Software & Cybersecurity Notes

World before computer:

Signing, legalizing a paper would authenticate it

PhotoCopying easily detected

Erasing, inserting, modifying words on a paper document easily

Electronic world:

the ability to copy and alter information has changed dramatically

• No difference between an "original" file and copies of it

Possible adversaries

Student: Businessman: Ex-employee: Accountant, Stockbroker, Convict,

Spy;Terrorist: Main adversaries.

NOTE: making a network or a communication secure involves

1)Keeping it free of programming errors

2) intelligent, dedicated and often well-funded adversaries

Information Security:

Types: Application Security, Internet Security, Cloud security, Cryptography, etc.

Classes of network security problems

Confidentiality (or secrecy)

— Keep the information out of the hands of unauthorized users, even if it has to travel over insecure links

— Privacy defines the ability to secure personally identifiable data

Authentication

— Determine whom you are talking to before revealing sensitive information

Data integrity (or message authentication)

— Make sure that the message received was exactly the message you sent (not necessarily interested here in the confidentiality of the document)

Non-repudiation (or signatures)

— the assurance that someone cannot deny the validity of something

Cryptography:

The study of mathematical techniques related to aspects of information security [confidentiality, data integrity, authentication, and non-repudiation.]

Crypto-secret Graphy-Write

Cryptology = Cryptography + Cryptanalysis

— Cryptography --- code designing

— Cryptanalysis --code breaking

Cryptanalysis — types of attacks

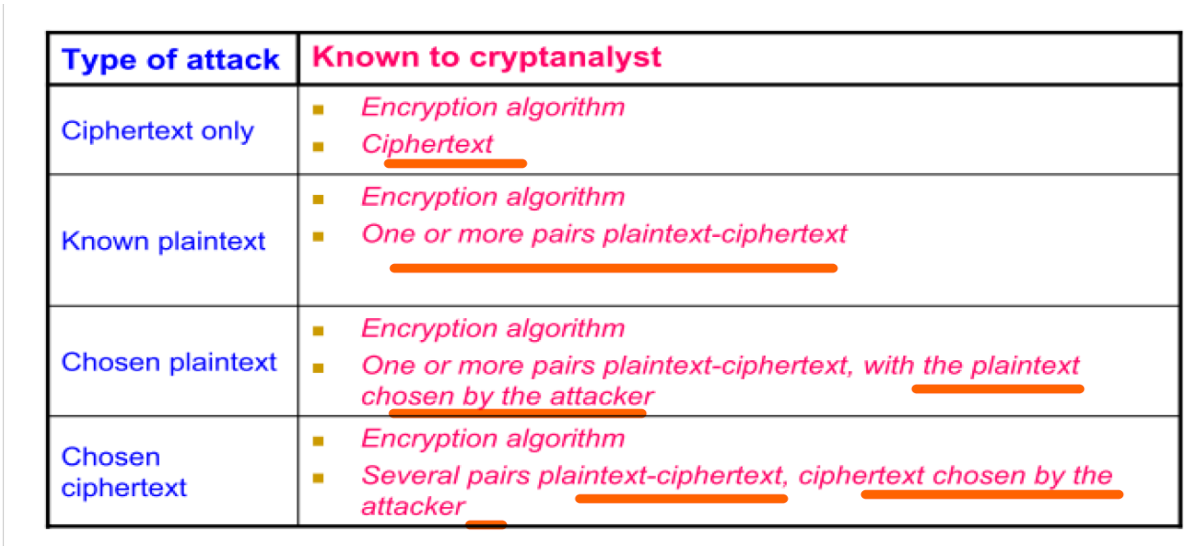
Fundamental rule: assumption: attacker knows the methods for encryption and decryption; target are **the keys.**

Passive attack: monitors the traffic attacking the confidentiality of the data

Active attack: alter the transmission attacking data integrity, confidentiality, and authentication.

Cryptanalysis: rely on the details of the encryption algo.

Brute-force attack: try every possible key on the ciphertext until an intelligible translation into a plaintext is obtained



Type of attack	Known to cryptanalyst
Ciphertext only	▪ *Encryption algorithm* ▪ *Ciphertext*
Known plaintext	▪ *Encryption algorithm* ▪ *One or more pairs plaintext-ciphertext*
Chosen plaintext	▪ *Encryption algorithm* ▪ *One or more pairs plaintext-ciphertext, with the plaintext chosen by the attacker*
Chosen ciphertext	▪ *Encryption algorithm* ▪ *Several pairs plaintext-ciphertext, ciphertext chosen by the attacker*

For all Types, Encryption algo is common;

Security Evaluation

Unconditionally secure system

— CT does not contain enough information to determine uniquely the corresponding plaintext:

---any PT may be mapped into CT with a suitable key

—the **attacker cannot find the plaintext regardless of how much time** and computational power he has because the information is not there!

—Available in: **one-time pad**

One time pad: use a (truly) random key as long as the plaintext — change the key for every plaintext

Unbreakable: the ciphertext bears no statistical relationship to the plaintext

The security is entirely given by the randomness of the key (making random key is large task)

Key distribution enormously difficult

Conditional or Complexity-theoretic security

— Consider the **weakest possible assumptions** and the **strongest possible**

attacker and **do worst-case or at least average-case analysis**

Provable security

Prove that breaking the system is equivalent with solving a supposedly difficult (math) problem (e.g., from Number Theory)

Computationally secure

— The **cost of breaking the system** exceeds the **value of the encrypted information**

— The time required to break the system exceeds the useful lifetime of the information

Symmetric Key Algorithms

- Historic ciphers — Caesar, shift, mono alphabetic, Playfair, Hill, Autokey, polyalphabetic, Rail fence, Affine
- Stream Ciphers and Block Ciphers
- DES, Double DES, Triple DES, AES
- RC4, RC6, RSA, Deffie-Hellman, ECC
- Hash functions

SOFTWARE SECURITY

Software is a MAJOR source of security problems and plays MAJOR role in providing security

Important game changers: (**ransomware, bitcoin**)

Zerodium

A famous premium bounties reward company

American information security company, expert in zero-day exploits

zero-day exploit

a software vulnerability discovered by attackers before the vendor has become aware of it.

Remote Code Execution (RCE) and Local Privilege Escalation (LPE):

attacker to discover and exploit internal vulnerabilities of server gaining access to connected systems

Virtual machine escape

an exploit in which the attacker runs code on a VM that allows an operating system running within it to break out and interact directly with the hypervisor.

Hypervisor

software that you can use to run multiple VMS on a single physical machine

Sandbox

isolated testing environment that enables users to run programs or open files without affecting the application

Sandbox escape

act of breaking out of a secure or quarantined environment, often called a sandbox.

security software != software security

Security is emergent property of entire system

AV (AntiVirus), WAF (Web Application Firewall)

NIDS (Network Intrusion Detection System)

EDR (Endpoint Detection and Response)

RASP (Runtime Application Self-Protection)

CVE (Common Vulnerability Enumeration): a list of publicly disclosed info. security vulnerabilities and exposures.

Types of software security Problems

Weaknesses vs vulnerabilities

(potential) security weaknesses (an application error or bug**)**

Things that could go wrong & could be better

(real) security vulnerabilities (Weakness may escalate to a vulnerability)

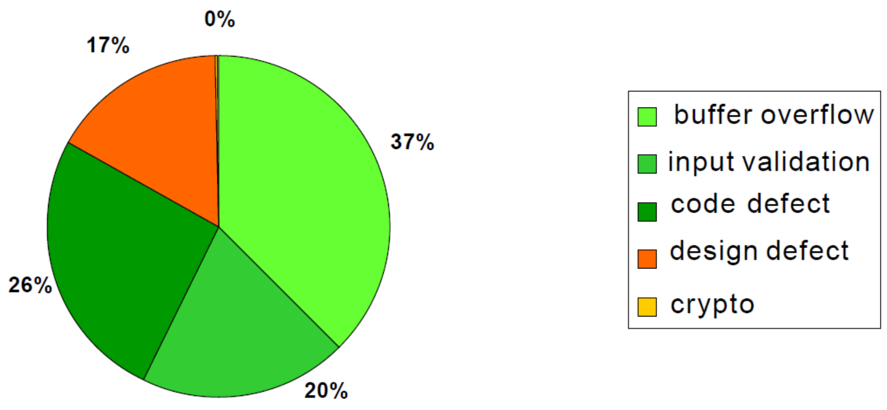
Requires flaw to be

accessible: attacker has to be able to get at it

exploitable: attacker has to be able to do some damage with it

Eg: by turning off Wifi and BlueTooth on my laptop, many vulnerabilities

become weaknesses



As a rule of thumb, coding & design flaws equally common

SQL Slammer is a 2003 computer worm (exploited a MS SQL Server 2000 vulnerability that caused DoS , slowed internet service and crashed routers round the world.)

Design flaws: introduced before coding

Implementation flaws aka bugs aka code level defects: introduced during coding

Types of implemention flaws

2a. Flaws that can be understood by looking at program itself: **TYPOS**

2b. Problems in the interaction with the underlying platform or other systems and services,

memory corruption in C(++) code

SQL injection in program that uses an SQL database

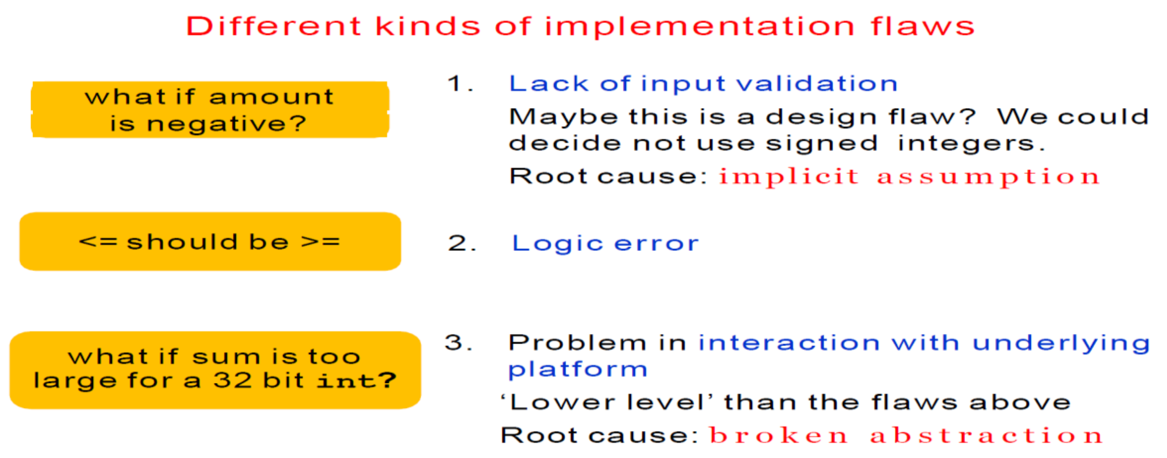
XSS, CSRF, SSI, XXE: in web-applications

Deserialisation attacks in many programming languages

Attacks can not only exploit bugs, but also features

Eg: SQL injection uses a feature of the back-end database

A bug is an unexpected problem with software or hardware.



Different kinds of implementation flaws

what if amount is negative?

1. Lack of input validation
Maybe this is a design flaw? We could decide not use signed integers.
Root cause: implicit assumption

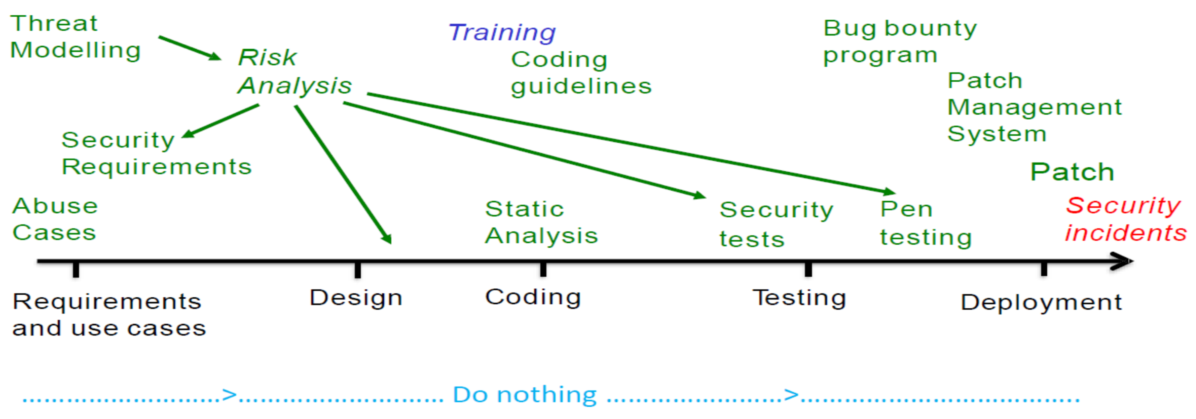
<= should be >=

2. Logic error

what if sum is too large for a 32 bit int?

3. Problem in interaction with underlying platform
'Lower level' than the flaws above
Root cause: broken abstraction

Security in Software Development Lifecycle



pen test: is an authorized simulated attack
abuse cases describe how users **misuse** or exploit the weaknesses

13

DAST: Dynamic Application Security Testing

— testing

SAST: Static Application Security Testing

---static analysis

IAST: Interactive Application Security Testing

---manual pen-testing

RASP: Run-time Application Security Protection

---monitoring

Methodologies for secure software development

Microsoft SDL: Secure DevOps (DevSecOps)[4 maturity level:basic, standardized,advanced,dynamic)

BSIMM (Building Security In Maturity Model)

- Governance
- Intelligence
- SSDL Touchpoints
- Deployment

Open SAMM (Software Assurance Maturity Model)[4 function,12 security]

- Governance
- Contruction
- Verification
- Deployment

Gary McGraw's Touchpoints

- Requirement
- Design
- Test Plans
- Code
- Test Results
- Field Feedbacks

OWASP SAMM (Open Worldwide Application Sec. Proj.)

Grip op SSD (Secure Software Development)

Prevention seems to be the way to ensure security, but detection & response often more important and effective

Threat Modelling

Negative model: prevention, generic default requirement

Positive Model: Not only prevention but also detection and reaction/response

Penetration Testing

- evaluates the strengths of all security controls on the computer system.

-evaluate procedural and operational controls as well as technological controls.

External vs. Internal

Penetration Testing can be performed from the viewpoint of an external attacker or a malicious employee.

Overt vs. Covert

Penetration Testing can be performed with (Overt) or without (Covert) the knowledge of the IT department of the company being tested.

Phases of Penetration Testing

1. **Reconnaissance and Information Gathering**

    Discover as much info about the target website.

    Nslookup, WHOIS, google search

2. **Network Enumeration and Scanning**

    Discover existing networks owned by target; live hosts

    DNS query, Route analysis(tracert)

    Nmap

    Open port: not blocked by firewall; active

    Closed port: blocked by firewall; inactive

    Filtered: something blocking the connection

3. Vulnerability Testing and Exploitation

    Check hosts for known vulnerabilities (Nessus, OpenVAS)

4. Reporting

Organize and document info found during previous 3 steps.

Tool: Dradis

Memory corruption

70% of high severity & critical security bugs are memory unsafety problems

char buffer [4];

buffer [4] ='a';

If we are lucky: program crashes with **SEGMENTATION FAULT**

If we are unlucky: program does not crash but silently allows **data corruption** or **remote code execution (RCE)**

For prioritizing efficiency over security **buffer overflow** is the No.1 security problem.

Errors with pointers and with dynamic memory (aka the heap)

C and C++ do not offer memory-safety

Causes of memory corruption problems

overread or overwrite

overreads are not a corruption issue, but **confidentiality issue**

Pointer trouble:

- buggy pointer arithmetic,
- dereferencing null pointer,
- using a dangling pointer aka stale pointer: caused by e.g use-after-free

Memory management problems:

Forgetting to check for failures in allocation

Forgetting to de-allocate, aka **memory leaks** (not corruption, but an **availability issue**)

Other ways to break memory abstractions:

missing null terminators, too many null terminators, type casts, type confusion.

Stack Overflow Attack:

Two types of attacks in these examples

- **Code Injection Attack**

  attackers inject their **own shell code** in **some buffer** and corrupt **return address** to point to this code.

  In the example, exec (' /bin/sh'); This is the classic buffer overflow attack

- **Code Reuse Attack**

  attackers **corrupt return address** to point **to existing code**

  In the example, format hard disk(the the code snippet)

Never use gets

It has been removed from the C library so this code will no longer compile

fgets (buf, size, file) instead

fgets() function reads one less than the size value

Keeping track of the space left in buffers when **using strncpy is error-prone**.

Strncpy (dest, src, size)

Better alternatives:

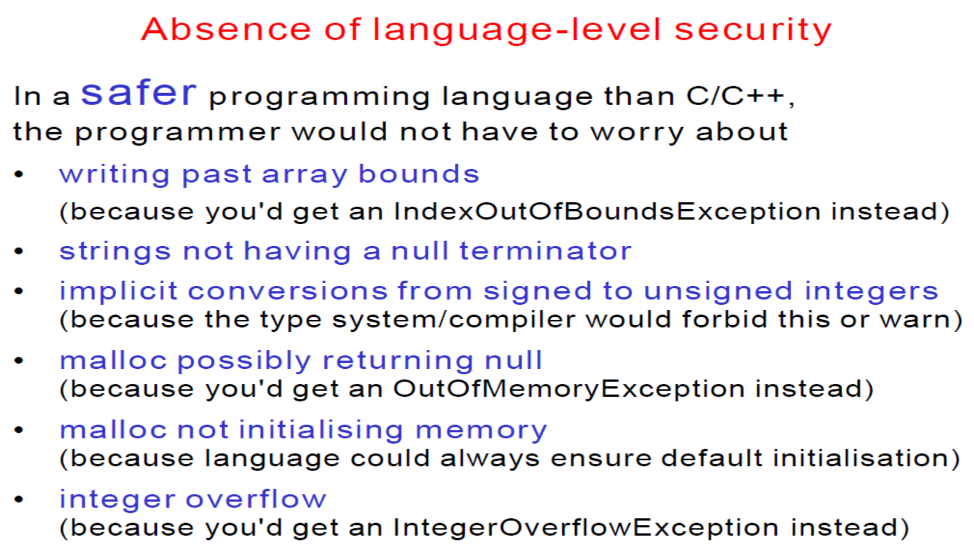
• **strlcpy (dst, src, size) and strlcat (dst, src , size)**

Here size: destination array dst, not max len. Used in **OpenBSD**.

Functions in Microsoft's **Strsafe.h** also always takes destination size as argument. Moreover

they **guarantee null-termination**.

The integer overflow is the **root problem**, the (heap) **buffer overflow** it causes **makes it exploitable**



Absence of language-level security

In a safer programming language than C/C++, the programmer would not have to worry about

- writing past array bounds
  (because you'd get an IndexOutOfBoundsException instead)
- strings not having a null terminator
- implicit conversions from signed to unsigned integers
  (because the type system/compiler would forbid this or warn)
- malloc possibly returning null
  (because you'd get an OutOfMemoryException instead)
- malloc not initialising memory
  (because language could always ensure default initialisation)
- integer overflow
  (because you'd get an IntegerOverflowException instead)

Switch from ASCI to UNICODE caused lots of buffer overflows

C++ code uses late binding to resolve (so-called virtual) method calls

Format string attacks

calling the program with strings containing special characters can result in a buffer overflow attack.

SUT

To test a SUT (System Under Test) we need two things

1. **test suite**: collection of input data

2. **test oracle:** to decide if response is ok or reveals an error (just looking if application doesn't crash)

Moral of the story: **crashes are good! (For testing)**

Code coverage criteria can measure how good a test suite:

statement coverage (req 1 test case)

branch coverage (req 2 test case)

Statement coverage does not imply branch coverage

Code coverage metrics can also be used to guide test case generation

High coverage criteria may discourage defensive programming; programmers may be tempted (or forced?) to remove this code to improve test coverage.

Annotations, e.g. SAL annotations of C/C++ code, informationflow policies can be **used as test oracle** by doing **runtime assertion checking.**

The annotations are defined in the header file <sal.h>

Normal Testing

• Normal (functional) testing focuses on **correct, desired behaviour** for sensible inputs (aka the happy flow but will include some inputs for borderline conditions (**reveal functional problems**)

Security Testing

• Security testing (also) —especially —looks **for wrong, undesired behaviour** for really strange inputs (**reveals security problems**)

Abuse cases give rise to negative test cases

FUZZING

fuzz testing of C/C++ code for memory corruption

generate 'random' inputs and check if an application crashes or misbehaves in observable way

Fuzzing aka fuzz testing is a highly effective, largely automated, security testing technique

URL Fuzzing

Fuzz urls to find hidden directories in a web application.

Fuzzing Types:
1. Basic fuzzing with random/long inputs

2. 'Dumb' mutational fuzzing

    example: ocpp

3. Generational fuzzing aka grammar-based fuzzing

example: GSM

4. Code-coverage guided evolutionary fuzzing with afl

aka grey box fuzzing or 'smart' mutational fuzzing

5.Whitebox fuzzing with SAGE (using symbolic execution)

Fuzzing Process

Depending on input type

1. very long inputs, very short inputs, or completely blank input
2. min/max values of integers, zero and negative values
3. depending on what you are fuzzing, include special values, characters or

keywords likely to trigger bugs, eg
nulls, newlines, or end-of-file characters
format string characters %s,%x,%n
semi-colons, slashes and backslashes, quotes
application specific keywords halt, DROP TABLES…….

Good validation and/or sanitisation would catch these problems.

pros

Very little effort: test cases are **automatically generated**, and **test oracle is trivial**

Fuzzing of a **C/C++ binary** quickly gives a **good indication of robustness of the code**

Cons

Only finds 'shallow' bugs and not 'deeper' bugs

— Ifa program takes complex inputs, smarter' fuzzing is needed to trigger bugs.

Crashes may be hard to analyse; but a crash is a clear true positive that something is wrong!unlike a complaint from a static analysis tool like **PREfast…..**

Ideally checks for both **spatial bugs (e.g. buffer overruns)** & **temporal bugs (e.g. malloc/free bugs)**

frankencert generator to auto-generate different test certificates involving complex

corner cases.