Where do security bugs come from?

Any how do you find them in the real world?

Stanford CS 155
Spring 2014

Alex Stamos

- **CISO, Yahoo**

Your humble narrator

- CISO at Yahoo!
- 2012 - founded Artemis Internet (.secure and .trust)
- 2004 – co-founded iSEC Partners
- Berkeley BS EECS '01

Agenda

- Who is looking for security bugs?
- Bug Finding Techniques
- Sample of real bugs found in the wild
- Bug Finding as Warfare
- I'm in love with security; whatever shall I do?

Who looks for security bugs?

- Engineers
- Criminals
- Security Researchers
- Pen Testers
- Governments
- Hacktivists
- Academics

Engineers (create and find bugs)

- Goals:
  - Find as many flaws as possible
  - Reduce incidence of exploitation
- Thoroughness:
  - Need coverage metrics
  - At least find low-hanging fruit
- Access:
  - Source code, debug environments, engineers
  - Money for tools and staff

Engineering challenges

- People care about features, not security (until something goes wrong)
- Engineers typically only see a small piece of the puzzle
- "OMG PDF WTF" (Julia Wolf, 2010)
  - How many lines of code in Linux 2.6.32?
  - How many lines in Windows NT 4?
  - How many in Adobe Acrobat?

Engineering challenges

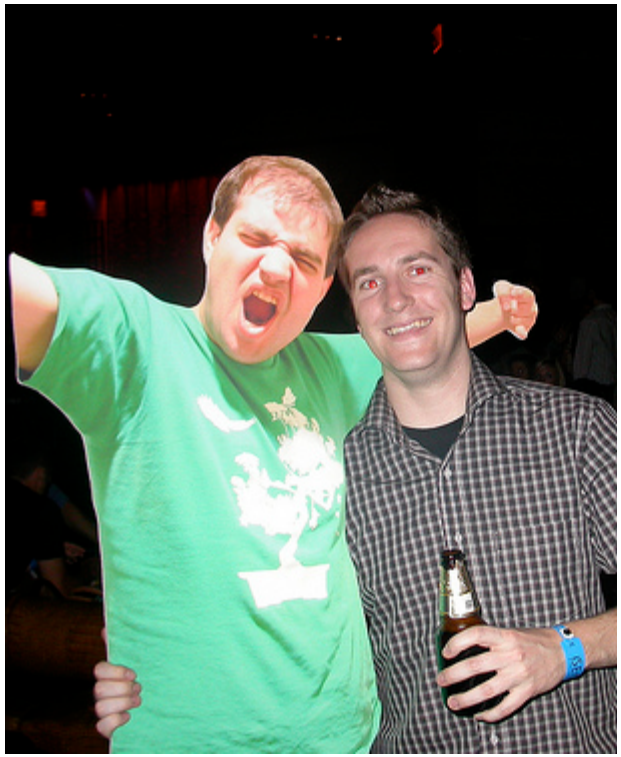
- People care about features, not security (until something goes wrong)
- Engineers typically only see a small piece of the puzzle
- "OMG PDF WTF" (Julia Wolf, 2010)
  - How many lines of code in Linux 2.6.32?
    - 8 – 12.6 million
  - How many lines in Windows NT 4?
    - 11-12 million
  - How many in Adobe Acrobat?
    - 15 million

Criminals

- Goals:
  - Money (botnets, CC#s, blackmail)
  - Stay out of jail
- Thoroughness:
  - Reliable exploits
  - Don't need 0-days (but they sure are nice)
- Access:
  - Money
  - Blackbox testing

Security Researchers

- Goals:
  - Column inches from press, props from friends
  - Preferably in a trendy platform
- Thoroughness:
  - Don't need to be perfect, don't want to be embarrassed
- Access:
  - Casual access to engineers
  - Source == Lawyers

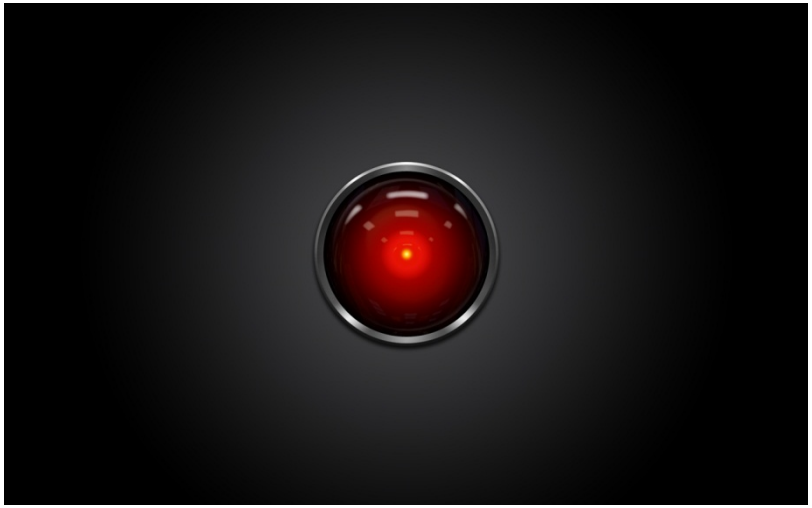


Pen Testers

- Goals:
  - Making clients and users safer
  - Finding vulns criminals would use
- Thoroughness:
  - Need coverage
  - Find low-hanging fruit
  - Find high impact vulnerabilities
  - Don't fix or fully exploit
- Access:
  - Access to Engineers
  - Access to Source
  - Permission

Governments

- Goals:
  - Attack/espionage
  - Defend
- Thoroughness:
  - Reliable exploits
- Access:
  - Money
  - Talent
  - Time



Hacktivists

- Goals:
  - Doing something "good"
  - Stay out of jail
- Thoroughness:
  - Reliable exploits
  - Don't need 0-days
- Access:
  - Talent
  - Plentiful targets

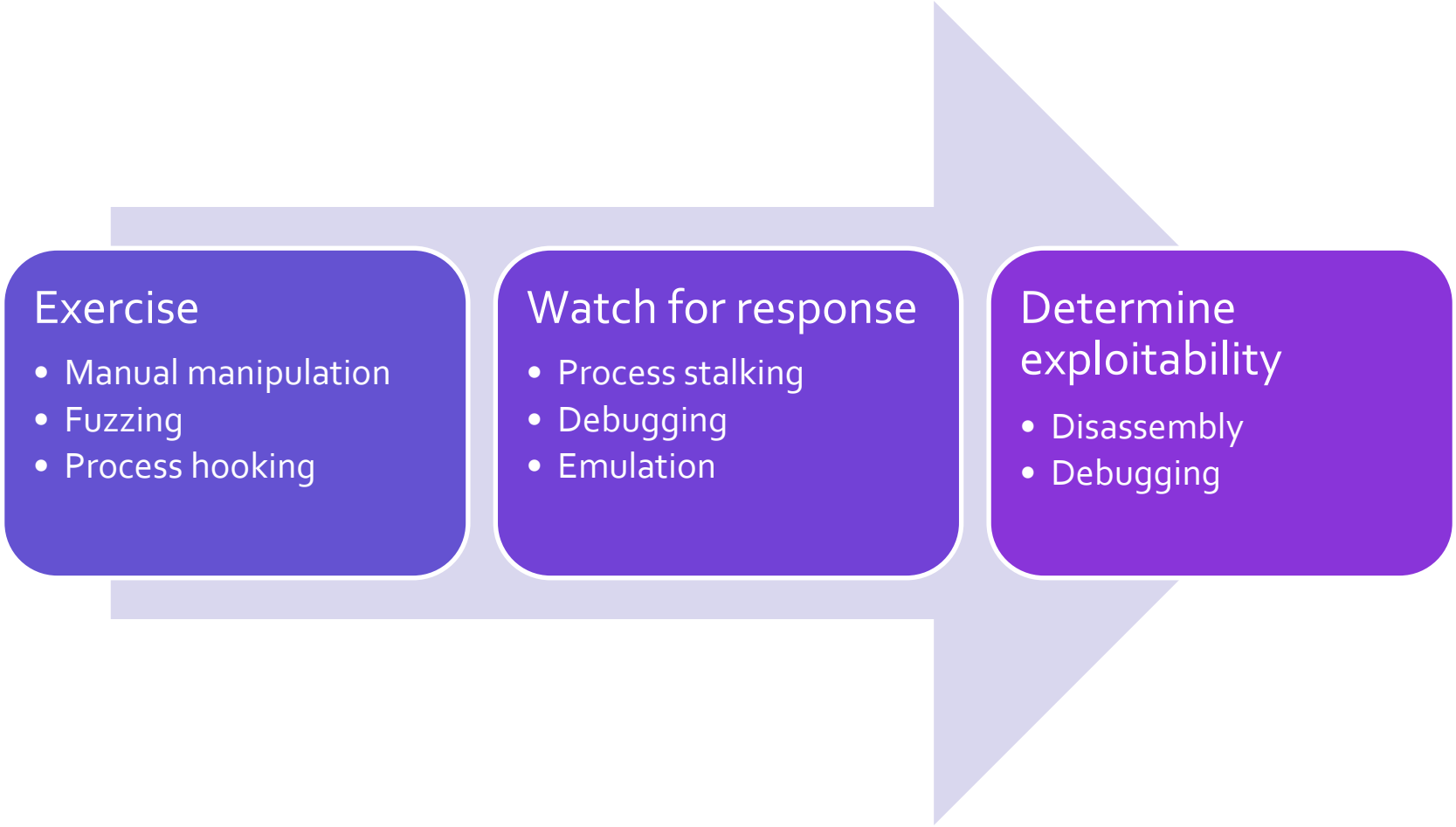
Academics

- Goals:
  - Finding common flaws and other general problems
  - Developing new crypto
  - Make something cool and useful
  - Make everyone safer
- Thoroughness:
  - Depth in area of research
- Access:
  - Creating new things
  - Blackbox

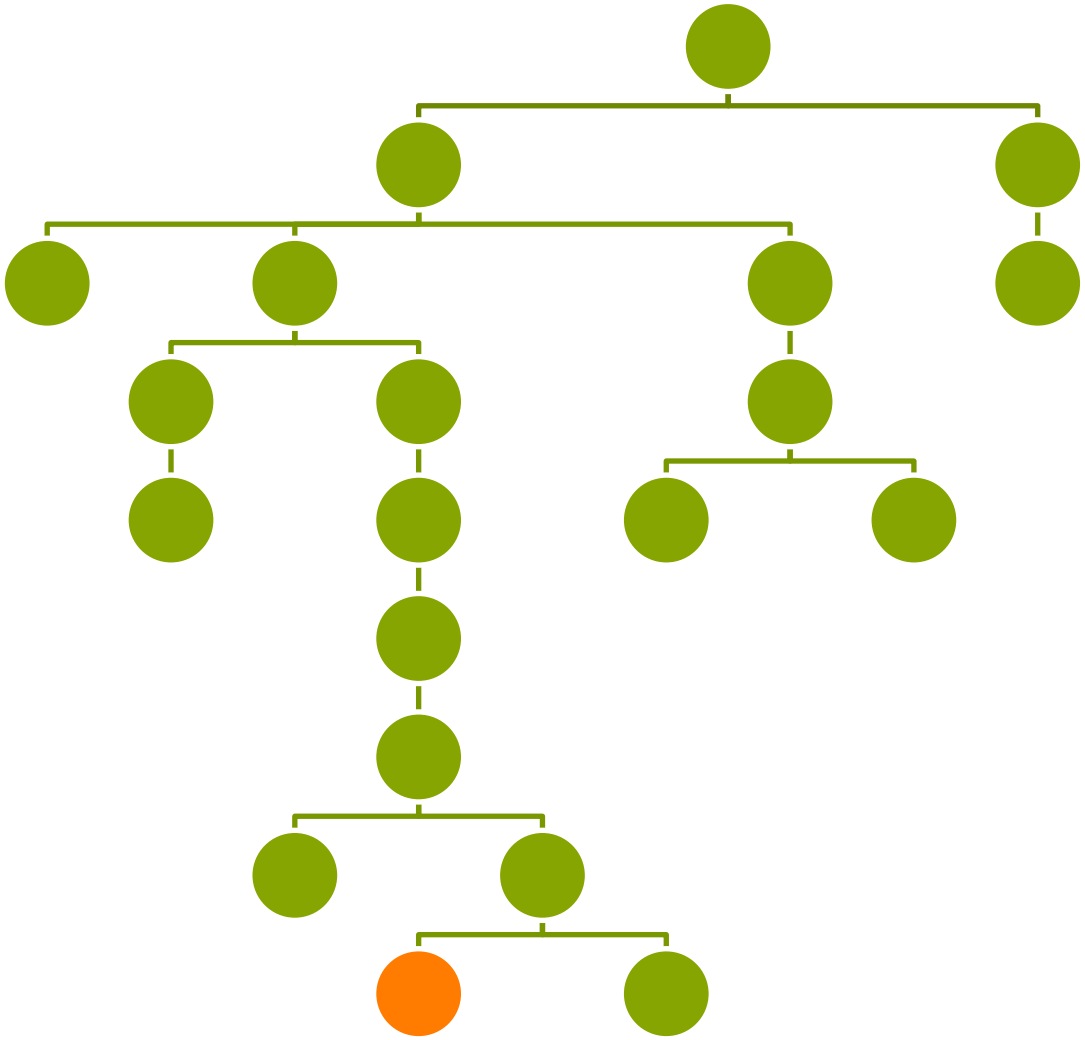
Bug Finding Techniques

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
0042DA24	81C7 4006000	ADD EDI,640	
0042DA2A	3B35 A40F4D0	CMP ESI,[4D0FA4]	
0042DA30	7C E4	JL SHORT 0042DA16	
0042DA32	E8 A9BB0400	CALL <JMP.&KERNEL32.GetTickC	KERNEL32.GetTickCount
0042DA37	8943 04	MOV [EBX+4],EAX	
0042DA3A	6A 01	PUSH 1	Time = 1 ms
0042DA3C	E8 A1BC0400	CALL <JMP.&KERNEL32.Sleep>	KERNEL32.Sleep
0042DA41	33C0	XOR EAX,EAX	
0042DA43	E9 51130000	JMP 0042ED99	
0042DA48	8B53 1C	MOV EDX,[EBX+1C]	
0042DA4B	3B15 A00C4D0	CMP EDX,[4D0CA0]	
0042DA51	74 2E	JE SHORT 0042DA81	
0042DA53	FF73 1C	PUSH [DWORD EBX+1C]	<%08lX> => [4D0CD4] = 0
0042DA56	FF73 18	PUSH [DWORD EBX+18]	<%08lX> => [4D0CD0] = 0
0042DA59	68 33244900	PUSH OFFSET ollydbg.00492433	Format = "Event %08lX from different pr
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 80010001	ContinueStatus = DBG_EXCEPTION_NOT_HAND
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.Continuel	KERNEL32.ContinueDebugEvent
0042DA7A	33C0	XOR EAX,EAX	
0042DA7C	E9 18130000	JMP 0042ED99	
0042DA81	833B 00	CMP [DWORD EBX],0	
0042DA84	75 22	JNE SHORT 0042DAA8	
0042DA86	68 61244900	PUSH OFFSET ollydbg.00492461	Format = "OllyDbg received debug event,
0042DA8B	E8 EC3BFDFF	CALL 0040167C	ollydbg.0040167C
0042DA90	59	POP ECX	
0042DA91	68 01000180	PUSH 80010001	ContinueStatus = DBG_EXCEPTION_NOT_HAND
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.Continuel	KERNEL32.ContinueDebugEvent
0042DAA1	33C0	XOR EAX,EAX	
0042DAA3	E9 F1120000	JMP 0042ED99	
0042DAA8	E8 97170000	CALL 0042F244	collydbg.0042F244
0042DAAD	8B53 18	MOV EDX,[EBX+18]	
0042DAB0	83FA 09	CMP EDX,9	Switch (cases 1..9, 10. exits)

Dest=0042ED99

Registers (FPU)

```
EAX 00000000
ECX 000000C0
EDX FFFFFFFF
EBX 004D0CB8 ollydbg.004D0CB8
ESP 0012D940 ASCII "\HJ"
EBP 0012F7FC
ESI 012ECEBC
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 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr 00000006 ERROR_INVALID_HANDI

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 77D3CD0F 000782F0
ST4 empty 3.1474364331401499740e-4932
ST5 empty 536.0000000000000000
ST6 empty 511.0000000000000000
ST7 empty 554.5000000000000000
               3 2 1 0      E S P U O Z I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0
FCW 1372  Prec NEAR,64  Mask    1 1 0 0
Last cmnd 0000:00000000
```

Address	Hex dump	ASCII
0047A000	00 20 F9 59 45 00 00 20 9F 21 46 00 00 00 98 A3	. ·YE.. ƒ!F...ÿü
0047A010	46 00 00 20 A4 A3 46 00 00 1C 88 93 47 00 00 20	F.. ñüF..LêôG..
0047A020	88 A8 46 00 00 20 B4 A9 46 00 00 00 24 BC 46 00	êżF.. ┤rF...$ªF.
0047A030	00 00 B8 BE 46 00 00 05 D8 E6 46 00 00 04 84 D2	..╕ªF..♣╤µF..◆äπ
0047A040	46 00 00 04 50 FF 46 00 00 0A C0 25 47 00 00 0A	F..◆P F...└%G...
0047A050	EC 2D 47 00 00 0A F4 28 47 00 00 0A E0 32 47 00	∞-G...⌠(G...α2G.

0012D940	004A485C \HJ.
0012D944	00000000 ....
0012D948	012ECEBC ªπ╓.☺
0012D94C	0018D1B8 ╕╤↑.
0012D950	0000008E A...
0012D954	004DC300 .├M.
0012D958	00000008 ◘...
0012D95C	00000000 ....

ASCII "Black on whit

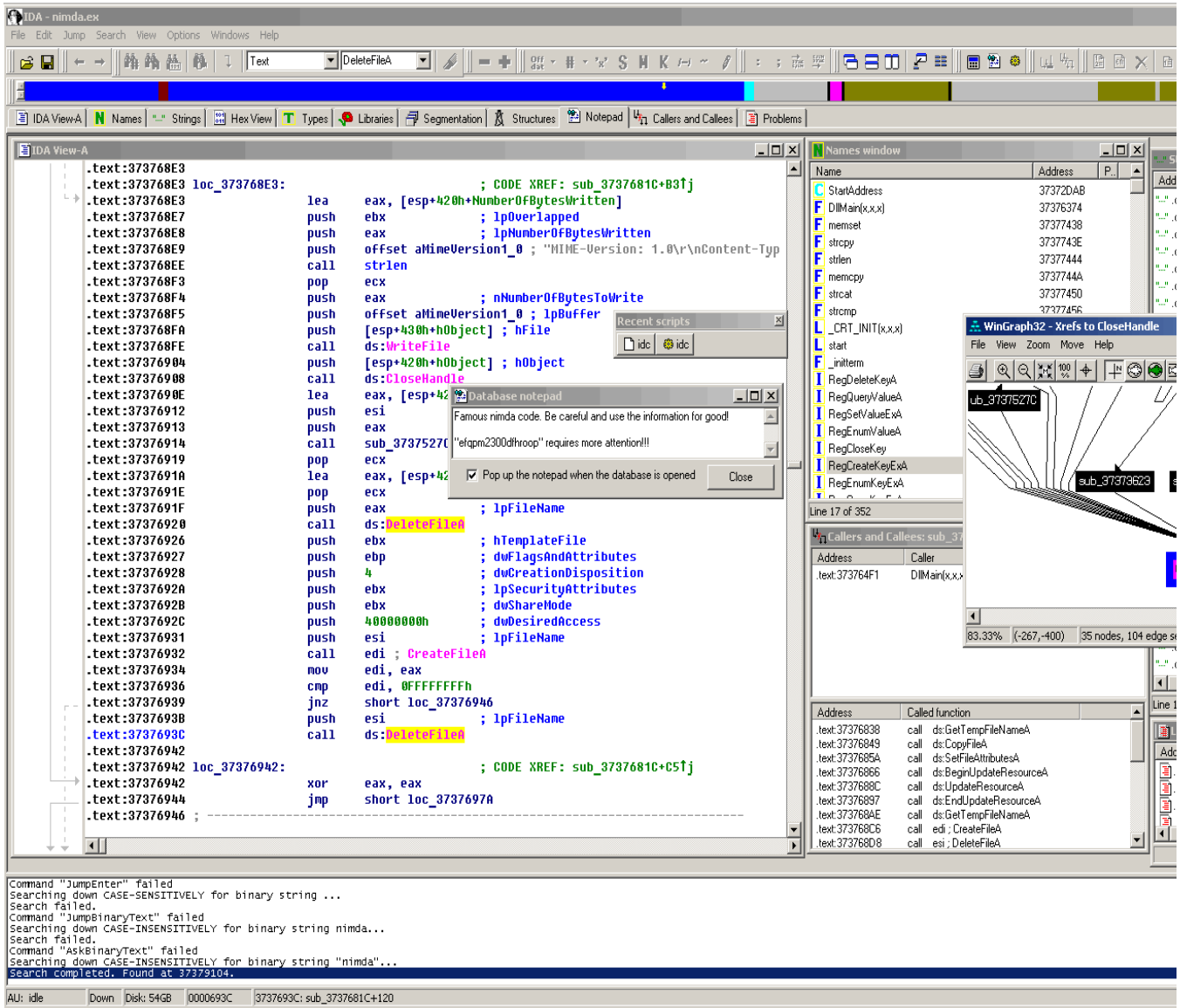
00401000	Entry point of main module
690A0000	Module C:\WINNT\system32\PSAPI.DLL
77920000	Module C:\WINNT\system32\IMAGEHLP.DLL
72A00000	Module C:\Oleg\Odbg2\DBGHELP.DLL
6E420000	Module C:\WINNT\system32\INDICDLL.dll
75E60000	Module C:\WINNT\system32\IMM32.dll
0042DA41	User code reached
0042DA41	User 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



Reversing Patches - BinDiff

Project Help

Search jp

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:004010E8ASN1PERDecU32Val
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:004010B0ASN1PERDecU16Val
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

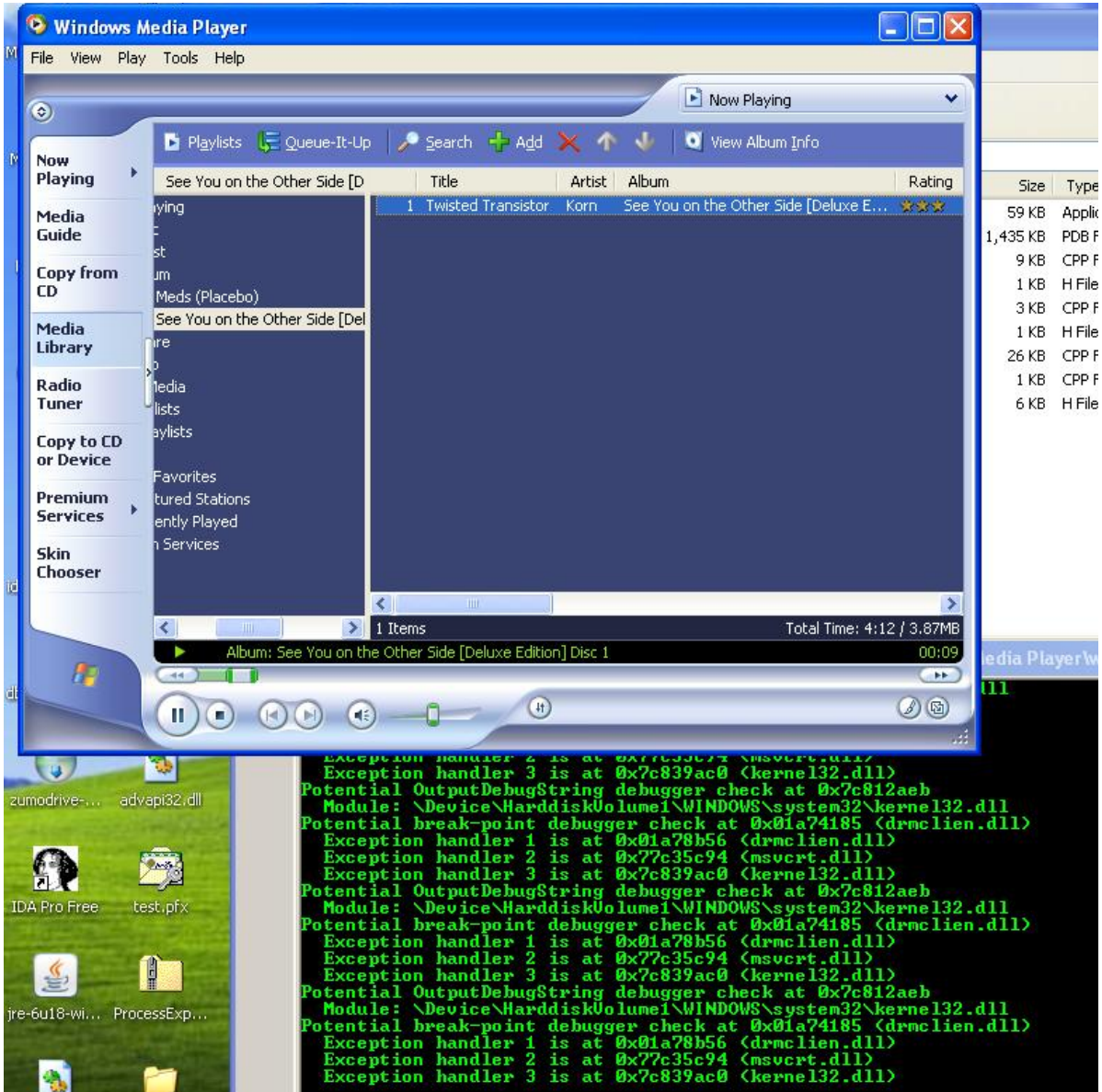
Basic Block		Address
push	ebx	**0042b98e**
push	esi	0042b98f
push	edi	0042b990
mov	edi,[esp+arg_4]	0042b991
push	edi	0042b995
push	4	0042b996
push	[esp+8+arg_0]	0042b998
call	ds:004010E8ASN1PERDecU32Val	0042b99c
test	eax,eax	0042b9a2
jz	short 0042B9D4loc_42B9D4	0042b9a4
inc	dword ptr[edi]	**0042b9a6**
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:004010B0ASN1PERDecU16Val	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



Snitch Output on WMP

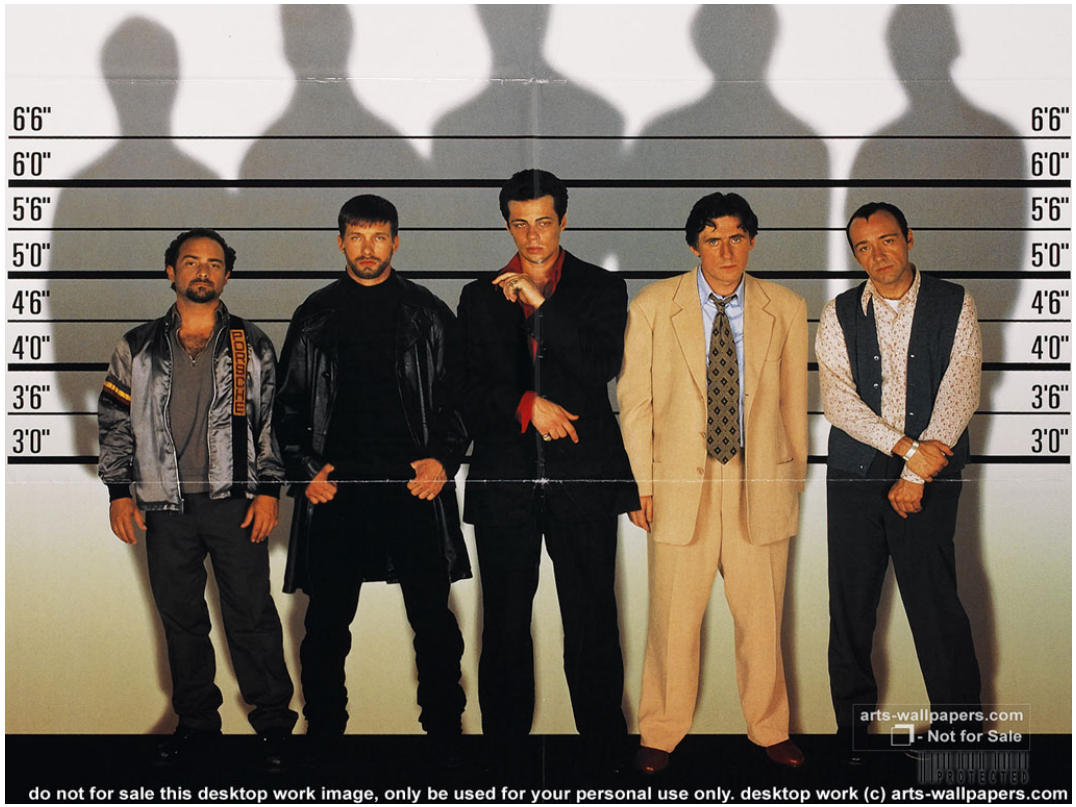
```
Potential break-point debugger check at 0x4bf9f889 (blackbox.dll)
  Exception handler 1 is at 0x4bf9fe71 (blackbox.dll)
  Exception handler 2 is at 0x7c839ac0 (kernel32.dll)
Potential break-point debugger check at 0x4bf9f9fc (blackbox.dll)
  Exception handler 1 is at 0x4bf9fe71 (blackbox.dll)
  Exception 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

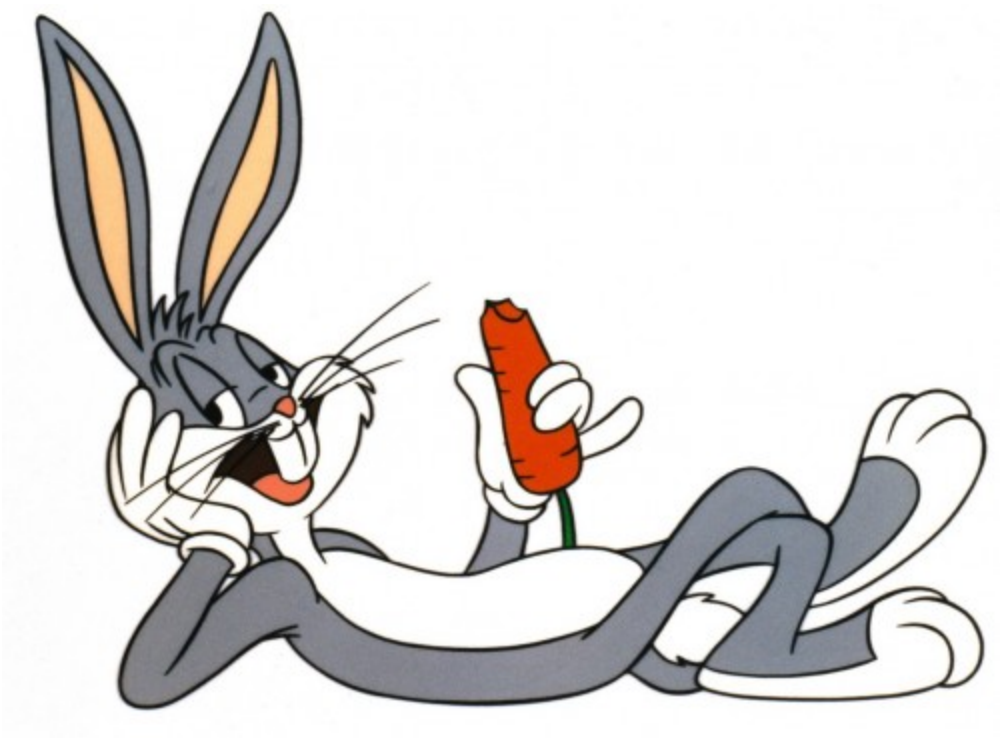
- Black Box techniques always work better with more context
  - More quickly triage flaws
  - Patch flaws much faster
- Analysis can start with source code
  - Look at sensitive areas
  - Use lexical analysis to give pointers
    - Flawfinder
    - RATS
  - Use semantic analysis
    - Coverity
    -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

Overall Goals

- All are looking for the similar things: vulnerable systems
- Let's dive in and look at vulns that we all look for



Sample of bugs found in the wild by us



Wild bug 1: (session management)

- Session cookie:
  - Looked random/binary
  - First 1/2 looks static for a given user, last 1/2 changes
  - Whole thing looks different for different users
- Crypto problem:
  - Cookie = RC4(fixedKey, username | date)
    = keystream $\oplus$ user1| timestamp
  - Server decrypts cookie and checks the username/ timestamp (why?)

Wild bug 1: (session management)

- What is RC4? (Ron's Code 4)
- Stream cipher leveraging XOR operations
- User1 cookie = keystream $\oplus$ (user1| timestamp)
- User2 cookie = keystream $\oplus$ (user2|timestamp)

Wild bug 1: (session management)

- What is RC4? (Ron's Code 4)
- Stream cipher leveraging XOR operations
- User1 cookie = keystream $\oplus$ (user1| timestamp)
- User2 cookie = keystream $\oplus$ (user2|timestamp)
- XOR either cookie by: user1 $\oplus$ user2
- Complete authentication bypass
- What does encryption get us?

Wild bug 2: (commerce)

- Sellers can post/modify goods
  - PostItem(description, price): sellers post items
  - ModifyPrice(itemID, newPrice): used by sellers to modify item prices
- Buyers can purchase goods
  - ViewItem(itemID): used by the web app when potential buyers click on an item
  - PurchaseItem(itemID): used during purchase flow
- What's a good place to look for a vuln?

Wild bug 2: (commerce)

- Flaw: Access controls didn't verify itemID is tied to the authenticated user (only enforced by UI)
- Impact: Shop for free

Wild bug 3: (server monitor)

Administrative page (internal) could monitor usage statistics through a web application

- Heaviest users (username and actual name)
- Load of services

Wild bug 3: (server monitor)

Flaw 1:

- Actual name wasn't sanitized when gathering statistics (only the username is sanitized)
- Actual name reappeared on the monitor page
- Try to inject code on the admin page

Wild bug 3: (server monitor)

- Challenge 1: must create a valid JSON object (a quoted string name/value pair object)
- Challenge 2: can't use capitalized characters
- Challenge 3: do something awesome

Wild bug 3: (server monitor)

- Challenge 1: must create a valid JSON object (a quoted string name/value pair object)
- Challenge 2: can't use capitalized characters
- Challenge 3: do something awesome
- Flaw 2: CSRF (doing an action as the logged in admin)
- `Actual Name: \" onmouseover=\"$.get(`**`<CSRF cmd>`**`')\"`

Wild bug 3: (server monitor)

- Flaws:
  - Tried to ignore actual name, but added it back in
  - Relied on insufficient input filtering and output encoding
  - Exploited the application's vulnerability to CSRF
- Impact: bridging the gap into the intranet

Vulnerabilities often need to be combined before exploit

Wild bug 4: (operating system)

```
void attachSuffix(char *userinputprefix, size_t inputSize)
{
        char    *withSuffix;

        withSuffix = malloc(inputSize + 2);
        if (withSuffix == NULL)
        {
                //meh, don’t care that much
                return;
        }
        memcpy(withSuffix, userinputprefix, inputSize);
        withSuffix[inputSize] = ‘.’;
        withSuffix[inputSize + 1] = ‘\n’;
        …
}
```

Wild bug 4: (operating system)

- Flaw: Integer overflow into heap overflow
- Impact: memory corruption or potential code execution

Wild bug 5: (forum text parser)

Some crazy person decided there should be double spaces after periods*.

Pseudo-Code: Loop through the text byte by byte. If a period and a single space (followed by a non-space character) is encountered in an text, do the following:

1. Allocate a larger buffer if there isn't room in the existing one (custom memory management)
2. memmove() the buffer following the ". " one byte over
3. Write an additional space character in the vacated spot
4. Loop back to #1 until entire text is processed

** That person is terrible and enforcing "grammar rules" from the age of typewriters.*

Wild bug 5: (forum text parser)

- What's the worst case runtime for parsing a text?

Wild bug 5: (forum text parser)

- What's the worst case runtime for parsing a text?
  - O(n^2)
- What happens if you send in a text of only ". "?

Wild bug 5: (forum text parser)

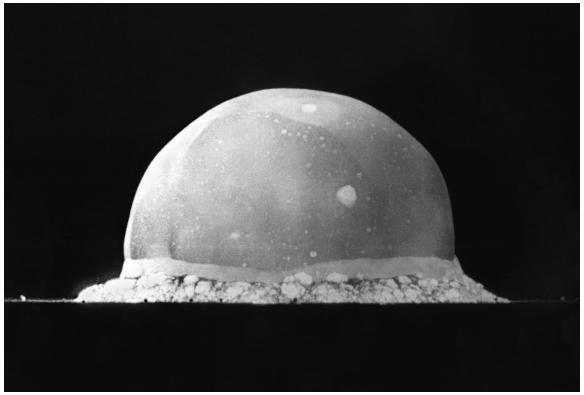
- What's the worst case runtime for parsing a text?
  - O(n^2)
- What happens if you send in a text of only ". "?
  - DoS (asymmetric effort between attacker and server)

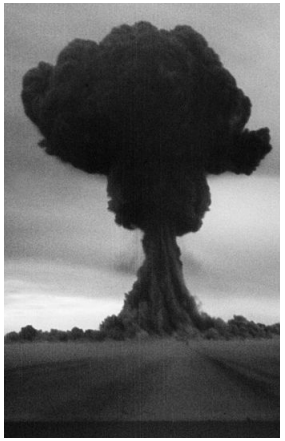
Bug Finding as Warfare

The Trickle Down Effect

- Innovations in warfare always decrease the cost for later adopters.

The Trickle Down Effect





The Trickle Down Effect

- How about with cyber warfare?

- Mid-2000's Nation State APT:
  - Spear-phish
  - Exploit tied to intelligence on AV
  - Active Directory attacks to spread horizontally
  - Access production data via internal interfaces

Well, maybe not that scary…

Operation Aurora

- Most public example of an "Advanced Persistent Threat"

- Advanced = 0-day custom malware
- Persistent = Slow and careful, non-financially motivated

- Hit at least 35 US companies in 2008-2009, five are clients

Aurora Attack Code

```
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;
}
```

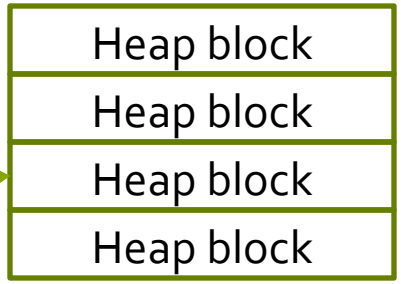
Operation Aurora

Use after free vulnerability (MS10-002 – Remote Code Execution in IE 5-8)

- Memory typically has a reference counter (how many people have a handle to me?)
- Improper reference counter allowed Javascript to still reference a function in a freed block of memory
  - Free memory
  - Heap spray attack code (likely it gets written to the freed block because of how IE memory management works)
  - Call function
  - Fairly reliable code execution

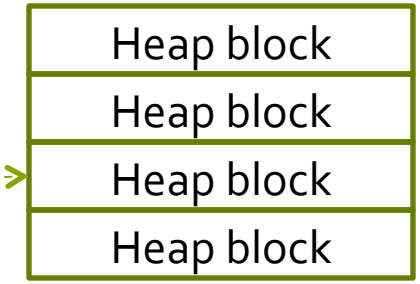
Operation Aurora

```
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;
}
```



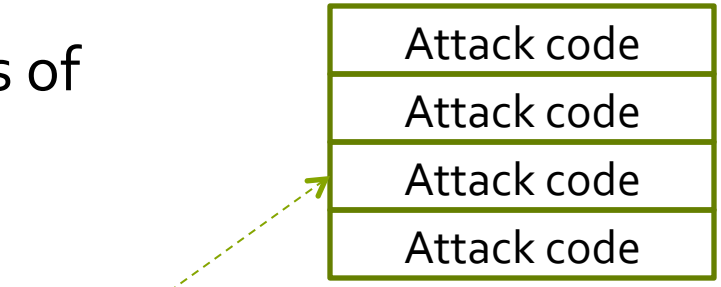
Operation Aurora

```
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;
}
```



Operation Aurora

- Heap Spray!
  - Create a bunch of elements with attack code and then free them (attack code gets written to lots of heap blocks)
  - IE Small Block Manager Reuses memory pages
- Call the event pointing to freed memory
- Code execution!



Operation Aurora

- Valuable exploit! How was it used?
- Social Engineering (get someone to click a link), almost always the weakest link
- Escalate privileges (cached credentials)
- Spread (Active Directory, brute force attack)
- Gather (source code, financial data)
- Exfiltration (to China, out of intranet on Christmas)

Operation Aurora

- Advanced Persistent Threat
  - Advanced attackers with talent (zero days) and time (months or years)
  - Targeted attacks (not just going after the vulnerable)
  - Non-traditional attacks, likely hard to monetize
- Whodunit?

Stuxnet

- [worm [rootkit [rootkit [sabotage]]]]
- Five zero-day vulnerabilities
- Two stolen certificates
- Almost surgically targeted
- Eight propagation methods
- Partridge in a malware pear tree

The Target

- Mixed MS Windows environment = *Redundant*
- Not exploiting memory corruption = *Reliable*
- Target: Iranian air-gapped networks operating centrifuges to enrich nuclear material (Natanz)
- How can you get a foot in the door? USB keys

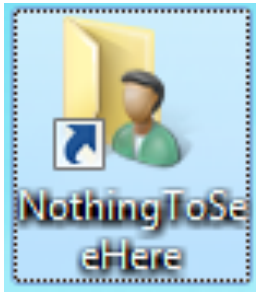
USB Vulnerability

Zero-Day* Vulnerabilities:

- **MS10-046 (Shell LNK / Shortcut)**
- MS10-061 (Print Spooler Service)
- MS10-073 (Win32K Keyboard Layout)
- MS08-067 (NetPathCanonicalize()), (Patched)

  http://www.phreedom.org/blog/2008/decompiling-ms08-067/
- MS10-092 (Task Scheduler)
- CVE-2010-2772 (Siemens SIMATIC Static Password)

MS10-046 (Shell LNK/Shortcut)



- You know, shortcuts and such
- Where does the icon come from?
- Loaded from a CPL (Control Panel File) specified by the user
- A CPL is just a DLL
- USB keys have attack DLL and a shortcut referencing the DLL
- Plugging in the USB stick leads to arbitrary code execution

MS10-046 (Shell LNK/Shortcut)

Flaw: we should run a user-specified DLL to display an icon for a shortcut?!

Pop Quiz: Which techniques that we have discussed could lead to discovery of this flaw?

A) Fuzzing
B) Disassembly
C) Debugging
D) BinDiff

I Love Security, What's Next?

- Ethics in security
- Possible Careers

Ethics in Security

- Big ethical debates used to be:

Responsible vs Full Disclosure

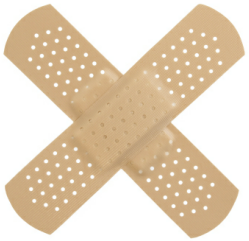
Ethics in Security

- Big ethical debates used to be:

  Responsible vs Full Disclosure

- Debate has shifted to:

  Disclosure vs Selling Weapons



Careers in Security

- Shape your job around your ethical standpoint, not vice versa

Careers in Security

- Shape your job around your ethical standpoint, not vice versa
- Write security relevant software

Careers in Security

- Shape your job around your ethical standpoint, not vice versa
- Write security relevant software
- Write (more) secure software

Careers in Security

- Shape your job around your ethical standpoint, not vice versa
- Write security relevant software
- Write (more) secure software
- Be a criminal

Careers in Security

- Shape your job around your ethical standpoint, not vice versa
- Write security relevant software
- Write (more) secure software
- Be a criminal
- Academia

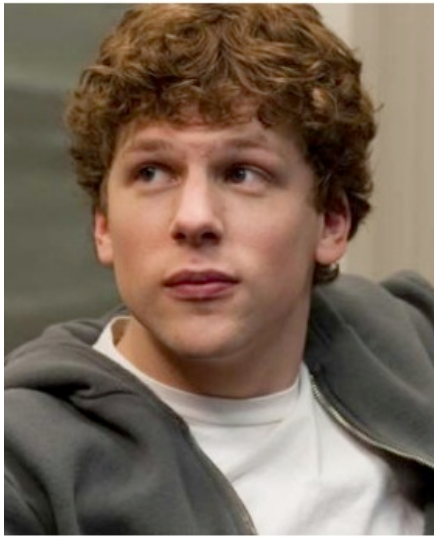
Careers in Security

- Shape your job around your ethical standpoint, not vice versa
- Write security relevant software
- Write (more) secure software
- Be a criminal
- Academia
- Pen testing!

What not to do…

- Go into a big company blindly

- Start your own company and think this is going to happen



Career Tips

- Always go into a meeting knowing what you want the outcome to be
- Ask for a small raise after an offer
- Common Stock is for Commoners
- Be part of the product, not the plumbing

Questions?

alex@stamos.org

@alexstamos