Introduction to Software Security

M. Ghafari
Software Composition Group
University of Bern

Fall 2017
Overview

- Security mindset
- CIA triad
- Secure Development Lifecycle
- Buffer overflow attack and the countermeasures
Why Security

We worry about security when we have something of value and there is a risk it could be harmed.

Sth of value like sensitive data (individuals, business and government), property, etc. Threats are adversaries and criminals can economically or politically abuse this data.
The real art of security is balance. Ask too many “what ifs”, and your security ends up costing too much. Ask too few, and you’ll leave critical risks unconsidered and wind up an easy target.

Security Mindset

- **Threats**: who are the bad actors?
- **Vulnerabilities**: what can possibility they exploit?
- **Risk**: if threats succeed to exploit a vulnerability, what is that attack/risk going to be?
We may lock the bike to protect it, as the one in the picture. Are there any vulnerabilities?
The thief is going walk away with the bike minus the wheel. We had to secure more than just a wheel.
The Threat Landscape

- Beyond 430 million unique pieces of malware exist;
- A new zero-day vulnerability was discovered, on average, once each week;
- Above 75% of all legitimate websites have unpatched vulnerabilities that can be exploited. And, 15% of them are critical vulnerabilities.

Symantec internet security report 2016
Threats

- **Cybercriminals**: want to profit from our sensitive data for financial gain.

- **Hacktivists**: activists who do not like something you are or something you do. E.g., Edward Snowden.

- **Nation-states**: countries do it for political advantage or for spying.
Vulnerability and Attack

- **Vulnerability**: The weak points in software that can lead to security concerns.

- **Attack**: When threats uncover the vulnerability, conduct research about it, and exploit it to launch their schemes.

[https://nvd.nist.gov](https://nvd.nist.gov) reports different common vulnerabilities
The alternatives

- Make threats go away
- Reduce vulnerabilities
  - Strive to meet security requirements of sensitive information:
    - Confidentiality
    - Integrity
    - Availability
Confidentiality

- Encryption
- Access control (rules and policies)
  - Based on identity, role
- Authentication
  - password, card, finger print
- Authorization
- Physical security
  - Locked windowless rooms, faraday cages
Integrity

- Backup
- Checksum
- Data correcting codes
Availability

- Physical protection
- Computational redundancies
Example

- **Confidentiality:**
  - encryption of traffic data, time out for invalid inputs, return invalid card, retain stolen card, use of TAN in net-banking

- **Integrity:**
  - consistency of data during transmission

- **Availability:**
  - diverse network, fair resource sharing
We strive to meet security requirements using various mechanisms but we never ensure there is no vulnerability in the system.

What should we do

- Detection
- Response
- Recovery
The role of software security

Software security,

• is not only about reactive technologies like firewalls, intrusion detection systems, and antivirus engines.
• is the property of software.
• is an engineering discipline.

Are you shipping a product to customer? Are you releasing a public web app/service? Are you consuming, storing, and returning customer data? If the answer is no, stop adopting SDL.
Every stakeholder in the software development life cycle has a part to play in ensuring software security and, therefore, every stakeholder requires training. Threat modeling is to anticipate the attacks to which the system may be subjected and also is used to help ensure that proper mitigations are in place to thwart or reduce the effectiveness of these attacks. Secure design, coding, and testing is to ensure that the mitigations compiled to address the threats identified in the threat model are properly designed, implemented, and tested. Privacy is learning best practices related to the proper privacy concerns for handling anonymous data, pseudo-anonymous data, personally identifiable information (PII), and sensitive PII.

Training

All stakeholders should stay informed about security basics and recent trends in the field.

Core secure trainings:
- Threat modeling
- Secure design, coding, and testing
- Privacy
Example of a quality gate in coding phase: all compiler warnings must be triaged and fixed prior to code check-in. A bug bar defines the class (e.g., spoofing, tampering, etc), scope, and severity of vulnerabilities.
Design

- Design requirements

- Attack surface reduction (ASR)
  Big Attack Surface = Big Security Work = Big Security Problems

- Threat modeling
Development teams should strive to use the latest version of the tools approved by the project security advisor to take advantage of new security analysis functionality and protections.
The Threat model and ASR review is to ensure that any design or implementation changes to the system have been accounted for, and new risks have been reviewed and mitigated.
Even programs with no known vulnerabilities at the time of release can be subject to new threats that emerge over time. Thus, having an incident response plan is necessary.

- Incident response plan
  - Sustained engineering resources
  - On-call contacts with decision-making authority
  - Service plans for codes inherited from others
- Final security review (FSR)
  - Passed
  - Passed with exceptions
  - FSR with Escalation
- Certifying release and archive
Response

- Execute incident response plan
Buffer Overflow Attack
Basic concepts

• **Buffer** is a contiguous block of computer memory that holds multiple instances of the same data type (like array in C).

• **Buffer Overflow** happens when one puts more data in a buffer that it can handle.
Address Space

- **Text**: machine code of the program, compiled from the source code.
- **Data**: static program variables initialized in the source code before execution.
- **BSS**: static uninitialized variables.
- **Heap**: data dynamically generated during the execution.
- **Stack**: structure that keeps track of the activated methods, their arguments, and local variables.
Stack is viewed as a continues block of memory. Its size is dynamically adjusted by the kernel at run time. A register called the stack pointer (SP) points to the top of the stack. The CPU implements instructions to PUSH onto and POP off of the stack.
When a program starts running in the memory, the stack always points to the very highest memory address that it can use. Assume the next to be executed instruction is the `sum` method call in the `main` method. The following steps would be taken:

```java
int sum(int a, int b){
    int sum;
    sum = a + b;
    return sum;
}

void main(){
    sum(10, 5);
    // next instruction
}
```
A function call is found, push parameters on the stack (in reverse order from right to left). So, 5 will be pushed first and then 10.

Example

```c
int sum(int a, int b){
    int sum;
    sum = a + b;
    return sum;
}

void main(){
    sum(10,5);
    // next instruction
}
```
We need to know where to return after function sum call is completed, so push the address of the next instruction on the stack.
As we are in a new function we need to update “BP” (to know in which function the execution happens). Before updating we save it on the stack so we can return later back to the main (the function we were before calling a new function). So, old BP is pushed on the stack.
Push local variable **sum** onto the stack.
Each register is 4-bytes (when the binary is compiled for a 32-bit operating system)
If buffer >= 500 then an overflow happens. An attack could happen when attacker changes the return address to point to his malicious command.

/x90 is a no-operation (NOP) instruction meant to "slide" the CPU's instruction execution flow. Attackers need NOPs as stack randomization and other runtime differences may make the address where the program will jump impossible to predict, so the attacker places a NOP sled in a big range of memory to make the target address bigger. If the program jumps to anywhere into the sled, it will run all the remaining NOPs, doing nothing, and then will run the shell code, just next to the sled.
In some cases it’s not easy to find room for your shell code on the stack and get it executed (e.g. the system does not allow the execution). There is a variation of Stack Overflow attack called return-to-libc in which a subroutine return address on a call stack is replaced by an address of a subroutine that is already present in the memory. Though the attacker could make the code return anywhere, libc, C library, is the most likely target, as it is almost always linked to the program, and it provides useful calls for an attacker such as the system function used to execute shell commands.
When control transfers to shell code, attacker gets the same privilege as the host program during execution (e.g., can be a system process with root access).
Protection (Safe Program level)

- Using strongly typed languages
- Automatic memory management and bound checking
  - Range checking: check that a number is within a certain range.
  - Index checking: check that in all expressions indexing an array the index value is checked against the bounds of the array.

C# and Java are examples of safe languages in which buffer overflow does not take place.
Protection (Unsafe Program level)

- Input checking
- Use safer functions that do “bounds checking”
  – e.g., think of “bounded string copy” in which a programmer should specify the number of characters to copy.
- Use automatic tools that analyze your code (source or executable) and warn about potential unsafe code fragments
  – e.g., see a list of them at https://www.owasp.org
**Protection (OS level)**

- **Canary**: checking the consistency of a particular value added before a return address in the memory.
- **ASLR**: randomizing the memory address space.
- **NX**: preventing to fetch instructions from the stack segment.