Web Application Security

Introduction to Secure Web Development

Overview

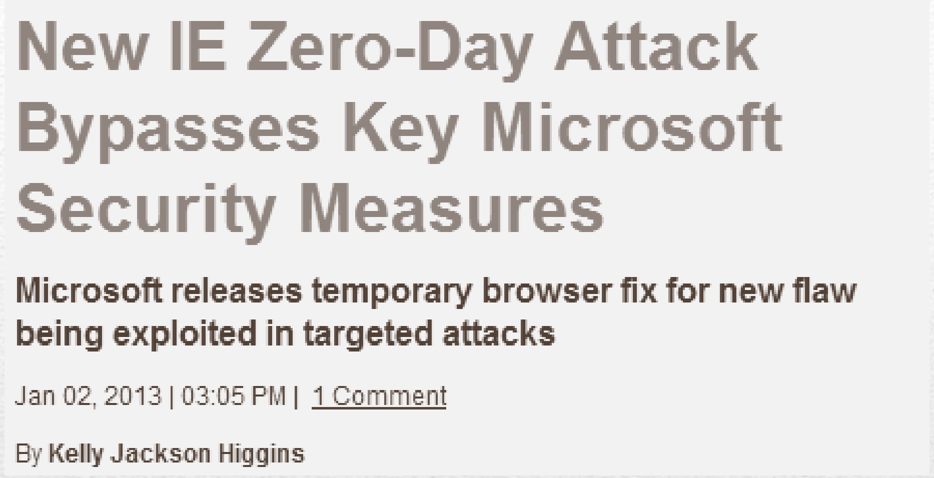
What is Software Security?

- From Technopedia:
  - "***Any compromise to integrity, authentication and availability makes a software unsecure. Software systems can be attacked to steal information, monitor content, introduce vulnerabilities and damage the behavior of software. Malware can cause DoS (denial of service) or crash the system itself.***"
- Who is responsible for introducing, and preventing the introduction of, security issues in software?
  - We are! We write the software.

What does the course cover?

- The course covers the following main topics:
  - Introduction to Software Security.
  - Web application security.
  - Using databases securely from a client application.
  - Introduction to C programming.
  - Buffer Overflows (using C).
  - Introduction to cryptology and its uses.
  - Privacy and legal issues surrounding software security.

Why Is Security Important?

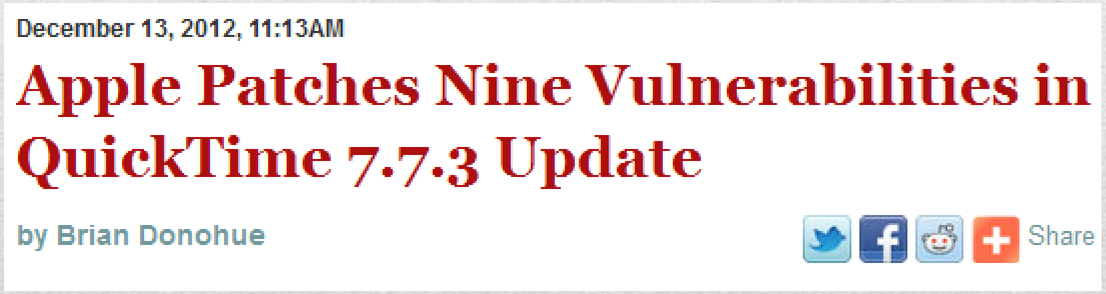


New IE Zero-Day Attack Bypasses Key Microsoft Security Measures

Microsoft releases temporary browser fix for new flaw being exploited in targeted attacks

Jan 02, 2013 | 03:05 PM | 1 Comment

By Kelly Jackson Higgins

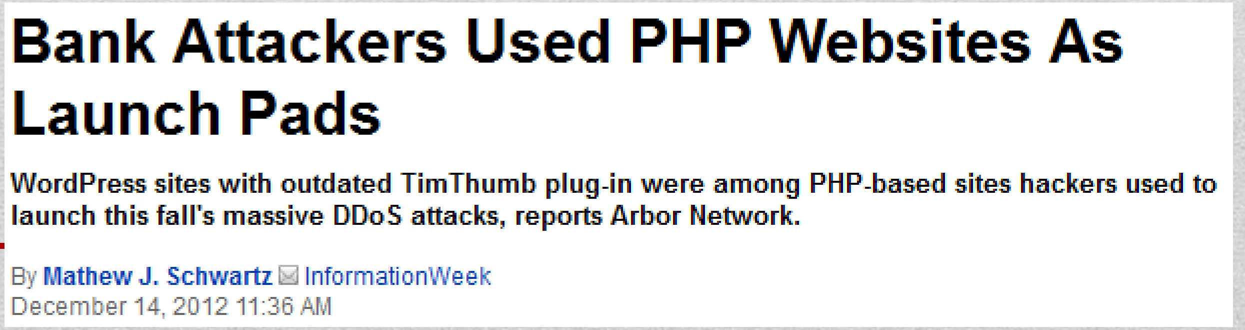


December 13, 2012, 11:13AM

Apple Patches Nine Vulnerabilities in QuickTime 7.7.3 Update

by Brian Donohue

Share

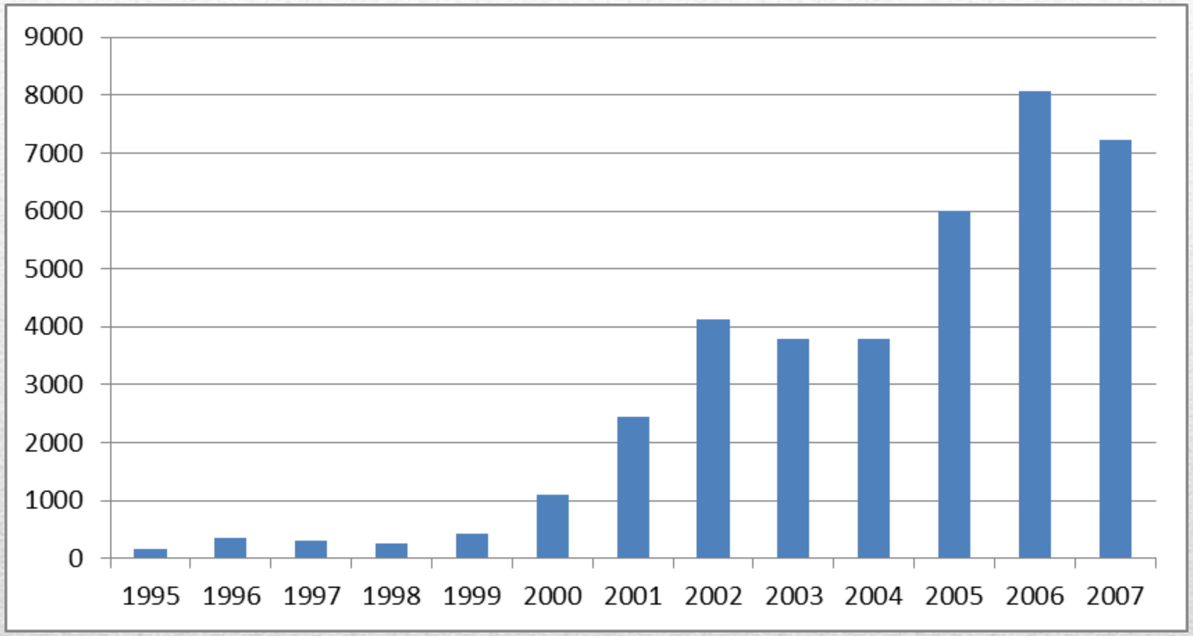


Bank Attackers Used PHP Websites As Launch Pads

WordPress sites with outdated TimThumb plug-in were among PHP-based sites hackers used to launch this fall's massive DDoS attacks, reports Arbor Network.

By Mathew J. Schwartz InformationWeek
December 14, 2012 11:36 AM

Why Is Security Important?



The number of software vulnerabilities reported to CERT/CC over 1995-2007

Why the Problem Is Growing?

- **Connectivity**
  - From PCs, to mobile devices, to supervisory control and data acquisition (SCADA) systems.
  - Networking makes remote attacks easy:
    - Access through a network does not require human intervention.
  - Ubiquity of networking → more software systems to attack.
  - Diversity of security implementation creates bottle-neck attack points.

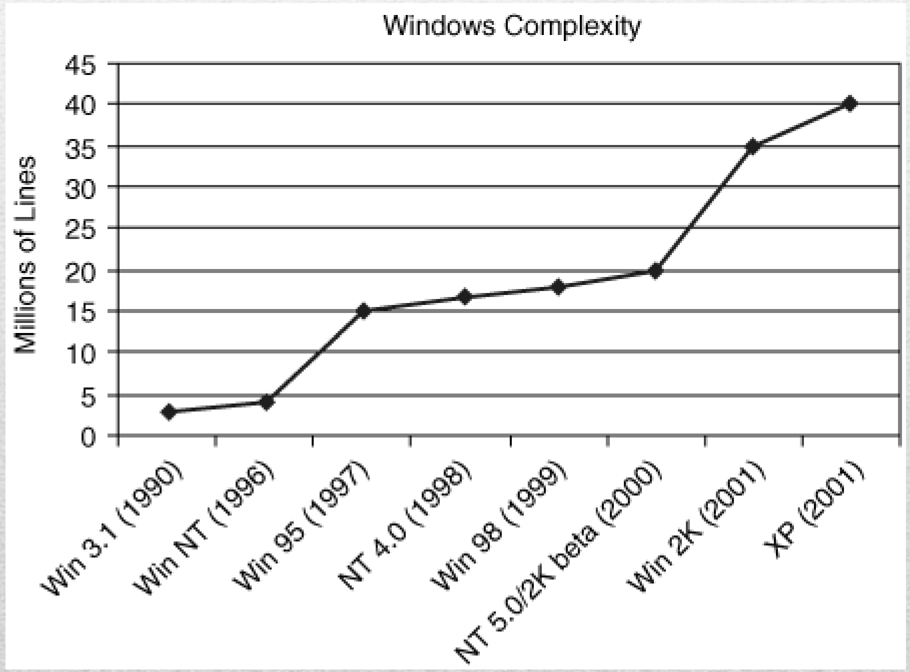
Why the Problem Is Growing?

- **Extensibility**
  - Software updates, plugins, scripting, applets, etc.
  - Service Oriented Architecture (SOA).
  - Economically interesting, but:

    Software vulnerabilities come as unwanted extensions.
  - E.g., advanced languages and platforms including Sun Microsystems' Java and Microsoft's .NET Framework.

Why the Problem Is Growing?

- **Complexity**
  - Comes with adding new features/improving functionality.
  - Perfect security check is impossible when code is larger than even a few 1000 lines!



Growth of the Microsoft OS code base over 1990-2001, taken from [Mc06].

Attacker’s Advantage - Defender’s Dilemma

- Defender must defend all points…
  - Attacker can choose the weakest point.
- Defender defends only against known attacks …
  - Attacker can probe for unknown vulnerabilities.
- Defender must be constantly vigilant…
  - Attacker can strike at will.
- Defender must play by the rules…
  - Attacker can play dirty.

Cost of Late Security ‘Fix’

1) Fix coordination/**planning**.
2) Developers **finding** the vulnerable code.
3) Developers **fixing** the code.
4) Testers **testing** the fix and its setup.
5) Creating and testing **international versions**.
6) Digitally **signing the fix** (if applicable).
7) **Posting** the fix to your website.
8) Writing the supporting **documentation**.
9) Productivity loss in **switching between projects**.
10) Customers **applying** the fix.
11) Handling **bad public relations**.
12) Potential **revenue loss**.

Paradigm Shift

- **Traditional software development paradigm:**
  - Quality of software means reliability and performance.
  - Software should be implemented / run faster.
- **Change your paradigm:**
  - Bringing security in after release has a lot of cost!
  - Learn from previous mistakes!
  - Build security when developing software!
  - Always assume the most hostile of environments!
  - Stay abreast of recent security flaws!

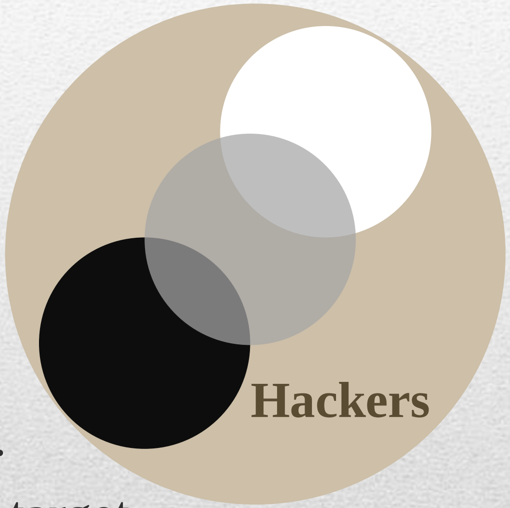
Hesitation about Instilling Security

- Security is usually **not the primary skill** or interest of the designers and developers creating the product.
- Security means **not doing something exciting** and new.
- Security is **boring**.
  - Not true! Will change when one knows more about security.
- Security is often seen as a **functionality disabler**.
  - Yeah! But disables functionality to **ATTACKER**.
- Security is **difficult to measure**.
  - True! But not a good reason to create insecure products.
  - There are mechanisms for measuring security.

Terminology

Hackers vs. Attackers

- **Hacker:**
  - An expert programmer who tries to find vulnerabilities.
  - **White-hat hackers:**
    - Good intentions.
    - Inform organizations.
    - Publish vulnerabilities & exploits.
  - **Black-hat hackers (Attackers):**
    - Malicious, but not always highly-skilled.
    - Use/write exploits to attack a vulnerable target.
    - Motivation: financial/personal/political/etc.
  - **Grey-hat hackers:**
    - In between, do this as a hobby



Defects, Bugs, Flaws, & Risk

- **Defects:**
  - Vulnerabilities in both implementation and design.
  - May lie dormant in software & cause major consequences.
  - Divided into two types: **bugs** and **flaws**.
    - **Bugs:**
      - Implementation-level software errors.
      - Easy to find and remedy, using tools, e.g., FIST.
    - **Flaws:**
      - Deeper-level problems, instantiated in implementation but may be rooted at design.
    - **Risks:**
      - The damaging consequences of flaws & bugs.

Managed vs. Unmanaged

- Different programming languages work in different ways (obviously)
  - OO vs. structured.
  - Compiled vs. interpreted.
  - Weakly vs. Strongly typed.
  - Memory management:
    - ***Unmanaged***: programmer is responsible for writing code to allocate, use and deallocate memory in a running program (amongst other things, e.g security).
      - Example: C, C++, etc.
    - ***Managed***: something else, e.g. the Java JVM, is responsible for memory management, e.g. garbage collection, and provides other functionality that a running program can use, e.g. security
      - Example: Java, C#, etc.

Examples of some Different Types of Code Security Vulnerabilities

Buffer Overflow

- ***Major*** source of vulnerabilities, esp. C-code family.
- **Cause**: input data longer than reserved space in memory(stack / heap).
    - "***Buffer***": an area of RAM used to temporarily store data in a program.
    - Scenario:
        - Program allocates a fixed-size area of memory, a ***buffer***, to hold some data.
        - User provides more data than can be stored in the buffer, e.g. via keyboard or file.
        - Data "overflows" buffer into memory addresses following end of buffer.
            - Might go unnoticed.
            - Might corrupt data already stored in the "overflow area"!!

Unvalidated Input

- **Unvalidated Input**
  - Classic security flaw: ***SQL Injection in a web application***

  - Toy code sample:

if $userProvidedPassword matches $dbPassword then….

User:	hacker
Password:	admin OR 1==1

Access Control, Cryptosystems

- ***Access-control problems***
  - Careless ***permissions*** for files access and process control.
  - Attackers wish for ***root/admin*** privileges (unrestricted permissions).
    - Attack can start with a flaw in the program that allows the attacker to take control of a part of the program as ***root/admin***.
  - Attack can also be caused by not setting access control rules to files/directories for users/programs.

- ***Weak use of Cryptographic systems***
  - Defining weak private keys/randomness which are predictable.
  - Using weak encryption/authentication/authorization functions.
  - Using encryption/authentication/authorization functions in a weak way/mode:
    - MD5, CRC, WEP, DES

Race Conditions

- ***Race Conditions***
  - Program's behavior depends on the order of threads/events (multithreaded/multiprocess programs).

  - An example: time-of-check vs. time-of-use.
    - <u>Application</u>:
    1. Chooses a filename say '***file_i***'.
    2. Checks whether file ***file_i*** in a public folder, say ***pubFold***.
       - If no creates (with ***restricted permissions***), opens it, and writes data to it.
       - If yes sets i=i+1 and repeats the check in 2.
    - <u>Attacker</u>:
      - Creates an ***file_i*** (with ***<u>un</u>restricted permission***) between check and create functions, above.

SD Security-related Sites/Projects To Remember!

CERT/CC

- **Computer Emergency Response Team - Coordinator Centre:** www.cert.org
  - Services:
    - Secure coding/language standards.
    - Secure development tools/libraries.
    - Vulnerability discovery/remediation.
    - Research/Awareness/Training.
    - Governance/forensics.
  - Has been extended internationally, with name “CSIRT”.
    - US-CERT, by Homeland Security.

SANS

- **Sysadmin, Audit, Networking, and Security:** www.sans.org
  - Annual top 25's in corporation with CWE (Common Weakness Enumeration project, http://cwe.mitre.org).
    - Top 25 software errors (3 categories):
      - Insecure Interaction Between Components.
      - Risky Resource Management.
      - Porous Defenses.
    - Top 25 programming errors.
    - Top 25 Q&A.

Bugtraq

- **Bugtraq electronic mailing list:** http://www.securityfocus.com/archive/1
  - Issues new computer security issues/vulnerabilities.
  - Provides:
    - Issue title.
    - Issue status.
    - Overview/details.
    - Proof of concept.

OWASP

- **Open Web Application Security Project:** www.owasp.org

  - Projects:
    - Articles/documentations, esp. book-length OWASP Guide.
    - Tools/technologies, esp. WebScarab and WebGoat.

Kali Linux (formerly *Backtrack*)

- A distribution of Debian-based Linux.
- With tools for digital forensics and penetration testing.
  - Metasploit
  - RFMON
  - Aircrack-ng
  - Gerix Wifi Cracker
  - Kismet
  - **Nmap**
  - Ophcrack
  - Ettercap
  - **Wireshark**
  - BeEF
  - Hydra

List of Ten Coding Practices

Top 10 List: Coding Practices

From: **https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices**

1. Validate Input.
2. Heed compiler warnings.
3. Architect and design for security policies.
4. Keep it simple.
5. Default deny.
6. Adhere to the principle of least privilege.
7. Sanitize data sent to other systems.
8. Practice defense in depth.
9. Use effective quality assurance techniques.
10. Adopt a secure coding standard.