1. Introduction to Software Security

Prof. Dr. Marc Rennhard
Institut für angewandte Informationstechnologie InIT
ZHAW School of Engineering
rema@zhaw.ch

Content

- Motivation for a Software Security Course
- Some examples of past security incidents and vulnerabilities
- Reasons for today's large number of security problems related to software security
- "Traditional" security measures that have been used in the past years
- Terminology and definitions

Goals

- You have an overview over some typical software security problems that we are facing today
- You can explain some of the reasons why we have so many software-related security issues today
- You understand and use the basic terminology correctly: bug, flaw, defect, vulnerability, threat, exploit, asset, risk, countermeasure

Motivation

Confidentiality – Integrity – Availability (CIA)

When talking about information security, we consider the following key concepts or security goals:

- Confidentiality
  - Sensitive data must be protected from unauthorized access
- Integrity
  - Data and systems must be protected from unauthorized modification
- Availability
  - The information must be available when it is needed

- Depending on the scenario, some of them may be more important than other...
- ...but in general, we talk about a "secure" application / environment / system, when CIA are satisfied "to a reasonable degree"

CIA – Exercise

- You are likely using many of the following information systems nearly every day

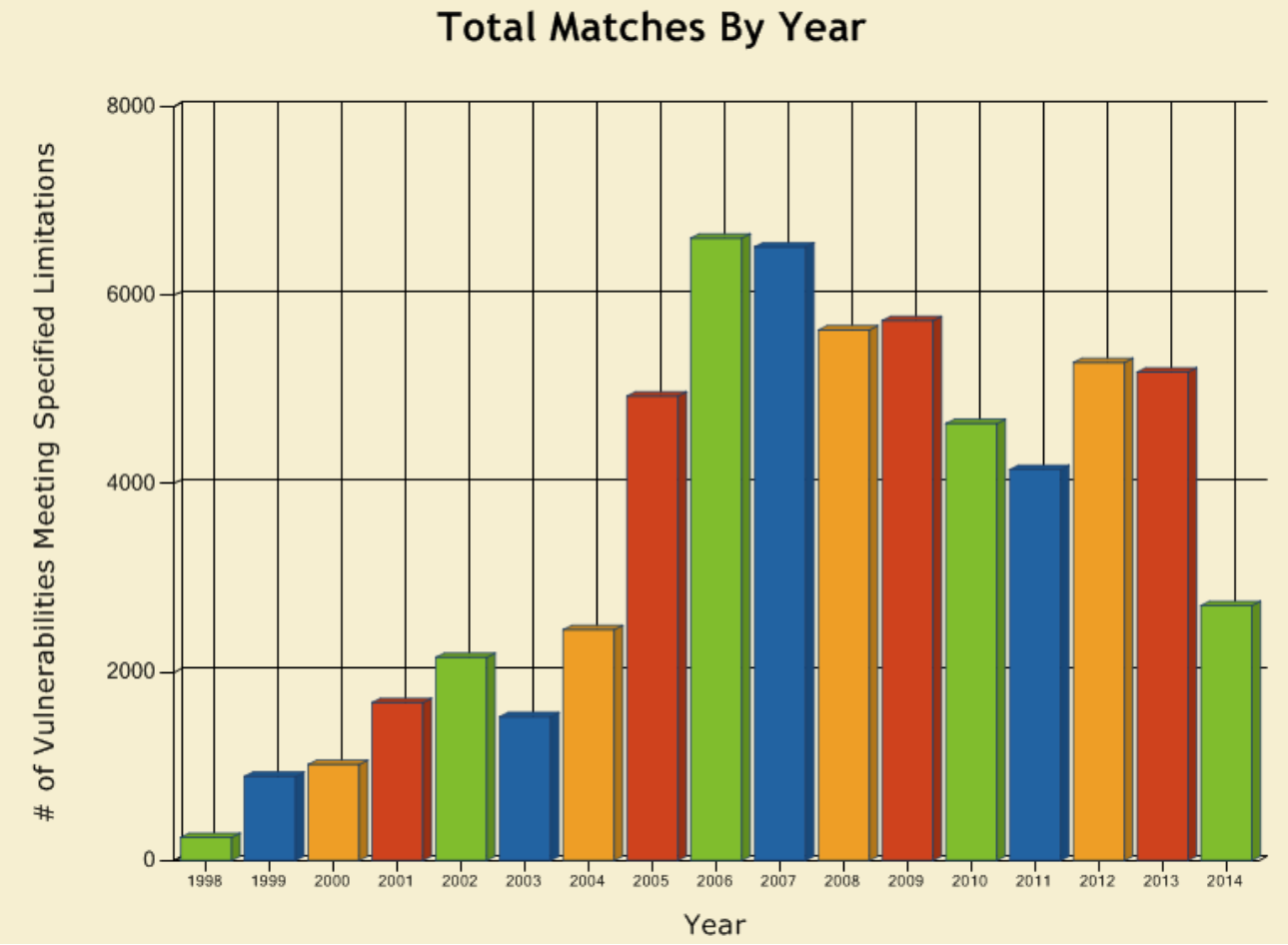
Search engines | Google Docs | e-mail | e-Shop | SBB timetable

e-Banking | OS update services | Evento | Online backup service

- Pick one or two of these systems (or any other you like) and think about the following:
  - How important are confidentiality, integrity and availability for you as the user?
  - Why?
  - Does the service actually provide this?

Reported Vulnerabilities

- Number of software vulnerabilities per year since 1998, that received a "Common Vulnerability and Exposure" (CVE) number (as of May 2014)



Conclusion:
The software we are using today is often full of defects!

What is needed for Secure Information Systems? (1)

- Basic security functions: cryptography, secure communication protocols, access control mechanisms...
  - E.g. make sure system components can communicate in a secure way
  - E.g. make sure users authenticate a system before sending it sensitive data

Main topic of course ISI (Internet-Sicherheit)

- Secure software
  - E.g. make sure your browser cannot be compromised by drive-by downloads
  - E.g. make sure there's no vulnerability in your e-shop so your customers can shop for free

Main topic of this course

Together with the topics of ISI, this course provides the capabilities to design, implement and test secure systems and applications

What is needed for Secure Information Systems? (2)

Beyond this, information security is concerned with further aspects that are not discussed in detail here, including e.g.:

- Security operations
  - E.g. regularly inspecting log files or using an IDS to detect security issues
- Security (incident) management
  - E.g. having appropriate processes when an issue has been detected
- Security awareness of users
  - E.g. train user not to open every e-mail attachment and install software from non-trusted sources
  - But: Don't use mistakes by users as an excuse when security fails, try to design and implement your systems such that the number of mistakes users can make are minimized!
- ...

Some Examples

Morris Worm (1)

- Written and launched in 1988 by Cornell University student Robert Morris (now a professor at MIT)
  - It became never totally clear why he did it (just as a proof of concept?)

- Considered as the first Internet worm
  - Worm: malware, that spreads and executes on its own, without requiring a host program (as is the case with viruses)

- Typical behaviour of a worm:
  - Scan other systems randomly to find targets
  - If a possible target has been found, exploit a vulnerability that may be present
    - In the case of the Morris worm, this included vulnerabilities in Unix systems (sendmail, finger, rsh and weak passwords)
  - If exploitation is successful, the infected system itself starts finding targets
  - This typically means that many (all) possible targets can be infected in a relatively short time span

Morris Worm (2)

- To avoid damage, Morris programmed the worm to only infect a newly found target if it hadn't been infected before
  - This is done by "asking" the remote computer...
  - ...but this allows administrators to easily defeat the worm
  - To avoid this, the worm was programmed to infect a remote computer in 14% of all cases even if a copy already appeared to be running
  - This mistake caused infected computers to eventually get unusable because of too many processes running on them

- Infected about 6'000 systems, est. damage: USD 10 - 100 millions

- Morris was convicted in 1990 to three years of probation, 400 hours of community service and a fine of USD 10'050

Morris Worm (3)

- Why is the Morris worm interesting from today's perspective?
  - It is assumed to be the first piece of malware that demonstrated the "mass infection phenomenon"
  - It is an example of a malware using the strategy: scan for possible targets, exploit vulnerabilities, infect target, scan for more targets... which remains a valid strategy until today
  - Modern malware is often much less "aggressive" in order to not to be discovered and rarely spreads from device to device in a fully automated way
- After the Morris worm, it took about 10 years before similar attacks started to appear at a large scale, driven by several factors
  - The massively growing Internet population during the second half of the 1990s and the growing number of services offered by companies
    - Resulting in many and attractive targets
  - The extremely poor security offered by the most popular operating system (Windows) and its applications
    - This was before personal firewalls and integrated update mechanisms became widespread (Windows firewall as part of Windows XP SP2 in 2004)

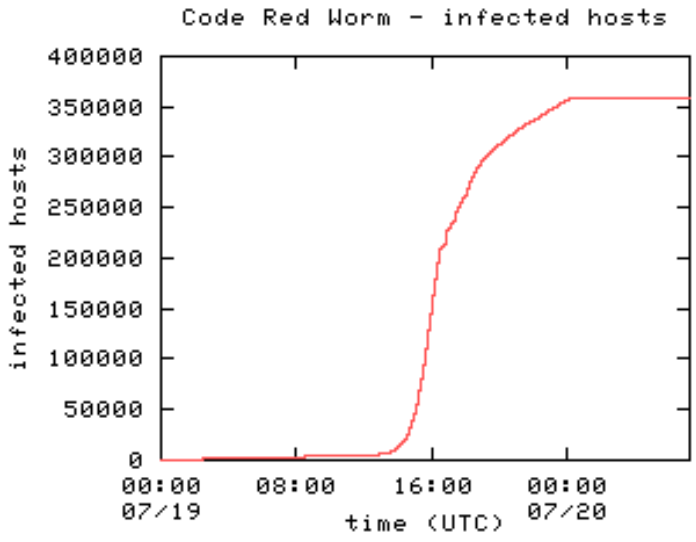
Code Red Worm (1)

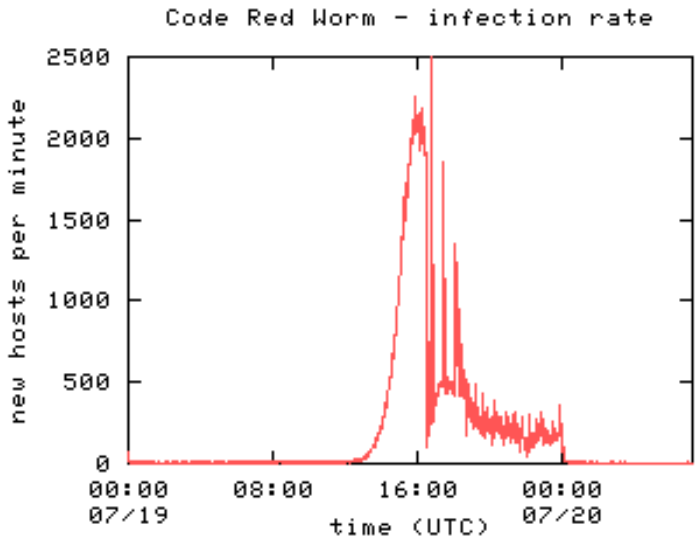
- Released on 13th July 2001, probably in the Philippines
- Targets computers running the Microsoft Internet Information Service (IIS) web server
  - Exploits a buffer overflow vulnerability in IIS that allows to execute an attacker's code on the web server (MS01-033)
  - A patch was made available by Microsoft a month before the attack
- Effect of the worm:
  - The attacked website displays "HELLO! Welcome to http://www.worm.com! Hacked By Chinese!"
  - If the system date is between the 1st and 19th of a month, the computer starts scanning for other vulnerable web servers and attacks them
  - If the system date is between the 20st and 28th, a DDoS attack against www.whitehouse.gov is launched (very little impact)

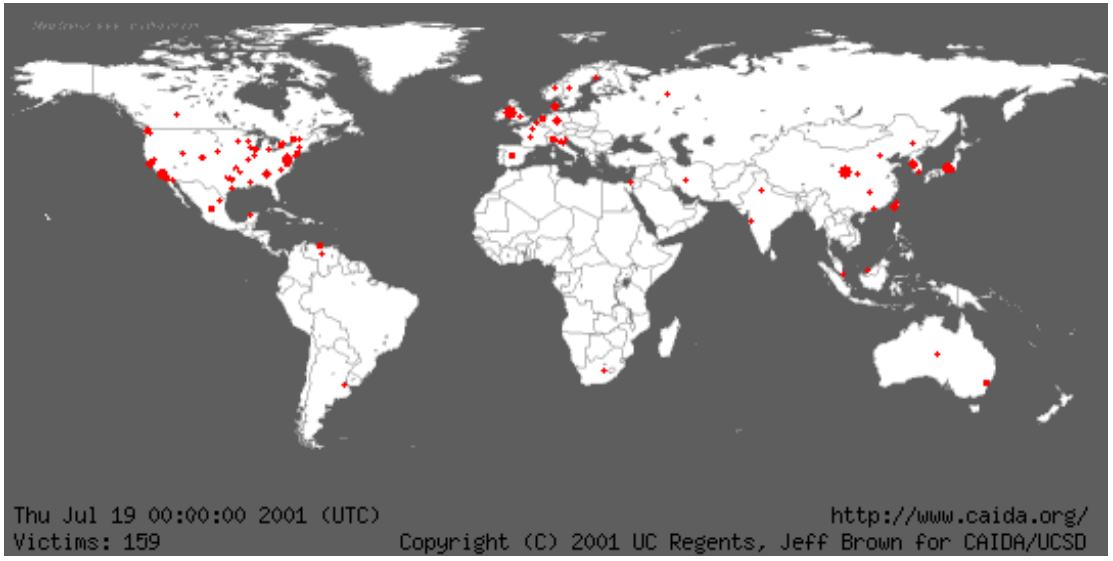
Code Red Worm (2)

- However, the first Code Red version (CRv1) had a minimal impact
  - The random number generator to generate IP addresses to scan used a static seed → all infected computers scanned the same other computers
  - The worm was only memory resident, so rebooting removed it
- A second version of Code Red (CRv2) appeared on 19th July 2001
  - Same vulnerability as the first version, same functionality
  - But: uses a more random seed to generate IP addresses to scan → potential to infect many more hosts!
- Effect of the second version
  - 359'000 infected web servers within 14 hours
  - Peak infection rate: 2'000 hosts per minute
  - Rendered the Internet virtually unusable by the huge amount of scanning traffic
  - The worm was still only memory resident, so rebooting was enough to remove the worm (but rapid re-infection was likely to occur…)
  - Luckily, the worm stopped spreading on 19th July 2001, 24:00 UTC, which gave the administrators time to patch the computers until 1st August…

Code Red Version 2 Analysis







Zotob Worm

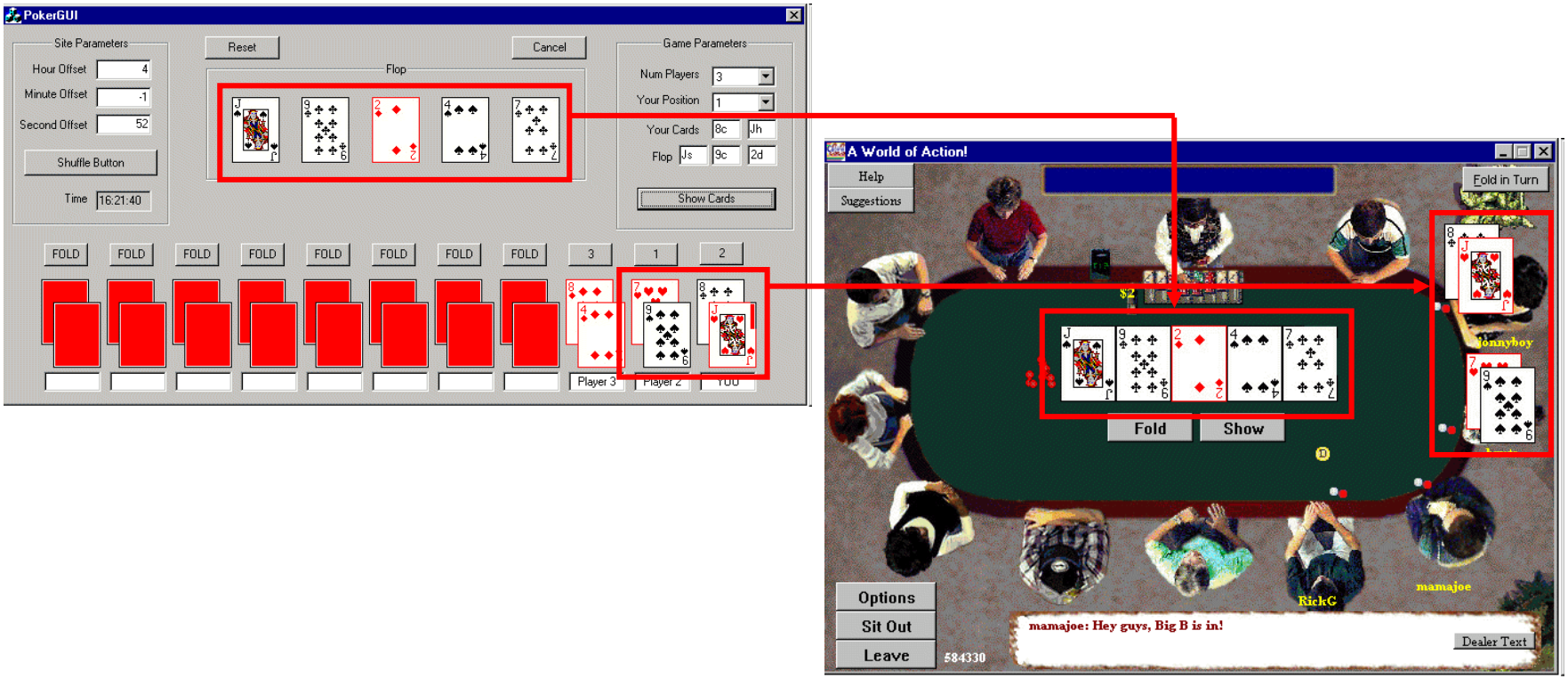
- Started spreading on 13th August 2005
  - Exploits a buffer overflow vulnerability in the plug and play component of Windows 2000 (via SMB protocol and ports 139 and 445)
  - Microsoft has released a patch only 4 days before the attack (MS05-093)
  - Caused minor direct problems by the worm propagation itself (CNN and some banks reported some problems) because it did not affect very many systems, estimation: 1'000 hosts compromised
- So what's interesting about Zotob?
  - Infected hosts are transformed into bots and connect to an IRC server controlled by the attacker → botnet
  - Gives the attacker the possibility to install software on the bots and control them at will (spy on users, attack other systems…)
  - First significant incident where automatically spreading malware is used as the basis to form a botnet (which is "state of the art" today)
- Suspected worm authors were arrested on 30st August 2005
  - Motivation: making money (install adware, spy on users…)

Online Poker Game Flaw (1)

- Detected by the Software Security Group at Reliable Software Technologies
- Exploiting the flaw allows knowing the cards in every opponent's hand as well as the next cards that will be dealt
- Problem: poor seeding of the pseudo random number generator
  - Uses the number of ms since midnight according to the server time
  - This allows to easily predict the output of the random number generator
  - As the output is used to shuffle the cards, they can be predicted as well
- Poor PRNG implementations have often resulted in serious security problems
  - Another example: Netscape browser seeded the PRNG with a combination of time of day, process ID, and parent process ID → this resulted in easy-to-guess 128-bit SSL keys

Online Poker Game Flaw (2)

- A demonstration program to exploit the flaw was written
  - First, the server time is determined by observing the game and trying different seeds
  - Once this is done, the program can compute the exact shuffling results of any game and therefore predict the sequence of the cards that will appear



CVE-2009-0081: Windows Kernel Input Validation Vulnerability

- **Original release date**: 03/10/2009
- **Overview**: The graphics device interface (GDI) implementation in the kernel in Microsoft Windows 2000 SP4, XP SP2 and SP3, Server 2003 SP1 and SP2, Vista Gold and SP1, and Server 2008 does not properly validate input received from user mode, which allows remote attackers to execute arbitrary code via a crafted (1) Windows Metafile (aka WMF) or (2) Enhanced Metafile (aka EMF) image file, aka "Windows Kernel Input Validation Vulnerability."
- **CVSS v2 Base Score**: 9.3 (HIGH)
- **Access Vector**: Network exploitable; Victim must voluntarily interact with attack mechanism
- **Impact Type**: Provides administrator access, Allows complete confidentiality, integrity, and availability violation; Allows unauthorized disclosure of information; Allows disruption of service

CVE-2009-1098: Buffer Overflow Vulnerability in Java

- **Original release date**: 03/25/2009
- **Overview**: Buffer overflow in Java SE Development Kit (JDK) and Java Runtime Environment (JRE) 5.0 Update 17 and earlier; 6 Update 12 and earlier; 1.4.2_19 and earlier; and 1.3.1_24 and earlier allows remote attackers to access files or execute arbitrary code via a crafted GIF image, aka CR 6804998. For example, an untrusted applet may grant itself permissions to read and write local files or execute local applications that are accessible to the user running the untrusted applet.
- **CVSS v2 Base Score**: 10.0 (HIGH)
- **Access Vector**: Network exploitable
- **Impact Type**: Allows unauthorized disclosure of information; Allows unauthorized modification; Allows disruption of service

CVE-2009-2016: SQL Injection in Commercial e-Shop Product

- **Original release date**: 06/09/2009
- **Overview**: SQL injection vulnerability in products.php in Virtue Shopping Mall allows remote attackers to execute arbitrary SQL commands via the cid parameter.
- **CVSS v2 Base Score**: 7.5 (HIGH)
- **Access Vector**: Network exploitable
- **Impact Type**: Allows unauthorized disclosure of information; Allows unauthorized modification; Allows disruption of service

Brief Analysis of these Examples

- All of these security problems are due to software defects
  - Morris worm: vulnerabilities in Unix daemons
  - Code Red: vulnerability in Internet Information Service (IIS)
  - Zotob: vulnerability in the Windows OS
  - Poker game: vulnerability in the implementation of the PRNG
  - Windows kernel input validation: vulnerability in Windows OS graphics engine
  - JRE/JVM: vulnerability in the Java implementation
  - E-Shop product: vulnerability in the e-shop program
- All vulnerabilities can be exploited over the network
  - Local exploits are powerful, too...
  - ...but remotely exploitable vulnerabilities are usually the most critical ones
- No help of ignorant / stupid / curious users needed
  - Users can be a security problem, but here it's all related to software defects

Why do we have so many Security Issues Today?

Exposure

Today, critical services are reachable (exposed) over the Internet, which means there are many attractive targets

- Years ago, when critical applications were used "in-house", the physical security perimeters (walls) provided good protection
  - With exposed systems, this is no longer the case as attacks can be carried out by anyone from anywhere
- Many services are attractive targets
  - Possible monetary gain by attacking e-banking and payment systems
  - Several companies are very dependent on their online presence, making them potentially vulnerable to DoS
    - DoS attacks are relatively easy to carry out; has happened several times against banks, e-shops, airlines...
- Legacy applications that were originally developed for internal purposes are offered to the public due to, e.g., new business opportunities
  - Difficult to do in a secure way if the original application was not designed for this purpose

Extensibility

Systems have become extensible, which makes it much more difficult to control (and secure) the software they run

- Plug-in architecture of web browsers
  - Check-out how many extensions your browser has currently installed – do you trust them?
- Operating systems can be supplied with device drivers to support all kinds of hardware
- Service-oriented Architecture (SOA): the entire concept is based on using components from different sources
  - It's difficult to assess the trustworthiness of other components or the security-checks they perform

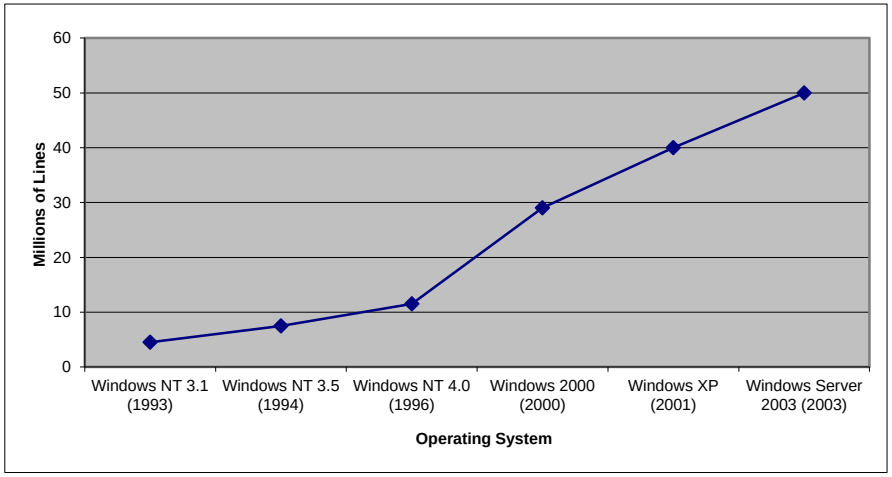
→ Trade-off between time-to-market and security

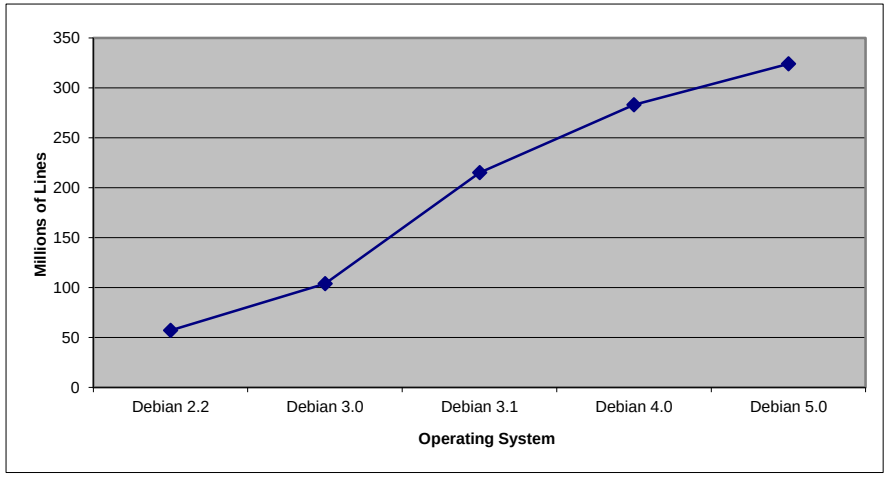
- From an economic standpoint, extensibility is good as it allows to ship a product early and provide additional functionality later
- But its very hard to prevent software vulnerabilities from "slipping in as unwanted extensions"

Complexity (1)

Software grows in size (gets more complex), which directly affects the number of security vulnerabilities (more lines → more bugs)

- Lines of codes of Windows OS and Debian Linux:





- When you run your program, often much more software than "just your code" is involved
  - Runtime environments (Java, .NET, application server...)
  - Application frameworks (JSF, Spring, Wicket...)
  - Libraries, OS (including drivers), network services (e.g. DNS)…

Complexity (2)

- The following graph shows that the number of incidents grows with the square of the code size

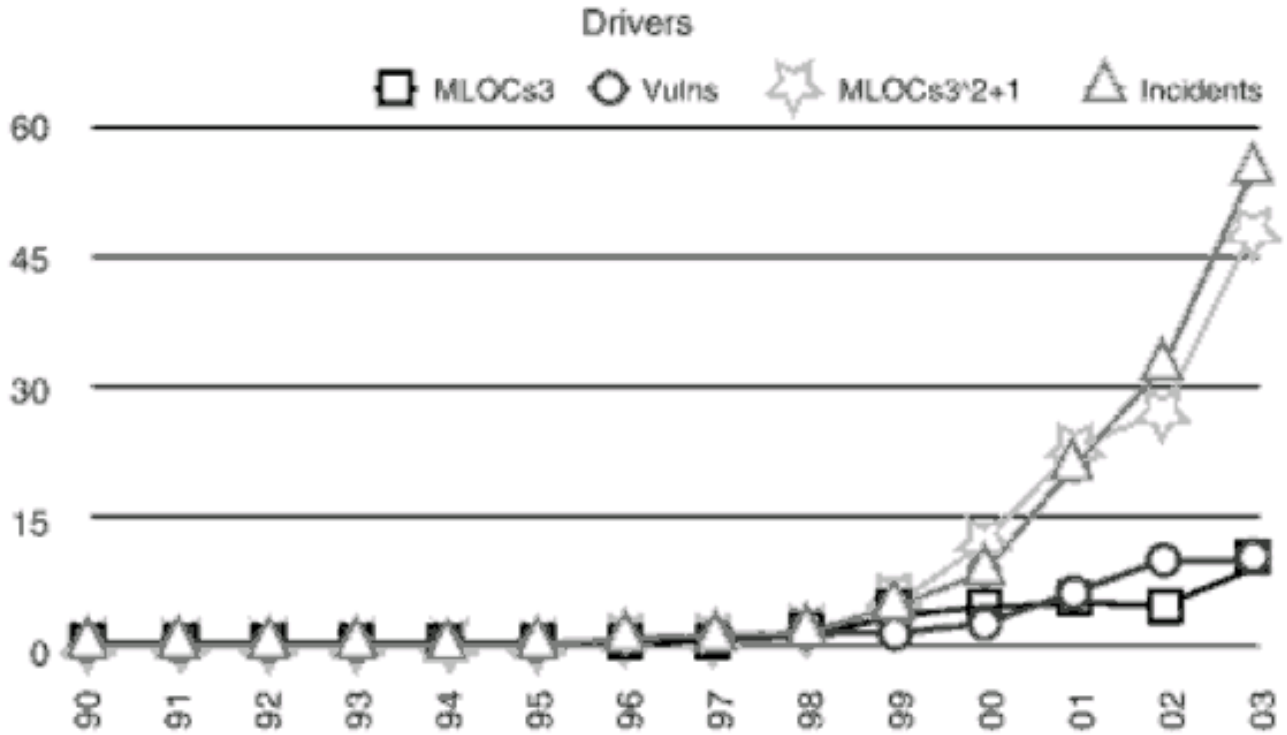


Figure 1–5 Computation of two kinds of code volume curves (MLOCs3 and MLOCs3^2+1; see text for definition) results in curves with some predictive power. (After Geer.)

more lines → more bugs → more security problems

Traditional Security Measures

Traditional Security Measures (1)

- During the past years, several security mechanisms have been developed and are widely used today

- Many of them focus on security functionality and not on preventing exploitable software defects
  - Secure communication protocols (SSL/TLS, IPsec, WLAN security etc.)
  - Authentication mechanisms, digital signatures (PKI, smartcards etc.)
  - Access control mechanisms (role-based access control...)
  - Hard drive encryption
  - ...

These are all important in their respective fields of application an should be employed where needed

Traditional Security Measures (2)

- Keep in mind that security mechanisms themselves are mostly software and using them may lead to new vulnerabilities
- E.g. OpenSSL with > 100 CVE entries
  - CVE-2014-0160, 04/07/2014, MEDIUM: OpenSSL 1.0.1 before 1.0.1g does not properly handle Heartbeat Extension packets, which allows remote attackers to obtain sensitive information (e.g., server private key) from process memory via crafted packets that trigger a buffer over-read
  - CVE-2010-3864, 11/17/2010, HIGH: Multiple race conditions in ssl/t1_lib.c in OpenSSL 0.9.8f through 0.9.8o, 1.0.0, and 1.0.0a, when multi-threading and internal caching are enabled on a TLS server, might allow remote attackers to execute arbitrary code…
  - CVE-2009-0653, 02/20/2009, HIGH: OpenSSL, probably 0.9.6, does not verify the Basic Constraints for an intermediate CA-signed certificate, which allows remote attackers to spoof the certificates of trusted sites via a man-in-the-middle attack...
  - CVE-2006-3738, 09/28/2006, HIGH: Summary: Buffer overflow in the SSL_get_shared_ciphers function in OpenSSL 0.9.7 before 0.9.7l, 0.9.8 before 0.9.8d, and earlier versions...

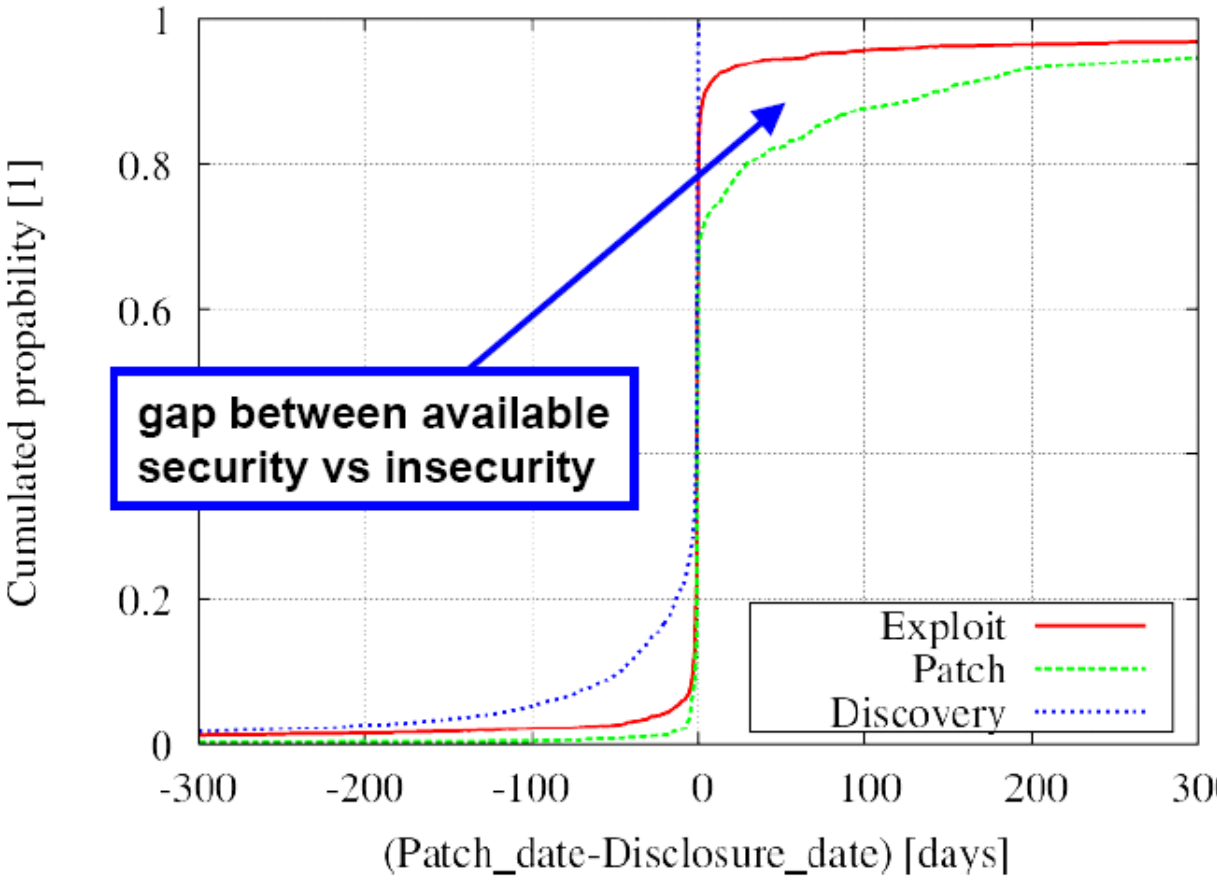
Traditional Security Measures (3)

- And what is the most popular traditional approach to protect from exploitable software defects? Penetrate and Patch!
  - From a security perspective, this means releasing poorly designed / implemented software to the commercial market
  - Patches are issued when security vulnerabilities are discovered
  - Even today, several companies mainly practice this approach
- There are several problems with this approach
  - Before a vulnerability is fixed, it must first be found
  - Who finds the vulnerability first – a cybercriminal or a "honest" user?
  - Quickly written patches may introduce new vulnerabilities
  - No guarantee users will install patches
  - Damaging to a vendor's reputation

Using such a reactive approach to software security is a sign of poor software security practice!

Traditional Security Measures (4)

- One major problem with "Penetrate and Patch" is that the exploit may be available earlier than the patch



Y-Axis:
cumulated probability for exploit- and patch-availability dates

X-Axis:
days from disclosure-date

Data:
- **3416 exploits**
- **1477 patches**
- **from 1996-2006**

Traditional Security Measures (5)

- There are many other security mechanisms
  - Firewalls, anti-malware software, intrusion detection/prevention systems (IDS/IPS), web application firewalls (WAF)...
  - ... but many of them are primarily necessary because of poorly implemented software or poorly configured systems
  - Example: Do I need a firewall, if...
    - ...I make sure all my remotely accessible software is free of defects...
    - ...and my system is configured that no non-authorized services are running?
  - Bruce Schneier: "*We wouldn't have to spend so much time, money, and effort on network security if we didn't have such bad software security*"

- Conclusion
  - There are many established security mechanisms for security functionality such as secure communication protocols and access control mechanisms
  - But with respect to preventing software vulnerabilities, the maturity level is much lower and several companies still depend on penetrate and patch

Secure Software Development (1)

So when it's so clear that secure software development would be good, why isn't it practiced more often?

- Primary goal of software is to provide some functionality, security has often not top priority
  - Within security, "obvious" security functionality (e.g. secure communication or user authentication) has usually much higher priority than prevention of software defects
- Lack of education / awareness
  - Only relatively few software engineers are truly security aware
  - Software security does not get much attention in programming courses or other security courses
  - It's important to realize that developing secure software requires additional skills compared to "just" implementing functional software

In particular, you must learn to think like an attacker because otherwise, you won't be aware of possible threats and won't design and implement appropriate countermeasures

Secure Software Development (2)

- Secure software development is an unfair battle
  - The developer has to get it right all the time (no vulnerabilities)
  - The attacker just has to find one vulnerability

  Developing secure software is hard!

- On the positive side, there are some signs that secure software development gets more attention
  - Specialized courses at the university-level are being offered
  - Several guidelines / best practices have been proposed
  - Some good software security books have been published
  - More and more companies have switched to more secure development practices, notably Microsoft (Secure Development Lifecycle (SDL))

Limits of Developing more Secure Software

- Even when putting significant focus on secure software development, the problem won't be solved 100%
  - It's unlikely (at least in the foreseeable future) that even with a very good software development process, all security defects will be spotted
  - New attack vectors appear, against which current software may be vulnerable
- This implies that many of the traditional approaches to "solve" software security will still be used
  - Security patching will still be necessary (hopefully less frequently)
  - Additional security measures will still be reasonable
    - Firewalls still make sense to get a basic separation/filtering of traffic
    - An IPS or WAF can still be useful to protect highly critical systems and applications (but operating them can be expected to get easier)
    - This approach is called "defence in depth"
- Just like with any security measurement, secure software development will significantly improve the situation, but won't give us 100% security

Terminology and Definitions

Defects: Bugs and Flaws

- Implementation Bug (or simply bug)
  - An implementation-level software problem (not necessarily security-related)
  - Can often be discovered by manual or automatic code inspection
- Design Flaw (or simply flaw)
  - A flaw is typically introduced at the design level (e.g. poorly designed random number generator)
  - While flaws are of course "visible" in the code, spotting them there is much more difficult
  - Flaws can be uncovered by performing threat modeling
- Defect
  - Both implementation bugs and design flaws are defects
- In practice, security problems are about 50/50 divided between bugs and flaws (so software security is not exclusively about coding issues!)

Bug Example

- One typical type of a bug is a buffer overflow
  - Happens when data is written beyond the end of an allocated buffer
  - A major problem in languages that do not perform automatic array bounds checking, e.g. C and C++

- This code contains a buffer overflow:

```
void main() {
  char buf[1024];
  gets(buf);
}
```

  - The `gets` function reads from stdin until a newline character is found
  - It does not check the size of the available buffer
  - If more than 1024 characters are read, the data is written beyond the allocated buffer on the stack
  - Carefully crafting the input may allow injecting code or modifying program control flow

Flaw Example

- Microsoft Bob was an add-on to Windows 3.11 and Windows 95
- It allowed the user to set a password to protect access to, e.g., personal documents
- However, Bob was so user-friendly that when the user failed to enter his/her password three times, Bob would pop up and allow the user to enter a new password – although the user had no idea of the old one

Range of Defects

There's a wide range of defects and the difficulty to spot them varies

- Simple implementation bug
  - For instance a call of an unsafe C function such as `gets`, which may easily result in a buffer overflow
  - Can easily be spotted by manual or automatic code analysis by looking at a single line of code
- "Midrange" implementation bug
  - For instance a race condition
  - More difficult (but still possible) to detect by manual or automated code inspection as it may require analyzing the behavior of several threads and their accessing of shared global variables or objects etc.
- Design-level flaws
  - For instance an insecure access control mechanism that uses tickets where a user can easily elevate his privileges by tampering with the ticket
  - Staring at code (you or a tool) won't help as a higher-level understanding is required (threat modeling may be helpful, though)

Vulnerability, Threat and Exploit

- Vulnerability
  - A vulnerability is a defect (bug or flaw) that an attacker can exploit
  - Not every bug or flaw in a program can be exploited, some of them may just result in functional problems
- Threat
  - A threat is a possible danger that may cause harm by exploiting a vulnerability
  - Can be intentional (a cracker/attacker) or accidental (a fire in the server room)
  - The attacker is often identified as the threat agent, while the actual attack is the threat action
- Exploit
  - The actual attack that takes advantage of a vulnerability
  - E.g. a piece of malware or sequence of commands/activities

Assets, Risk and Countermeasure

- Asset
  - Anything that is of value to an organization (and therefore also to a potential attacker)
  - E.g. an organization's business operations and their continuity, including information resources (e.g. customer data) that support the organization's mission

- Risk
  - Risk identifies the potential that a given threat will exploit vulnerabilities to cause harm to the organization (its assets)
  - It is measured in terms of a combination of the probability of an event and its consequence: risk = probability × impact

- Countermeasure
  - An action, device, process, or technique that reduces a risk
  - E.g. by removing a vulnerability or by reducing the harm it can cause

Summary

- Today's software typically contains defects, which results in security risks
- Today's security incidents are frequently related to poor software security
- One main reason for the fact that the problem has become worse over the years is because of the growing exposure, extensibility, and complexity of software and systems
- Traditional approaches to tackle the software security problem are often based on "penetrate and patch", which has several drawbacks
- The only reasonable way to solve the problem is to employ secure software development practices
- Bugs and flaws are defects; a defect that can be exploited is a vulnerability, which leads to risk, which can be reduced by countermeasures

Incentives for Secure Software…