
0042DA48	8B53 1C	MOV EDX,[EBX+1C]	
0042DA4E	3B15 000C4D00	CMP EDX,[4D0CA0]	
0042DA51	74 2E	JE SHORT 0042DA81	
0042DA53	FF73 1C	PUSH [DWORD EBX+1C]	<2081X> => [4D0CD4] = 0
0042DA56	FF73 18	PUSH [DWORD EBX+18]	<2081X> => [4D0CD0] = 0
0042DA59	68 33244900	PUSH OFFSET ollydbg.00492433	Format = "Event %08lX from different process"
0042DA5E	6A 00	PUSH 0	Arg2 = 0
0042DA60	6A 00	PUSH 0	Arg1 = 0
0042DA62	E8 290AFEFF	CALL 0040E490	ollydbg.0040E490
0042DA67	83C4 14	ADD ESP,14	
0042DA6A	68 01000180	PUSH 00010001	ContinueStatus = DBG_EXCEPTION_NOT_HANDLED
0042DA6F	FF73 20	PUSH [DWORD EBX+20]	ThreadId => [4D0CD8] = 0
0042DA72	FF73 1C	PUSH [DWORD EBX+1C]	ProcessId => [4D0CD4] = 0
0042DA75	E8 64BA0400	CALL <JMP.&KERNEL32.ContinueDebugEvent>	KERNEL32.ContinueDebugEvent
0042DA7A	33C0	XOR EAX,EAX	
0042DA7C	E9 18130000	JMP 0042ED99	
0042DA81	833B 00	CMP [DWORD EBX],0	
0042DA84	75 22	JNE SHORT 0042DA88	
0042DA86	68 61244900	PUSH OFFSET ollydbg.00492461	Format = "OllyDbg received debug event, ollydbg.0040167C"
0042DA8B	58 EC3BDFDF	CALL 0040167C	
0042DA90	E9	POP ECX	
0042DA91	68 01000180	PUSH 00010001	ContinueStatus = DBG_EXCEPTION_NOT_HANDLED
0042DA96	FF73 20	PUSH [DWORD EBX+20]	ThreadId => [4D0CD8] = 0
0042DA99	FF73 1C	PUSH [DWORD EBX+1C]	ProcessId => [4D0CD4] = 0
0042DA9C	E8 3DBA0400	CALL <JMP.&KERNEL32.ContinueDebugEvent>	KERNEL32.ContinueDebugEvent
0042DA9F	33C0	XOR EAX,EAX	
0042DAA3	E9 F1120000	JMP 0042ED99	
0042DAA8	E8 97170000	CALL 0042F244	ollydbg.0042F244
0042DAAD	8B53 18	MOV EDX,[EBX+18]	
0042DAB0	83FA 09	CMP EDX,9	
0042DAB3	8507 00120000	JS SHORT 0042DAB7	
0042DAB7	8507 00120000	JS SHORT 0042DAB7	
Dest=0042ED99			

Registers (FPU)	
EAX	00000000
ECX	000000C0
EDX	FFFFFFFF
EBX	004D0CB8 ollydbg.004D0CB8
ESP	0012D940 ASCII "\HJ"
EBP	0012F7FC
ESI	012CEBC
EDI	004A485C ASCII "Black on white"
EIP	0042DA43 ollydbg.0042DA43
C 0	ES 0023 32bit 0(FFFFFFFF)
P 1	CS 001B 32bit 0(FFFFFFFF)
A 0	SS 0023 32bit 0(FFFFFFFF)
Z 1	DS 0023 32bit 0(FFFFFFFF)
S 0	FS 0038 32bit 7FDE00(FFF)
T 0	GS 0000 NULL
D 0	
0 0	LastErr 00000006 ERROR_INVALID_HANDLE
EFL	00200246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0	empty -UNORM 8191 0006FBBC 00000000
ST1	empty +UNORM 3E21 0006F960 FFFFFFFF
ST2	empty -UNORM DF24 000782F0 00000000
ST3	empty +UNORM 0001 7D3CD0F 000782F0
ST4	empty 3.1474364331401499740e-4932
ST5	empty 536.0000000000000000
ST6	empty 511.0000000000000000
ST7	empty 554.5000000000000000
FST	0000 Cond 3 2 1 0 Err 0 0 0 0 0
FCW	1372 Prec NEAR,64 Mask 1 1 0 0
Last cmd	0000:00000000

Address	Hex dump	ASCII
0047A000	00 20 F9 59 45 00 00 20 9F 21 46 00 00 00 38 A3	.VE..fF...yu
0047A010	46 00 00 20 04 A3 46 00 00 1C 88 93 47 00 00 20	F..RuF..Le0G..
0047A020	88 88 46 00 00 20 B4 09 46 00 00 24 BC 46 00	eCF..rF...dF.
0047A030	00 00 08 BF 46 00 00 05 D8 E6 46 00 00 04 84 D2	.7?F..*?F...m
0047A040	46 00 00 04 50 FF 46 00 00 0A C0 25 47 00 00 0A	F..*F...%G...
0047A050	EC 2D 47 00 00 0A F4 28 47 00 00 0A E0 32 47 00	w-G...r(G...x2G.
0047A060	00 01 F3 20 47 00 00 00 00 00 00 00 00 00 00	...G...x2G...
00481000	Entry point of main module	
69000000	Module C:\WINNT\system32\PSAPI.DLL	
72920000	Module C:\WINNT\system32\IMAGEHLP.DLL	
72A00000	Module C:\ole32\ole32\DBGHELP.DLL	
6E420000	Module C:\WINNT\system32\INDICDLL.dll	
75E60000	Module C:\WINNT\system32\IMM32.dll	
0042DA41	User code reached	
0042DA41	User code reached	

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

The screenshot displays the IDA Pro interface with the following components:

- Assembly View:** Shows assembly code for a function. Key instructions include `push offset aMineVersion1_0`, `call ds:WriteFile`, `call ds:CloseHandle`, `call ds:DeleteFileA`, and `call ds:CreateFileA`. Cross-references are shown as `CODE XREF: sub_3737681C+B3fj` and `CODE XREF: sub_3737681C+C5fj`.
- Names Window:** Lists symbols such as `StartAddress`, `DllMain(x,x,x)`, `memset`, `strcpy`, `strlen`, `memcpy`, `strcat`, `strcmp`, `L_CRT_INIT(x,x,x)`, `start`, `_initem`, `RegDeleteKeyA`, `RegQueryValueA`, `RegSetValueExA`, `RegEnumValueA`, `RegCloseKey`, `RegCreateKeyExA`, and `RegEnumKeyExA`.
- Callers and Callees:** Shows the caller `DllMain(x,x,x)` at address `text:373764F1`.
- WinGraph32 - Xrefs to CloseHandle:** A graph showing cross-references to `sub_3737627C` and `sub_3737623`.
- Database notepad:** A small dialog box with the text: "Famous nimda code. Be careful and use the information for good!" and "efqpm2300dlhroop" requires more attention!!!. It has a checked option "Pop up the notepad when the database is opened" and a "Close" button.

```
Command "JumpEnter" failed
Searching down CASE-SENSITIVELY for binary string ...
Search failed.
Command "JumpBinaryText" failed
Searching down CASE-INSENSITIVELY for binary string nimda...
Search failed.
Command "AskBinaryText" failed
Searching down CASE-INSENSITIVELY for binary string "nimda"...
Search completed. Found at 37379104.
```

Reversing Patches - BinDiff

Project Help


 Search
 Regular Expression Case sensitive

Flowgraph Assembler

primary

<Address>	Basic Block	
0041ed06	push	ebx
0041ed07	push	esi
0041ed08	push	edi
0041ed09	mov	edi, [esp+arg_4]
0041ed0d	push	edi
0041ed0e	push	4
0041ed10	push	[esp+8+arg_0]
0041ed14	call	ds:004010E8ASN1PERDecu32va1
0041ed1a	test	eax, eax
0041ed1c	jz	short 0041ED54loc_41ED54
0041ed1e	inc	dword ptr[edi]
0041ed20	mov	eax, [edi]
0041ed22	cmp	eax, 0EH
0041ed25	ja	short 0041ED54loc_41ED54
0041ed27	xor	ebx, ebx
0041ed29	test	eax, eax
0041ed2b	jbe	short 0041ED4Bloc_41ED4B
0041ed2d	lea	esi, [edi+4]
0041ed30	push	esi
0041ed31	push	4
0041ed33	push	[esp+8+arg_0]
0041ed37	call	ds:004010B0ASN1PERDecu16va1
0041ed3d	test	eax, eax
0041ed3f	jz	short 0041ED54loc_41ED54
0041ed41	inc	word ptr[esi]
0041ed44	inc	ebx
0041ed45	inc	esi
0041ed46	inc	esi
0041ed47	cmp	ebx, [edi]
0041ed49	jb	short 0041ED30loc_41ED30
0041ed4b	push	1
0041ed4d	pop	eax
0041ed4e	pop	edi
0041ed4f	pop	esi
0041ed50	pop	ebx
0041ed51	retn	8
0041ed54	xor	eax, eax



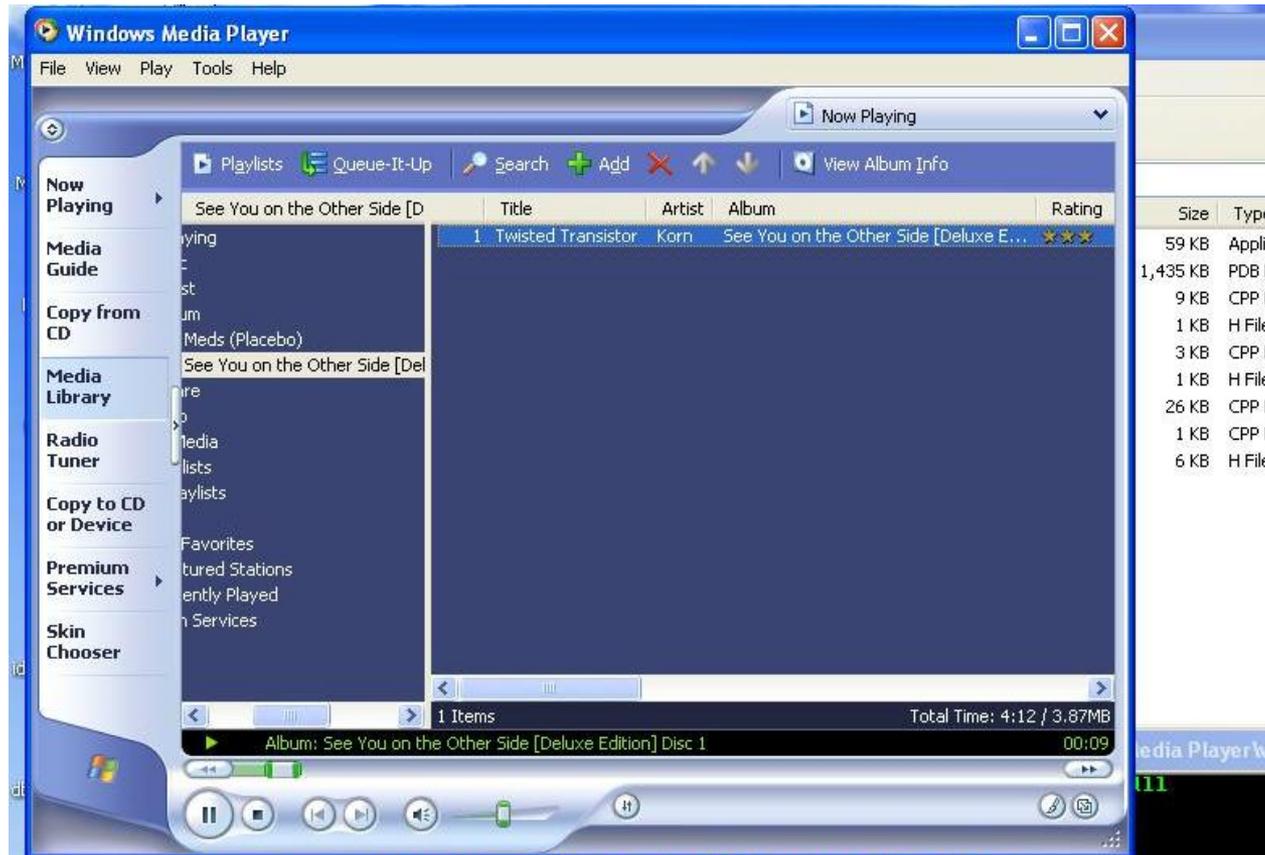
secondary

Basic Block		Address
push	ebx	0042b98e
push	esi	0042b98f
push	edi	0042b990
mov	edi, [esp+arg_4]	0042b991
push	edi	0042b995
push	4	0042b996
push	[esp+8+arg_0]	0042b998
call	ds:004010E8ASN1PERDecu32va1	0042b99c
test	eax, eax	0042b9a2
jz	short 0042B9D4loc_42B9D4	0042b9a4
inc	dword ptr[edi]	0042b9a6
push	0	0042b9a8
pop	ebx	0042b9aa
jz	short 0042B9CBloc_42B9CB	0042b9ab
lea	esi, [edi+4]	0042b9ad
push	esi	0042b9b0
push	4	0042b9b1
push	[esp+8+arg_0]	0042b9b3
call	ds:004010B0ASN1PERDecu16va1	0042b9b7
test	eax, eax	0042b9bd
jz	short 0042B9D4loc_42B9D4	0042b9bf
inc	word ptr[esi]	0042b9c1
inc	ebx	0042b9c4
inc	esi	0042b9c5
inc	esi	0042b9c6
cmp	ebx, [edi]	0042b9c7
jb	short 0042B9B0loc_42B9B0	0042b9c9
push	1	0042b9cb
pop	eax	0042b9cd
pop	edi	0042b9ce
pop	esi	0042b9cf
pop	ebx	0042b9d0
retn	8	0042b9d1
xor	eax, eax	0042b9d4

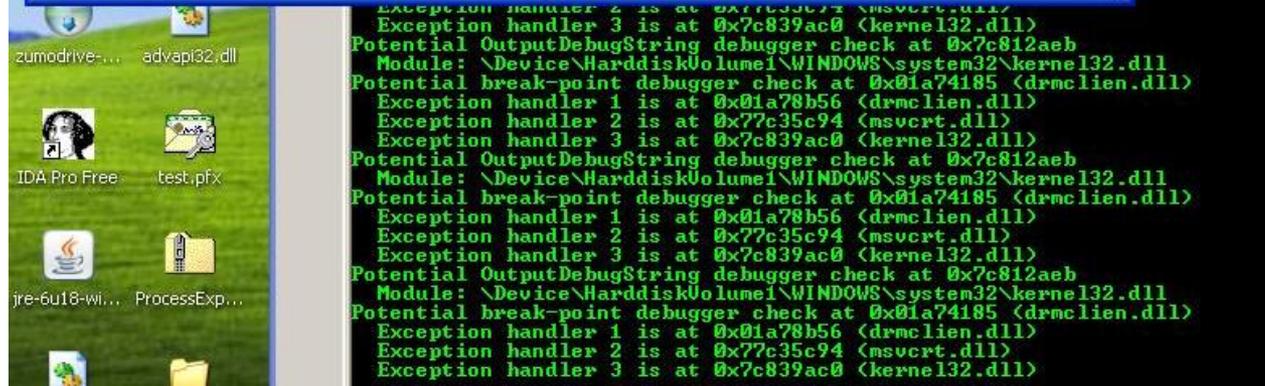
Defeating Black Box Bug Analysis

- 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



Size	Type
59 KB	Appli
1,435 KB	PDB F
9 KB	CPP F
1 KB	H File
3 KB	CPP F
1 KB	H File
26 KB	CPP F
1 KB	CPP F
6 KB	H File



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/blackhato8.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