Where do security bugs come from?
Any how do you find them in the real world?

Stanford CS 155
Spring 2015

Alex Stamos
Your humble narrator

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

• What is happening right now?
• 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?
Year in Review: Fundamental Bugs
Year in Review: Breaches
Year in Review: User Impact
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

Exercise
  - Manual manipulation
  - Fuzzing
  - Process hooking

Watch for response
  - Process stalking
  - Debugging
  - Emulation

Determine exploitability
  - Disassembly
  - Debugging
Fuzzing
“Smarter Fuzzing”

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

http://lcamtuf.coredump.cx/afl/
Debugging
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
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
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
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
err = sslRawVerify(...);
Overall Goals

- All are looking for the similar things: vulnerable systems
- Let’s dive in and look at vulns that we all look for
Bugs found in the wild
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 = $\text{RC}_4$(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 ⊕ (user1|timestamp)
- User2 cookie = keystream ⊕ (user2|timestamp)
Wild bug 1: (session management)

- What is RC4? (Ron’s Code 4)
- Stream cipher leveraging XOR operations
- User1 cookie = keystream ⊕ (user1|timestamp)
- User2 cookie = keystream ⊕ (user2|timestamp)
- XOR either cookie by: user1 ⊕ 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
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 3: (operating system)

- Flaw: Integer overflow into heap overflow
- Impact: memory corruption or potential code execution
Wild bug 4: (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 4: (forum text parser)

- What’s the worst case runtime for parsing a text?
Wild bug 4: (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 4: (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
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

1. Social Engineering (get someone to click a link), almost always the weakest link
2. Escalate privileges (cached credentials)
3. Spread (Active Directory, brute force attack)
4. Gather (source code, financial data)
5. 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

- 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/](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
Where are we going with cyber-weapons?

1. (Windows service / driver)
   - Loads (64 bit)
   - (Hard drive area)

2. (NTFS extended attributes / registry)
   - Loads (32 bit)

3. (kernel «VMEM.sys»)
   - Loads

4. 32 bit usermode dispatcher (disp.dll)
   - Contains (32 bit system)
   - Loads
   - Executes

5. Virtual file system with plugins
   - Contains
   - Loads
   - Executes
   - Plugins:
     - 64 bit:
       - Usermode dispatcher (disp.dll)
       - (hard drive area)
     - Data theft, traffic interception, keylogger, taking screenshots, C&C code, general libraries

Stage 2
Stage 3
Stage 4
Stage 5
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
- Consulting and 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
Career Tips

- Always go into a meeting knowing what you want the outcome to be
- Ask for a small raise after an offer
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
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