Welcome to Secure Programming I

• This is an introductory course that aims to make you “security-aware”

• So far, as a engineers, you have learned to write code and build applications…
  … we now show you how to break them 😊

• Our aim is to help you to learn typical and common security mistakes by showing how to break systems
OK, but Why?

- In computer science education, you learn to design and program code, but security education falls short
  - Simple programming mistakes lead to serious security problems
  - Today, failing to protect yourself and not being security-aware can be very costly
  - The number of security-related incidents on the Internet increasing fast (e.g., look at recent Trojan attacks on banks)
  - People got falsely accused of performing illegal activities because their computers were hacked
Some Interesting Numbers

• Adware industry is worth 2 billion dollars per year
• Malware industry is worth 105 billion dollars per year

• 50%-80% of computers connected to Internet are infected with spyware
• 81% of emails is spam (symantec report feb11)
• 90% of web applications is vulnerable (cenzic report 09)
What we expect from you

• Technical interest for security issues
  (Doing security without being interested… is useless)

• Interest in understanding how things work, often from a very low-level point of view
  (If you are scared of binary codes... secprog is not for you)

• Basic programming knowledge and experience

• Lot of patience
  (security exercises aren’t like Hollywood scenes 😊)
Administrative Issues

• Mode
  – Lectures covering different practical security aspects
  – Regular security challenges (e.g., cracking web applications, using security tools, stack-based buffer overflows,...)
  – More or less one challenge every 2 weeks
    • You need to register to the course to get an account! (more info on this later)
  – Written final exam (end of June)

• Slides and News (please visit regularly!)
  – http://www.iseclab.org/secprog/
First Assignment

• We are collecting data about user awareness on important security topics

• Please register and complete the questionnaire available at: http://lupin2.iseclab.org:8131/
  – It should take you about 1h to answer to all the questions

• The questionnaire is anonymous
  – It is not important how many questions you answer correctly, just answer to the scenarios as you would do in a real situation
  – The result has no effect whatsoever on the final grade
  – It seems that there is a problem with the adblock plugin. If you installed it in your browser, disable it for the test
Registration

• Once you have completed the questionnaire, send me an email from the same address you used to register for the questionnaire

• Put [SECPROG] in the message subject

• When I will receive the mail, I will register you to the course and I will create you an account for the challenges

• Please try to send me the mail by next friday
Topics we will likely cover
(but they may slightly change along the road)

1. Web security and vulnerabilities
2. System security and vulnerabilities
3. Malware
4. Architectural principles
5. Stack-based buffer overflows
6. Software testing (i.e., finding vulnerabilities)
7. Operational practices
SecProg Lab

• Assignments
  – The first challenge will be announced next week
  – 5 challenges
  – 5 points per challenge solved

• Environment
  – assignments should be mostly solved at home / any computer with Internet connection
  – small “hacking” network, which is remotely accessible via ssh

• Submission
  – hard deadlines (with sufficient time)
  – automatic checking with immediate feedback
Grading for the lab

• Each challenge (assignment) brings you 5 points
• The written exam has 50 possible points
• Total of 75 points for the course
• You need to have a total of 38 points to pass the course

• Example: John Hacker solves 3 challenges, and gets 35 points in the written exam. John has
  \[ 3\times 5 + 35 = 50 \] points
• Hence, if you solve 5 challenges, you will get the maximum amount of points for the lab part of the course
SecProg I and SecProg II

• The current setup
  – SecProg1 in the spring
    • Short course in combination with Marc
  – SecProg2 in the fall
    • Long course
  – Similar organization
  – Large overlapping of topics and security challenges
SecProg I and SecProg II

• The new (from fall 2011) setup
  – SecProg1 in the fall
    • Long course presenting all the basis of system and network security
    • Include some of topics that were covered in SecProg2
  – SecProg2 in the spring
    • Long course
    • Focusing on advanced topics
    • Show students the current (both from a technical and a research perspective) of the fight against cybercrime
  – Almost no overlapping of topics
  – Different types of homeworks
Intro and History
One big problem

• System and network administrators are not prepared
  – Insufficient resources
  – Lack of training

• Intruders are now leveraging the availability of broadband connections
  – Many connected home computers are vulnerable
  – Collections of compromised home computers are “good“ weapons (e.g., for distributed denial of service attacks).
Number of Reported Incidents

* www.cert.org
Bugs And Failure

- Hardware and software are developed by humans and therefore are not perfect
- A human error may introduce a **bug** (or fault)
- When a fault gets triggered, it might generate a **failure**...

- If the fault is “security-related”, it is usually called a **vulnerability**
- When the vulnerability is triggered (exploited) can lead to the **compromise** of the system (or of part of it)
Vulnerabilities

A little bit of history…

- **1960s** - mainframe computers like the MIT’s Artificial Intelligence Lab became staging ground for hackers. *Hacker was a positive term*

- **1970s** - hackers start tampering with phones (the largest network back then)
  - **1972**, John Draper finds that the whistle that comes with the Cap’n Crunch cereal produces a sound at the 2600 Hz (the same used by AT&T to authorize long-distance calls)
  - It is the start of phone phreaking
A little bit of history…

- **1973** - Bob Metcalfe wrote RFC 602: “The Stockings Were Hung by the Chimney with Care”
  - ARPA computer network is susceptible to security violations
  - “many people still use passwords which are easy to guess: their first names, their initials, their host name spelled backwards, a string of characters which are easy to type in sequence”
- **1980/81** - Two hacker groups form
  - Legion of Doom (US)
  - Chaos Computer Club (DE)
- **1982** - The term “cyberspace” is coined in the novel *Bournig Chrome*
A little bit of history…

• 1983 - The movie Wargames introduces hackers to the public
• 1986 - German hackers penetrate Lawrence Berkeley Laboratory systems and try to obtain secrets to be sold to the KGB
  – Cliff Stoll (a sysadmin at LBL) found an intruder while investigating a 75 cent accounting discrepancy for CPU time
  – He decided to monitor the intruder in order to find out who he/she was and how he was able to gain privileged access
  – The investigation ends with the arrest of Markus Hess in Germany, who apparently worked for the Eastern Bloc
A little bit of history…

• **1988** - The *Internet worm*, developed by Robert T. Morris, brings down the Internet
  – A mistake in the replication procedure led to unexpected proliferation
  – The Internet had to be “turned off”
  – Damages were estimated in the order of several hundred thousand dollars
  – The **CERT** (Computer Emergency Response Team) is formed

• **1994** - Kevin Mitnick attacks the Supercomputer Center in San Diego using a TCP spoofing attack
  – Arrested in 1995 and sentenced to 46 months in prison
A little bit of history…

- **1990** - Operation Sundevil: secret service arrests hackers in 14 U.S. Cities for credit-card theft and telephone and wire fraud
- **1992** - Release of the movie *Sneakers*
- **1993** – The first *DefCon* conference is held in Las Vegas. It is so popular that it will become an annual event
- **1995** – A russian cracker siphon 10M $ from Citibank and transfer the money to banks around the world
- **1995** – The movie *Hackers* is released
- **1999** – The *melissa* worm causes large problems to the email systems
A little bit of history…

• **2000** – ILOVEYOU, a VBScript worm infects millions of computers within a few hours of its release

• **2002** - Bill Gates announces the 'Trustworthy Computing' initiative, a new direction in Microsoft's software development strategy aimed at increasing security

• **2003** – The SQL Slammer worm infected 75,000 machines (90% of the possible targets) in 10 minutes
  
  – Starts the fear for the **flash worms**

• **2005-today** – Worms are slowly replaced by botnets

• **2010** – Stuxnet attacks centrifuge systems in nuclear facilities in Iran
  
  – Completely new (and unexpected) level of sophistication
Changing Nature of the Threat

- Nowadays Intruders are more prepared and organized
- Internet attacks are easy, low-risk and difficult to trace
- Intruder tools are increasingly sophisticated and easy to use (e.g., by Script kiddies)
- Source code is not required to find vulnerabilities
- The complexity of Internet-related applications and protocols are increasing – and so is our dependency on them
Online Crime is a Business Now

Klikparty, 2007
The term “hacker” was introduced at MIT in the 60s to describe “computer wizards” someone who lives and breathes computers, who knows all about computers, who can get a computer to do anything. Equally important, though, is the hacker's attitude. Computer programming must be a hobby, something done for fun, not out of a sense of duty or for the money.

- Brian Harvey, University of Berkeley

The term was later associated to “malicious hackers” or “crackers”, that is, people that perform intrusions and misuse computer systems
Terminology

• **Black Hat**: a cracker, someone bent on breaking into the system you are protecting

• **White hat**: usually associated to friendly security specialists

• **Script Kiddie**: lowest form of cracker; script kiddies do mischief with scripts and programs written by others, often without understanding the exploit they are using
Terminology

• What is an attack?
  – no easy answer, it depends

• Security Policy
  – The framework within which an organization establishes needed levels of information security to achieve the desired **integrity, confidentiality, and availability goals**. A policy is a statement of **information values**, protection responsibilities, and organization commitment for a system.
    (US Congressional Office of Technology)

  – A set of guidelines defining **what you want to protect** and what you want to allow at your site.
Terminology

- **What you want to protect?**
  - defines assets

- **What are the goals of your protection efforts?**
  - **Integrity**
    - Data has not been altered or destroyed in an unauthorized manner
  - **Confidentiality**
    - Information is not made available or disclosed to unauthorized individuals, entities or processes
  - **Availability**
    - Data/Service being accessible and usable upon demand by an authorized entity
Terminology

• What do you want to protect against?
  – threat model
  – risk analysis

• Different security policies
  – bank answers questions different than home user

• Attack
  – any maliciously intended act against a system or a population of systems
  – any action that violates a given security policy
<table>
<thead>
<tr>
<th>Threat</th>
<th>Vulnerability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thieves could break into our facility and steal our equipment</td>
<td>The lock we are using on the building doors is easy to pick or bump</td>
</tr>
<tr>
<td>Adversaries might install malware in the computer so they can steal social security numbers for purposes of identity theft.</td>
<td>The computer do not have up to date virus signatures</td>
</tr>
</tbody>
</table>
Malicious hacking/cracking is illegal

However, discussing vulnerabilities and how they are actually exploited is useful to educate and increase awareness

A full disclosure policy has been advocated by many respected researchers, provided that...

- The information disclosed has been already distributed to the parties that may provide a solution to the problem (e.g., vendors)
  - See: Responsible vulnerability disclosure process (IETF Internet Draft)
- The ultimate goal is to prevent similar mistakes from being repeated
Security Overview
Security Threats

Information Domain

• Leakage
  – acquisition of information by unauthorized recipients. e.g. Password sniffing

• Tampering:
  – unauthorized alteration/creation of information (including programs)
  – e.g. change of electronic money order, installation of a rootkit
Security Threats

Operation Domain

• Resource stealing
  – (ab)use of facilities without authorization

• Vandalism
  – interference with proper operation of a system without gain
Security Overview

• Security issues at various stages of application life-cycle
  – mistakes, vulnerabilities, and exploits
  – avoidance, detection, and defense

• Architecture
  – security considerations when designing the application

• Implementation
  – security considerations when writing the application

• Operation
  – security considerations when the application is in production
Security Overview

**Architecture and design**
- validation of requirements (building the right model)
- verification of design (building the model right)

**Common problems**
- authentication and privileges
  - session reply
  - principle of least privilege
- communication protocol design
  - sniffing, man-in-the-middle
  - session killing, hijacking
- parallelism and resource access
  - race conditions
- denial of service
Security Overview

Implementation
- verification of implementation
- classic vulnerabilities (often programming-language-specific)

Common problems
- buffer overflows
  - static (stack) overflows
  - dynamic (heap) overflows
- input validation
  - URL encoding
  - document root escape
  - SQL injection
- back doors
Security Overview

**Operation**
- decisions made after software is deployed
- often not under developer’s control

**Common problems**
- denial of service (DOS)
  - network DOS
  - distributed DOS, zombies
- administration problems
  - weak passwords
  - password cracking
  - unsafe defaults
Insecure Software

….or, why good people write bad code

• **Technical factors**
  – complexity of task

• **Economic factors**
  – deadlines
  – insufficient funding

• **Human factors**
  – mental models
  – social factors
Technical Factors

• Complexity
  – algorithmic complexity
  – parallel processes, threads
  – multi-user
  – Indeterminism

• Composition
  – incorrect assumptions
  – surprising interactions

• Changes
  – consequences are hard to predict
  – example: Sun tarballs
Economic Factors

• Production pressure
  – not enough time
  – not enough manpower for testing

• Security is not a feature
  – just secure enough

• Open-source vs. closed-source debate
  – open-source is peer-reviewed
  – closed-source is written by professionals

• Legacy software
Human Factors

• Poor risk assessment
  – invisible enemy

• Mental models
  – only check for errors that are understood
  – assume software is used for a specific task
    example: mouse driver exploit
Improvement

• Tools
  – detect mistakes and vulnerabilities
  – support programmer
  – formal verification

• Standards and metrics
  – hold vendors accountable
  – allow for comparison between products

• Education
  – that’s what we are trying to do here ;-)}
Methods of attacking

• Eavesdropping
  – getting copies of information without authorization

• Masquerading
  – sending messages with other’s identity

• Message tampering
  – change content of message
Methods of attacking

• Replaying
  – store a message and send it again later, e.g. resend a payment message

• Exploiting
  – using bugs in software to get access to a host

• Combinations
  – Man in the middle attack
    • emulate communication of both attacked partners (e.g., cause havoc and confusion)
Social Engineering

• “The art and science of getting someone to comply to your wishes”
  – Remember the film “Sneakers”? Security is all about trust. Unfortunately, the weakest link, the user, is often the target

• Performed in many different forms
  – Social engineering by phone
  – Dumpster Diving
  – Reverse social engineering

• According to report, secret services often use social engineering techniques for intrusion
Choosing a good password

• Retina checks are currently not possible, so guard your password ;-)  
  – NEVER give your password to anyone  
  – Make your password something you can remember  
  – Make your password difficult for others to guess

• Easy to break passwords: 
  – Words in any dictionary, Your user name, Your name, Names of people you know, substituting some characters (a 0 for an o, or a 1 for an l), words from book or movies, sequences of keys on a keyboard (QWERTY), all the previous + some numbers (davide223) 
  – http://www.openwall.com/john/ (John the ripper, passwd cracker)
Choosing a good password

• Guidelines…
  – at least 8 characters long
  – mix of lower- and upper-case characters, numbers, and punctuation marks
  – take a phrase and try to squeeze it into eight characters (e.g., this is an interesting lecture == *tiail*), Throw in a capital letter and a punctuation mark or a number or two (== 0*Tiail4*)
  – Something that no one but you would ever think of. The best password is one that is totally random to anyone else except you. It is difficult to tell you how to come up with these, but people are able to do it. Use your imagination!
Password Examples

• The “Bad”
  – acmilan1
  – mymusic2
  – bermuda6
  – konrad4868

• The “Good”
  – #bdiBuM1a
  – Qa56Fge(/
  – sdFOiKqw”= 
Design and Architectural Principles
Overview

• Security issues at various stages of application life-cycle
  – mistakes, vulnerabilities, and exploits
  – avoidance, detection, and defense

• Architecture
  – security considerations when designing the application

• Implementation
  – security considerations when writing the application

• Operation
  – security considerations when the application is in use
• Software architecture
  *A representation of an engineered software system, and the process and discipline for effectively implementing the design(s) for such a system*

• Representation
  – architecture concerned with components and their relationships

• Process
  – steps are provided that describe how to change design within set of constraints

• Discipline
  – set of principles how to design system within constraints
Architecture – A Quick Recap

• Software architecture has emerged as crucial part of design process
  – much work was done in the early 90s
  – today, there are research issues such as product family architectures, architectural description languages, flexibility, fault tolerance, etc.

• Software architecture encompasses the structures of large software systems
  – architectural view is abstracted
  – mostly concerned with interface descriptions (behavior)
  – distills details of implementation (such as algorithmic aspects and data representation)
What is a security architecture?

A *body* of high-level *design principles* and decisions that allow a programmer to say "Yes" with confidence and "No" with certainty.

A *framework* for *secure design*, which embodies the four classic stages of information security: *protect*, *deter*, *detect*, and *react*.

Security is a measure of the architecture’s ability to resist unauthorized usage

- at the same time, services need to be provided to legitimate users
What happens if architecture is flawed?

• Some history: The Swedish warship Vasa
  – now in Stockholm, Vasa Museum
  – a solemn reminder for engineers
  – the ship was built well, but its architecture was flawed.
  – on its first voyage, it fired its guns to salute the port and…

• So what does Vasa have to do with security?
  – your code might be engineered well, but if your architecture is bad from a security point of view, your system may be broken by attacker
Vasa Today
Architecture is Important

Cost of fixing security flaws during different development phases

- Design: (cost = 1)
- Implementation: (cost = 6.5)
- Testing: (cost = 15)
- Post-Release: (cost = 60)
Security and Design

• Systems are often designed without security in mind
  – application programmer is often more worried about solving the problem than protecting the system
  – often, security is ignored because either the policy is generally not available, or it is easier to ignore security issues

• Organizations and individuals want their technology to survive attacks, failures and accidents
  – critical systems need to be survivable
Design Principles

• Design is a complex, creative process
  – But general rules derived from experience

• 8 principles according to Saltzer and Schroeder (1975)
  “The protection of information of computer systems”
  – Economy of Mechanism
  – Fail-safe defaults
  – Complete mediation
  – Open design
  – Separation of privilege
  – Least privilege
  – Least common mechanism
  – Psychological acceptability
Economy of Mechanism

- Design should be as simple as possible
  - KISS -- keep it simple, stupid
  - Brian W. Kernighan
    “Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.”

- When things are complex, users get them wrong
Fail-safe Defaults

• **Allow** as default action
  – grant access when not explicitly forbidden
  – in case of mistake, access allowed (often not noticed)
  – improves ease-of-use
  – wrong psychological model

• **Deny** as default action
  – grant access only on explicit permission
  – in case of mistake, access denied (noticed quickly)
  – improves security
  – important for firewall configurations and input validation tasks
Fail-safe Defaults

• Configuration
  – secure initial configuration
  – easy (re)configuration

• Secure initial configuration
  – no default passwords
  – no test users
  – files are write-protected, owned by root/admin

• Error messages
  – should be very generic
  – additional information in log files
Complete Mediation

• Complete access control
  – check every access to every object
  – include all aspects (normal operation, initialization, maintenance, ..)
  – caching of checks is dangerous
  – identification of source of action (authentication) is crucial

• Trusted path
  – make sure that user is talking to authentication program
  – important for safe login (thwart fake logins)
  – Windows “control-alt-delete” sequence
Complete Mediation

• Secure interface
  – minimal
  – narrow
  – non-bypassable (e.g., check at server, not client)

• Input validation

• Trust input only from trustworthy channels
  – any value that can be influenced by user cannot be trusted
    • do not authenticate based on IP source addresses / ports
    • E-mail sender can be forged
    • hidden fields or client side checks are inappropriate
  – safely load initialization (configuration)
Open Design

• Design must not be secret
  – security mechanisms must be known
  – allows review
  – establishes trust
  – unrealistic to keep mechanism secret in widely distributed systems

• Security depends on secrecy of few, small tokens
  – keys
  – passwords
Open Design

- **Kerckhoff's principle** for cryptography:
  
  “A cryptosystem should be secure even if everything about the system, except the key, is public knowledge”

- Don't rely on secrecy does not mean make everything public

- Companies often keep secret the details of a system
  - **Security through Obscurity**
  - May improve security in the short term, but it is generally a bad idea on the long run
Separation of Privilege

• Access depends on more than one condition
  – for example, two keys are required to access a resource
  – two privileges can be (physically) distributed
  – more robust and flexible

• Classic examples
  – launch of nuclear weapons requires two people
  – bank safe

• Related principle
  – compartmentalization
Separation of Privilege

• **Compartmentalization**
  – break system in different, isolated parts and minimize privileges in each part
  – don’t implement all-or-nothing model
  \[\rightarrow\] minimizes possible damage

• **Sandbox**
  – traditional compartmentalization technique
  – examples
    • Java sandbox (bytecode verifier, class loader, security manager)
    • virtual machines
    • Rendering in google Chrome
    • System jails (chroot)
Least Privilege

• Operate with least number of rights to complete task
  – minimize damage
  – minimize interactions between privileged programs
    • reduce unintentional, unwanted use

• Minimize granted privileges
  – avoid *setuid* root programs (UNIX/Linux)
    • use groups and setgid (e.g., group *games* for high scores)
    • use special user (e.g., *nobody* for web server)
  – make file owner different from setuid user
    • taking control of process does not allow to modify program images
Least Privilege

• Minimize granted privileges
  – database restrictions
    • limit access to needed tables
    • use stored procedures

• Minimize time that privilege can be used
  – drop privileges as soon as possible
  – make sure to clear saved ID values

• Minimize time that privilege is active
  – temporarily drop privileges
  – can often be re-enabled by the attacker, but still protects against some kinds of attacks (e.g., file access)
Least Privilege

• Minimize modules that are granted privilege
  – optimally, only single module uses privileges and drops them
  – two separate programs
    • one can be large and untrusted
    • other is small and can perform critical operations
    • important for GUI applications that require privileges

• Limit view of system
  – limit file system view by setting new root directory
    chroot() – on Unix
  – more complete virtual machine abstraction
    BSD system call jail(2)
  – Honeypot
Least Privilege

• Do not use `setuid` scripts
  – “race condition” problems
  – Linux drops `setuid` settings

• Minimize accessible data
  – CGI scripts
    • place data used by script outside document root

• Minimize available resources
  – quotas

• Paper: Provos et al., *Preventing Privilege Escalation*,
  12th USENIX Security Symposium, 2003
Least Common Mechanisms

• Minimize shared mechanisms
  – reduce potentially dangerous information flow
  – reduce possible interactions

• Problems
  – beware of “race conditions”
  – avoid temporary files in global directories
Psychological Acceptability

• Easy-to-use human interface
  – easy to apply security mechanisms routinely
  – easy to apply security mechanisms correctly
  – interface has to support mental model
    • do what is expected intuitively (e.g., personal firewalls)

• Authentication
  – passwords
    • enforce minimum length (what is the minimum length?)
    • enforce frequent changes
  – PKI (public key infrastructure)
    • overhead vs. security
One more Design Principle

• Separate data and control
  – failed separation is reason for many security vulnerabilities
    • from buffer overflows to macro viruses
  – distinction between control information and data has to be clear

• Problematic
  – with automatically executing code in data files
    • JavaScript in web pages
    • automatic preview of web pages in emails
    • macros in Word
  – when using mobile code
    • code that is downloaded and executed locally
Practice Defense in Depth

- Have several layers of security
  - Preventing is not enough, you also need detection and mitigation mechanisms
  - Two controls are better than one

- No single point of failure
“The only system which is truly secure is one which is switched off and unplugged, locked in a titanium-lined safe, buried in a concrete bunker, and surrounded by nerve gas and very highly paid armed guards. Even then, I wouldn't stake my life on it”

-- Gene Spafford
Minimize Attack Surface

• Minimize
  – number of open sockets
  – number of services
  – number of services running by default
  – number of services running with high privileges
  – number of dynamic content webpages
  – number of accounts with administrator rights
  – number of files & directories with weak access control

• Minimize the “time” surface
  – Automatically lock screen after n minutes
  – it’s good practice to zero-out memory that contains sensitive information (usually, decrypted information) as soon as it’s no longer needed
Retrofitting Applications

• Applying security techniques to existing applications
  – element of overall system design
  – when no source code available or
  – complete redesign too complicated

• Wrappers
  – move original application to new location and replace it with small program or script that
    • checks (and perhaps sanitizes) command-line parameters,
    • prepares a restricted runtime, and
    • invokes the target application from its new location
  – can provide logging
  – can provide possibility for prologue and epilogue code
Retrofitting Applications

• **Example wrappers**
  - AusCERT Overflow Wrapper
    - exits when any command line argument exceeds a certain length
  - TCP Wrappers
    - replaces inetd (for telnet, ftp, finger, …)
    - access control
    - logging
  - **sendmail restricted shell (smrsh, replacement for /bin/sh)**
    - sendmail known for security problems
    - smrsh restricts accessible binaries
    - interestingly, was vulnerable to two exploits that allow arbitrary code execution
Retrofitting Applications

• **Interposition**
  - insert program that we control between two pieces of software that we do not control
  - filtering of data
    • add security checks and constraints
  - network proxy
    • application policy enforcement
    • SYN flood protection
  - input sanitization
Bad Practice

• Being too specific too soon
  – without having a design, solve technical problems and start implementation

• Focus only on functionality
  – security must be built in from the beginning

• Not considering economic factors
  – ignoring the cost of security features
Bad Practice

• Not considering the human factor
  – propose solutions that users strongly dislike
    • biometric scanners instead of passwords

  – propose solutions that are annoying
    • change passwords to frequently
    • terminate idle sessions too fast

  – propose solutions that require considerable additional effort
    • producing too many alerts (e.g., snort -- “useless”)
    • require checking of many different log-files
Conclusion

• We looked at introductory topics
  – Social engineering, passwords, importance of security

• We discussed architectural considerations and issues

• To register for the course and get an account, send me an e-mail (davide@iseclab.org) after you completed the questionnaire
  – Put [SECPROG] in the message header !!!