Software Security
and
Secure Software Development Life Cycle
(S-SDLC)

P.R.Lakshmi Eswari

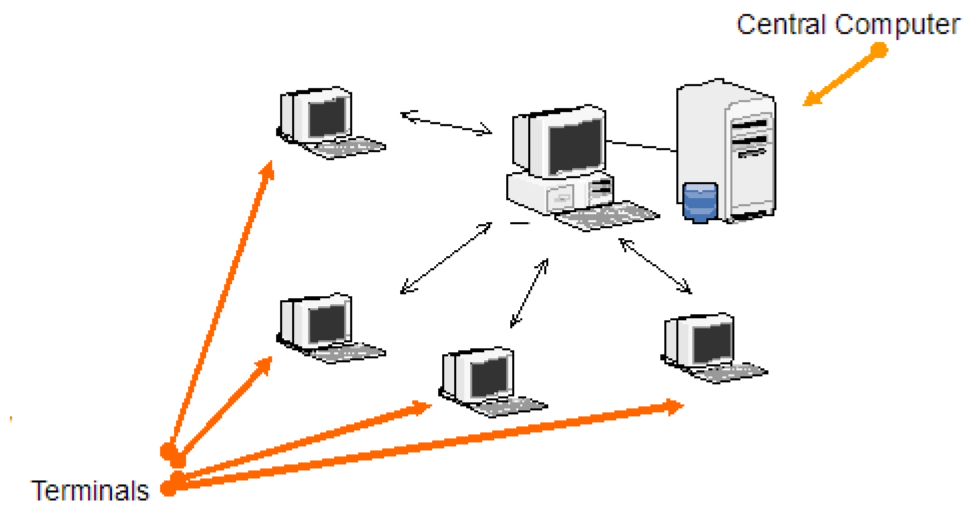
Senior Director and Centre Head

C-DAC, Hyderabad

Agenda

- Evolution of Computer & Network Systems and Security Threats
- Significant Breaches
- Cyber Security & Solutions
- Holistic Approach to Cyber Security
- Need for Software Security
- Software Vulnerabilities & Patching
- Evolution of Software Security
- Secure Software Development Life Cycle (S-SDLC)
- Important Steps towards Software Security

Centralized Environments



- Central Computer (Server) is accessed through terminals (dummy)
- Setup is limited to a building

Security Mechanisms

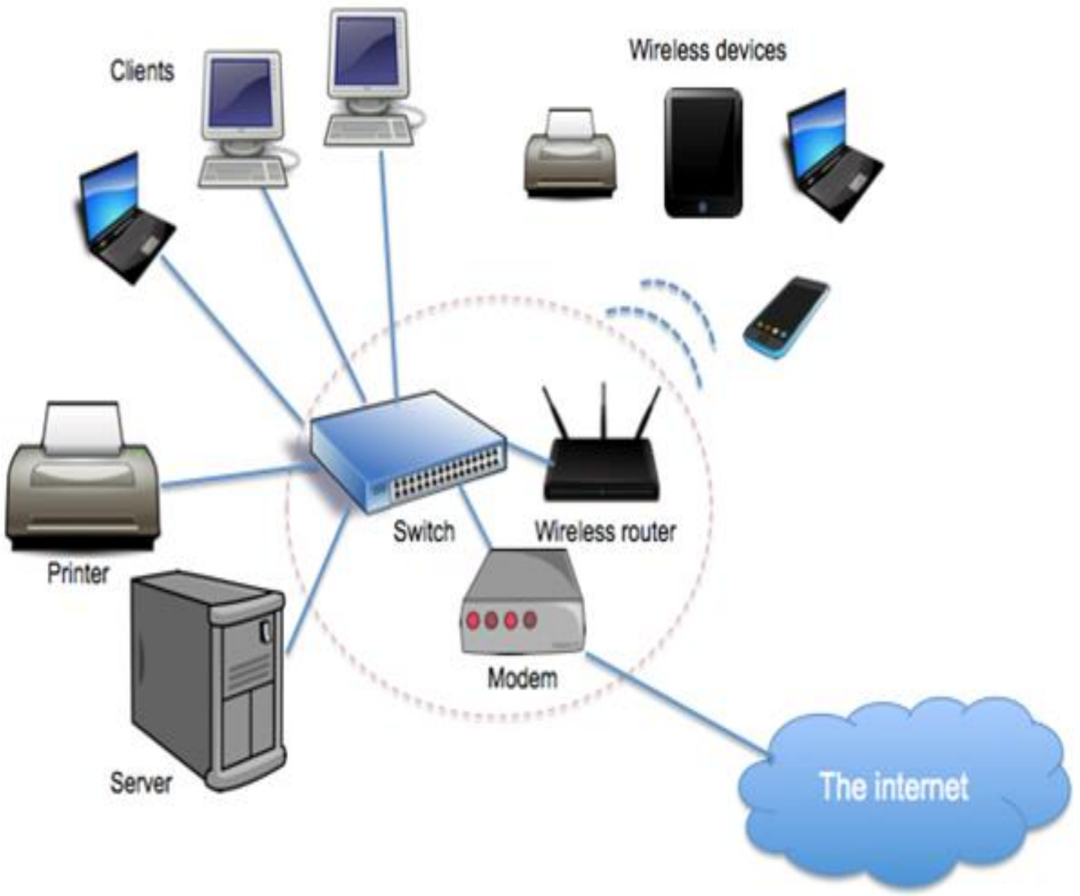
•Password based Authentication

•Access Controls for resources (file system)

- ✓ **Privileges (Read, Write and Execute)**
- ✓ **Type of user (owner, group, others)**

Present Day Computer Networks & Security Threats

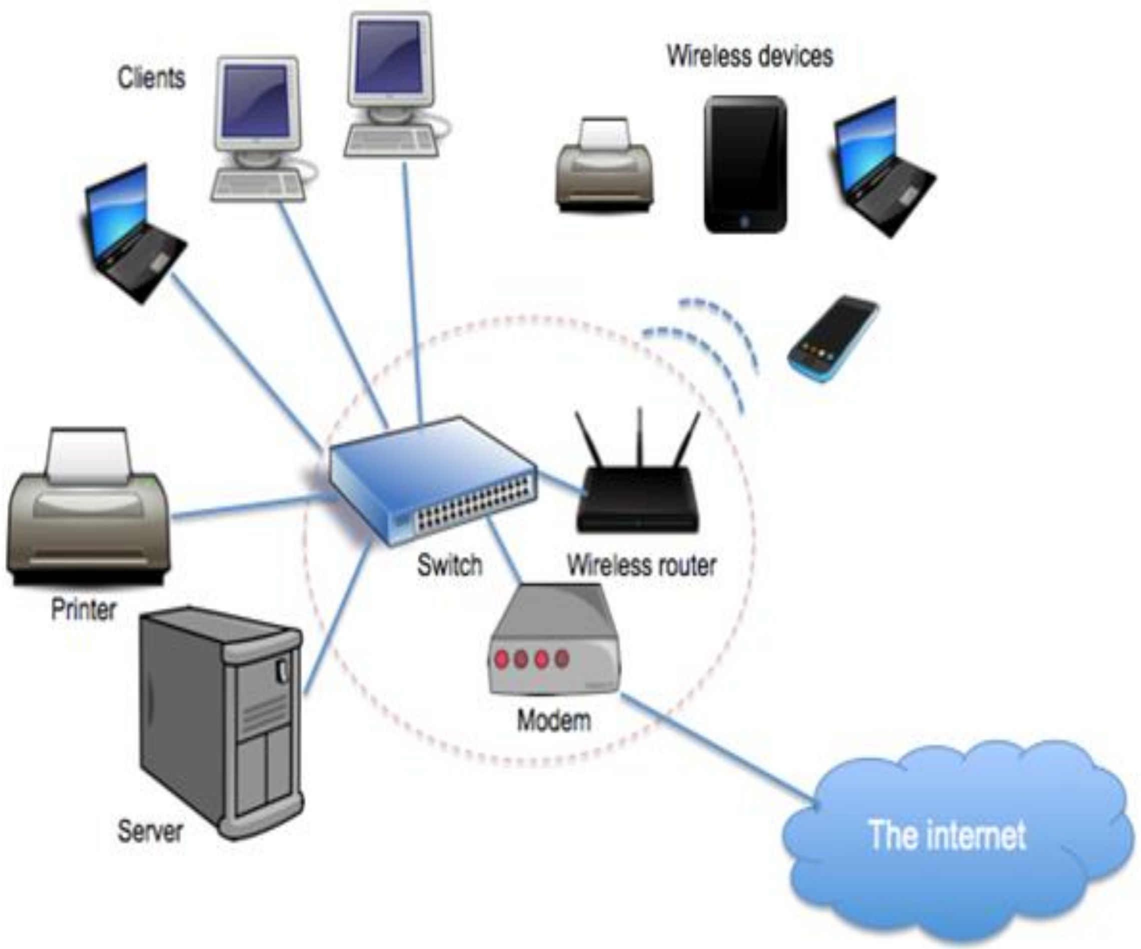
- Viruses & Worms
- Ransomware
- Botnets
- Phishing Attacks
- Distributed Denial of Service (DDoS)
- Drive-by-Download Attacks
- SQL Injection
- Exploit Kits
- Cryptojacking or Malicious Cryptomining
- Shadow IT Threats
- Advanced Persistent Threats (APTs)
- Zero-Days
- ……..



Present Day Computer Networks & Required Security Services

Security Services

- Authentication (Peer entity & Data Origin)
- Access Control
- Data Confidentiality
- Data Integrity
- Non-Repudiation



Types of Malicious / Undesired Behavior

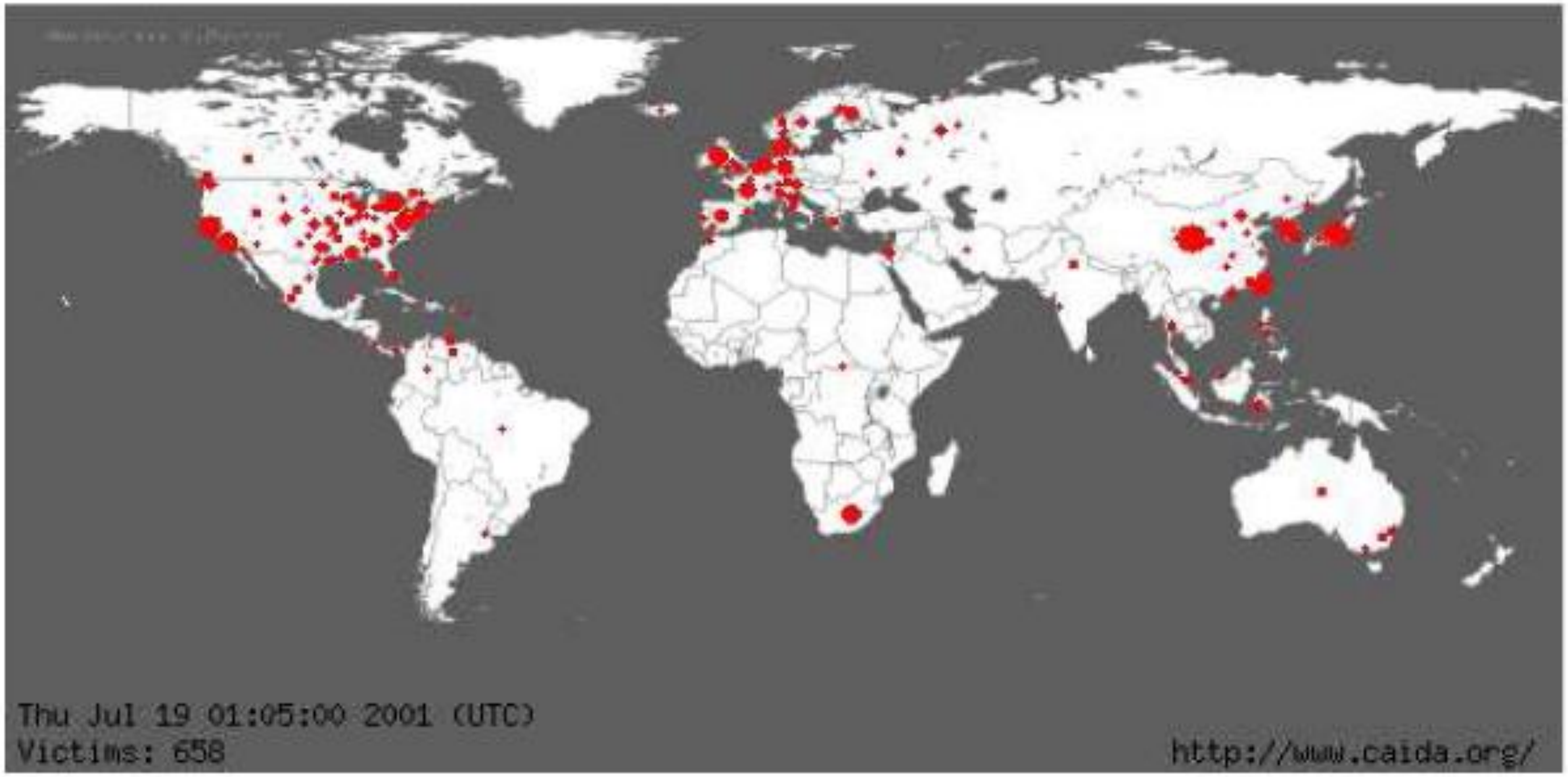
- Stealing information: ~~Confidentiality~~
  - ✓ Corporate Secrets (Source code, financial details, trade secrets etc)
  - ✓ Personal information (passwords, PIN, credit card numbers etc)

- Modifying information or functionality: ~~Integrity~~
  - ✓ Installing unauthorized software (spyware, botnet client etc)
  - ✓ Destroying Records (accounts, logs etc)

- Denying access: ~~Availability~~
  - ✓ Unable to purchase products
  - ✓ Unable to access banking service

Significant Security Breaches

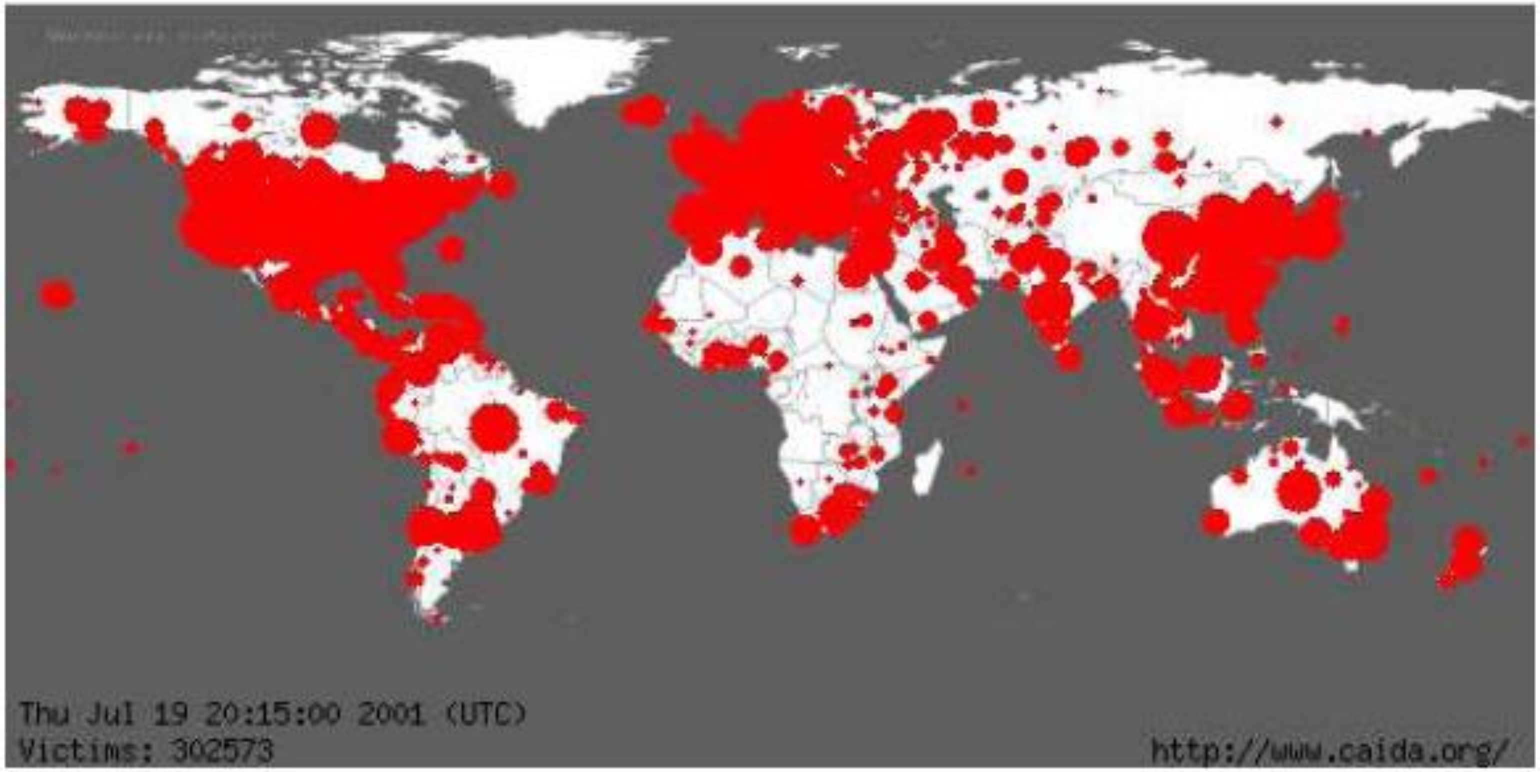
Code Red Worm



July 19 01:05:00 2001

19 Hours Later

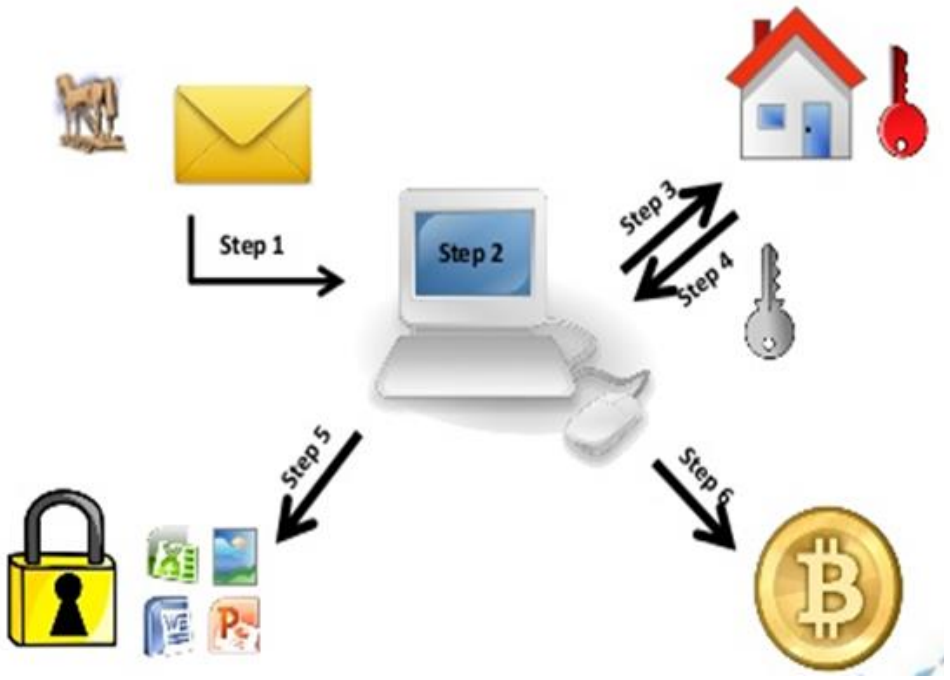
Code Red Worm



July 19 20:15:00 2001

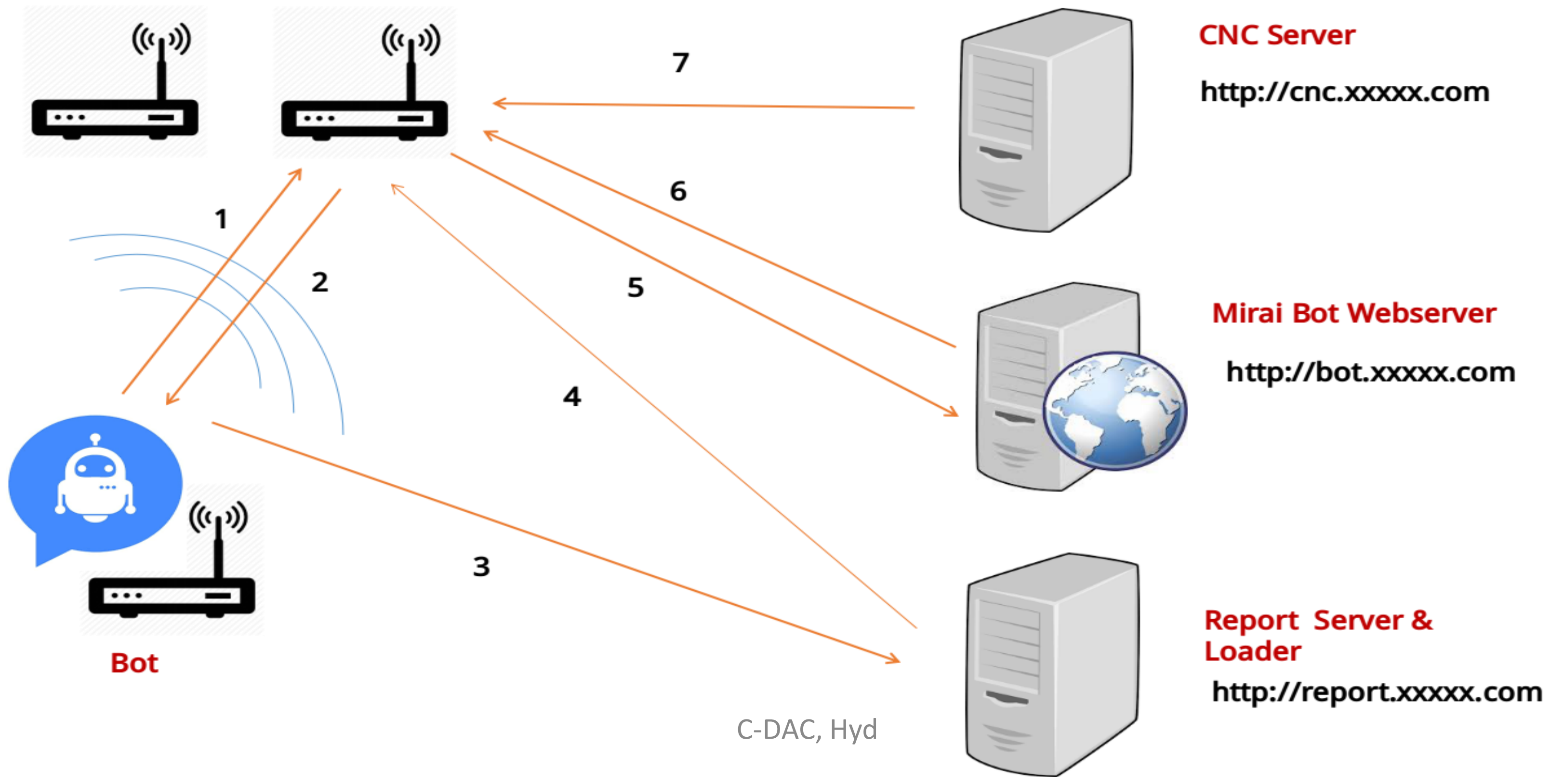
WannaCry Ransomware

- WannaCry - ransomware spread rapidly over Internet during May 2017
  - Infects Windows computers
  - Encrypts files on the hard disk, making impossible for users to access them
  - Demands a ransom payment in bitcoin in order to decrypt them



Mirai

- Mirai - malware targeting networked devices remotely controlled "bots“ used as part of a botnet in large-scale network attacks.
  - targets online consumer devices such as IP cameras and home routers. Launched DDoS attacks.



Other Major Breaches in Recent Times

PoS malware targeting point-of-sale (PoS) terminals with the intention of stealing payment card data

- A PoS malware **enters through compromised or weakly secured systems** and scrapes the RAM to find payment card data

Stuxnet targets supervisory control and data acquisition (SCADA) systems

- Believed responsible for causing severe damage to the nuclear program of Iran
- **Exploited four previously unknown vulnerabilities**
- Duqu and Flame were in similar lines

Other Major Breaches in Recent Times

Heartbleed in OpenSSL's implementation allows attackers to read portions of the affected server's memory

- Revealing users data that the server was not supposed to reveal

Operation Aurora exploited zero-day vulnerability in Microsoft Internet Explorer.

Google is Fighting A Massive Android Malware Outbreak

Malware Attacks

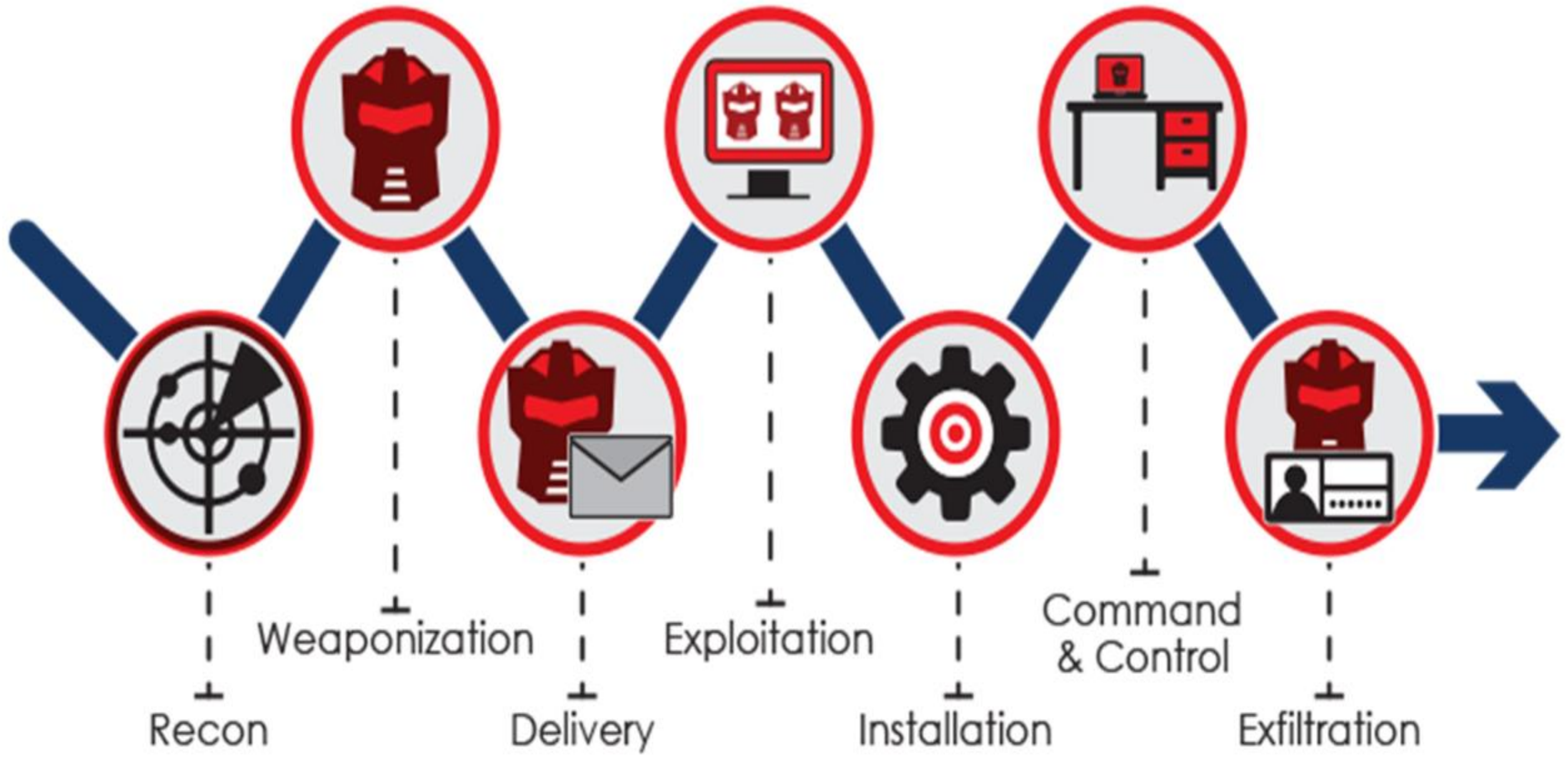
File-based

- Common / mass malware
- Targeted malware
- Mutated Malware

File-less

- Memory-based
- Weaponized Content
- Script-based
- Registry-based
- Rootkits

Advanced Persistent Threats (multi-stage)

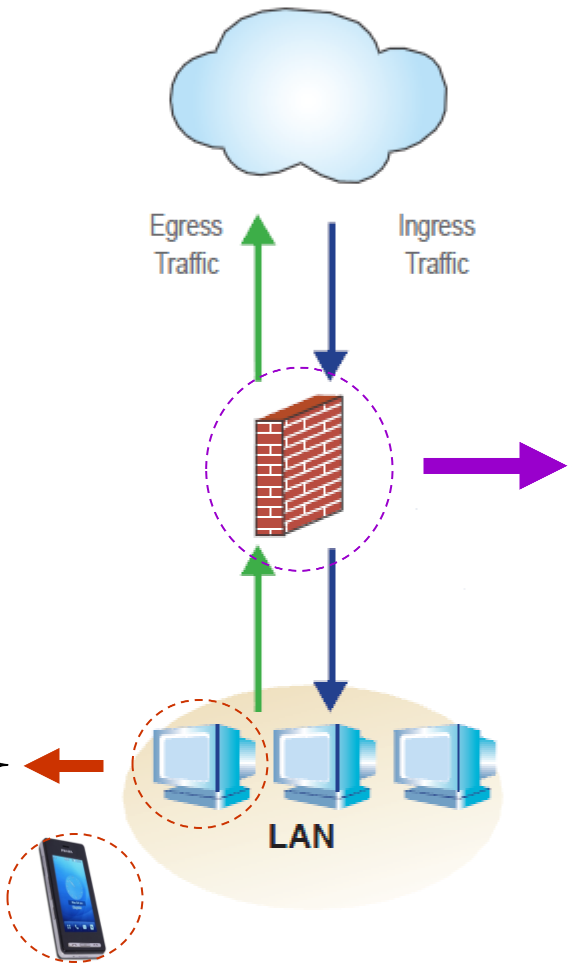


Modern attacks have **multiple stages** and persistent, follow a "low-and-slow" approach over a long period of time. The **attack** can take **months** to **develop** and even take **longer time to deploy**.

What is Cyber Security?

- Focus of computer / online applications
  - Achieving desired behavior
  - Online banking site, online shopping site, MS word, Browser etc
- Hackers / adversary actively make efforts to cause undesired (malicious) behavior
- Cyber Security is concerned with preventing undesired (malicious) behavior
  - Collection of mechanisms / tools designed to protect data & operations and to prevent hackers / adversary
  - Cryptographic algorithms, Firewalls, Intrusion Detection Systems, Anti Malware solutions etc
- Holistic approach to Cybersecurity to deal with Vulnerabilities

Cyber Security Solutions & Deployment - Technology



- ✓ Anti Spyware
- ✓ Anti Virus
- ✓ IDP / IPS
- ✓ VPN
- ✓ UTM box
- ✓ Threat Intelligence

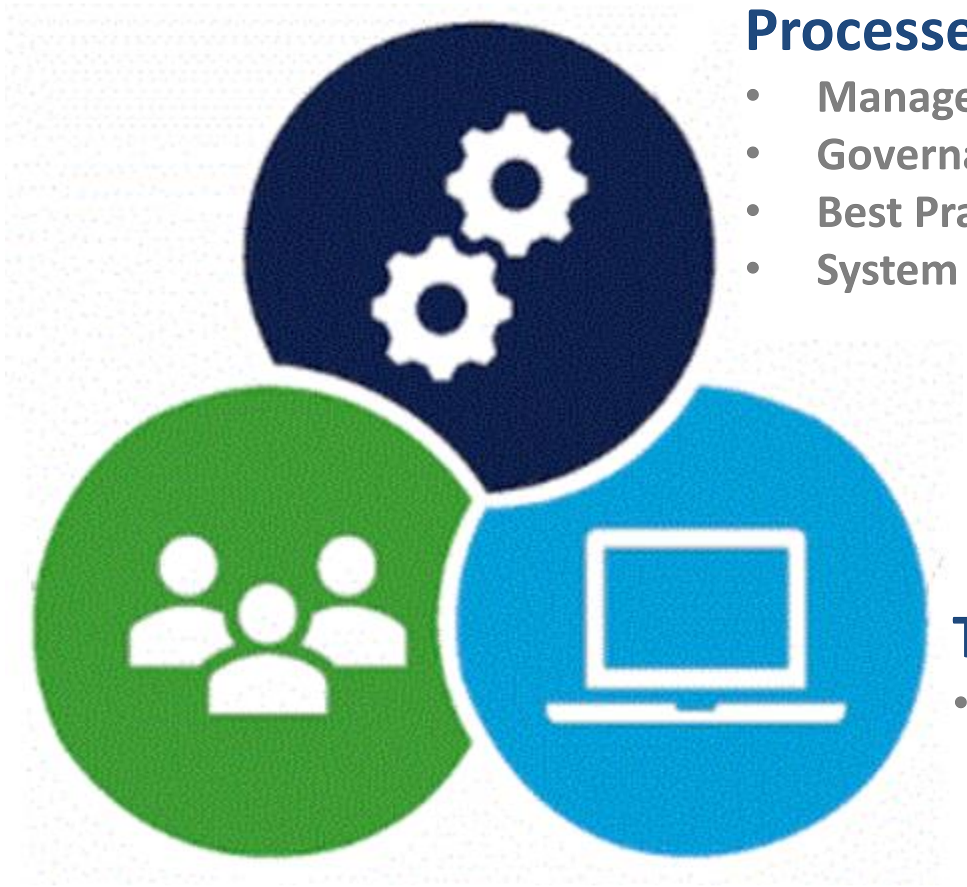
End Point Security Management

- ✓ Anti Virus (Signature, Behavior & heuristic based)
- ✓ Desktop Firewall
- ✓ App Control, Device Control
- ✓ Host IDS
- ✓OS Hardening
- ✓Trusted Execution Environment

Vulnerability Types

- Vulnerabilities in
  - Hardware, Software and Network Layers
  - Processes
  - Human or People

Holistic Approach to Cyber Security



Processes

- Management Systems
- Governance Frameworks
- Best Practices
- System Audits

People

- Training and Awareness
- Qualified and Skilled manpower
- Human Ethics

Technology

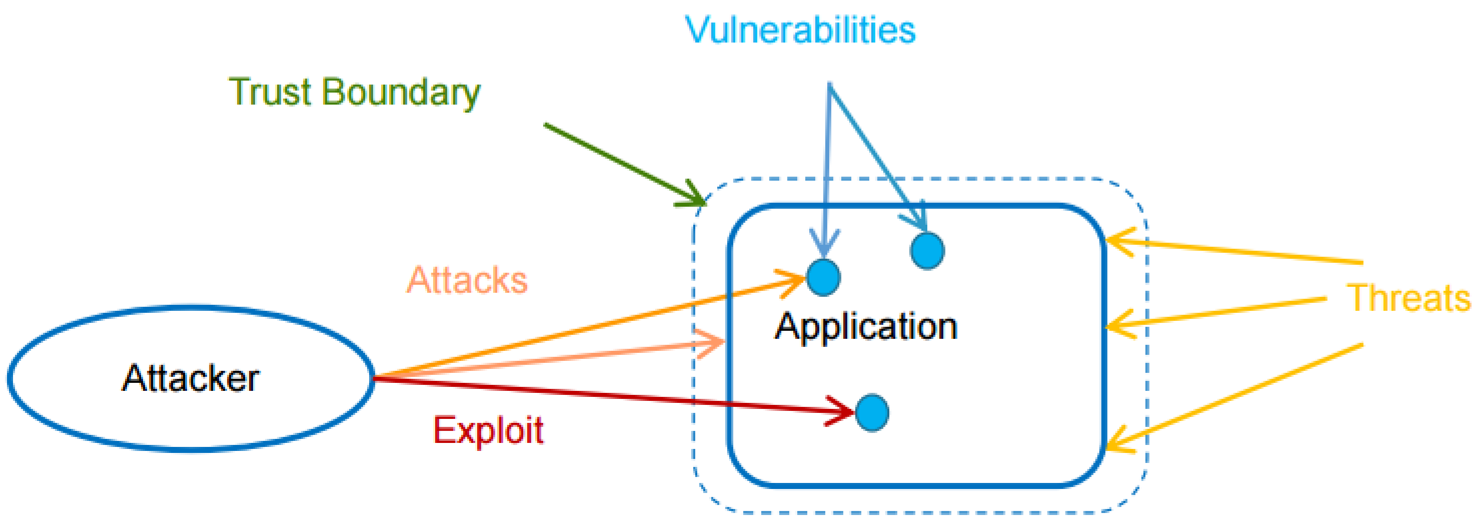
- Defense in Depth solution with evolving threats

Need for Software Security

- Firewalls, intrusion detection / prevention systems, malware detection solutions etc., are mostly reactive in nature, in addressing the cyber security problem.

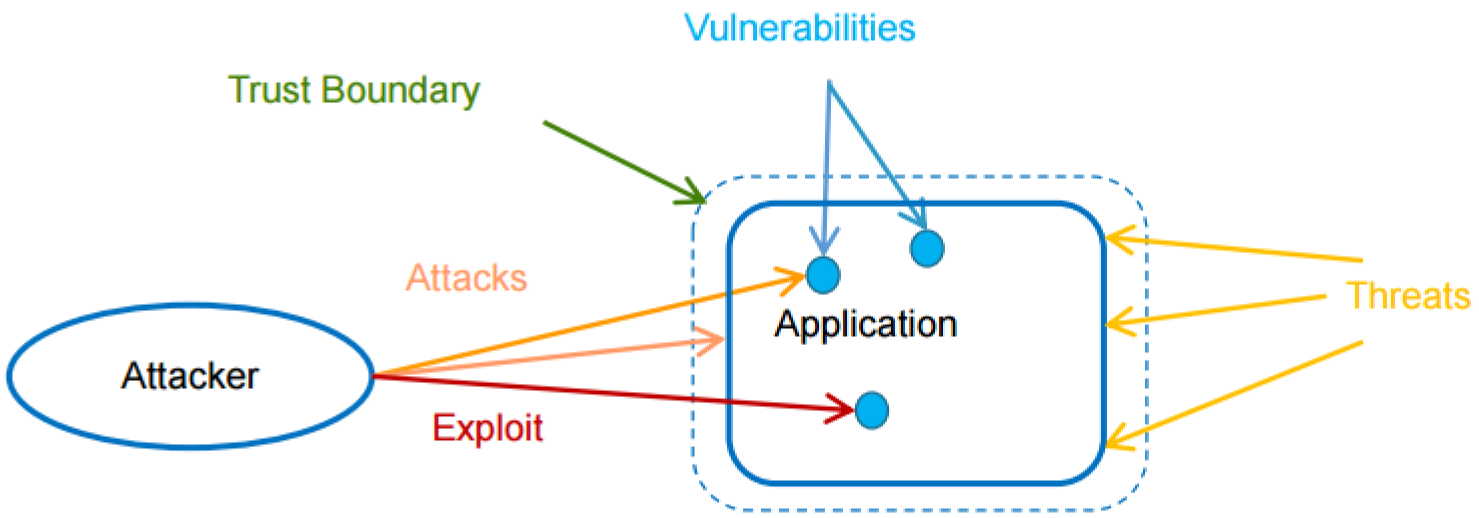
- *These technologies don’t prevent the vulnerabilities.*

Trust Boundary, Bug and Vulnerability



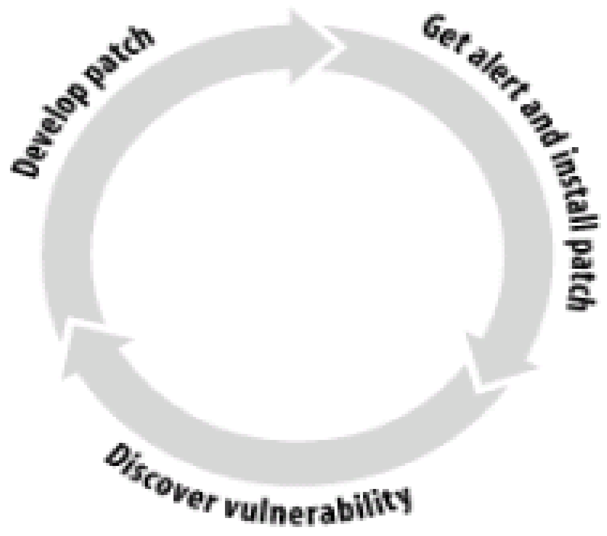
- **Trust Boundary:** Where the level of trust changes for data or code.
- A **bug** is a system flaw / error that makes it possible to bring the vulnerability to life.
- A **vulnerability** is a flaw in a system which is an exploitable weakness to breach confidentiality, integrity and availability. It has potential to be leveraged by a threat agent in order to compromise a secure network.

Threat, Exploit and Attack



- A **threat** is something bad that can happen in a business sense. An example - Unauthorized release of confidential financial projections.
- An **attack** occurs when an attempt is made to manipulate a system to execute a threat.
- **Exploit:** is a piece of code or sequence of commands that takes advantage of vulnerability to cause malicious behaviour

Vulnerability/Patch/Alarm Cycle



Zero-day Attack

- A **zero-day attack** or threat is an attack that exploits a previously unknown vulnerability in a computer application.
- This means that the developers have had zero days to address and patch the vulnerability.
- Zero-day exploits are used or shared by attackers before the developer of the target software knows about the vulnerability.

Types of Software Security Vulnerabilities

- Buffer overflows
- Invalidated input
- Race conditions
- Access-control problems
- Weaknesses in authentication, authorization, or cryptographic practices
- …..

Buffer Overflow

- In C and C++, there is no automatic bound checking on the buffer, which allows the user to write past the buffer

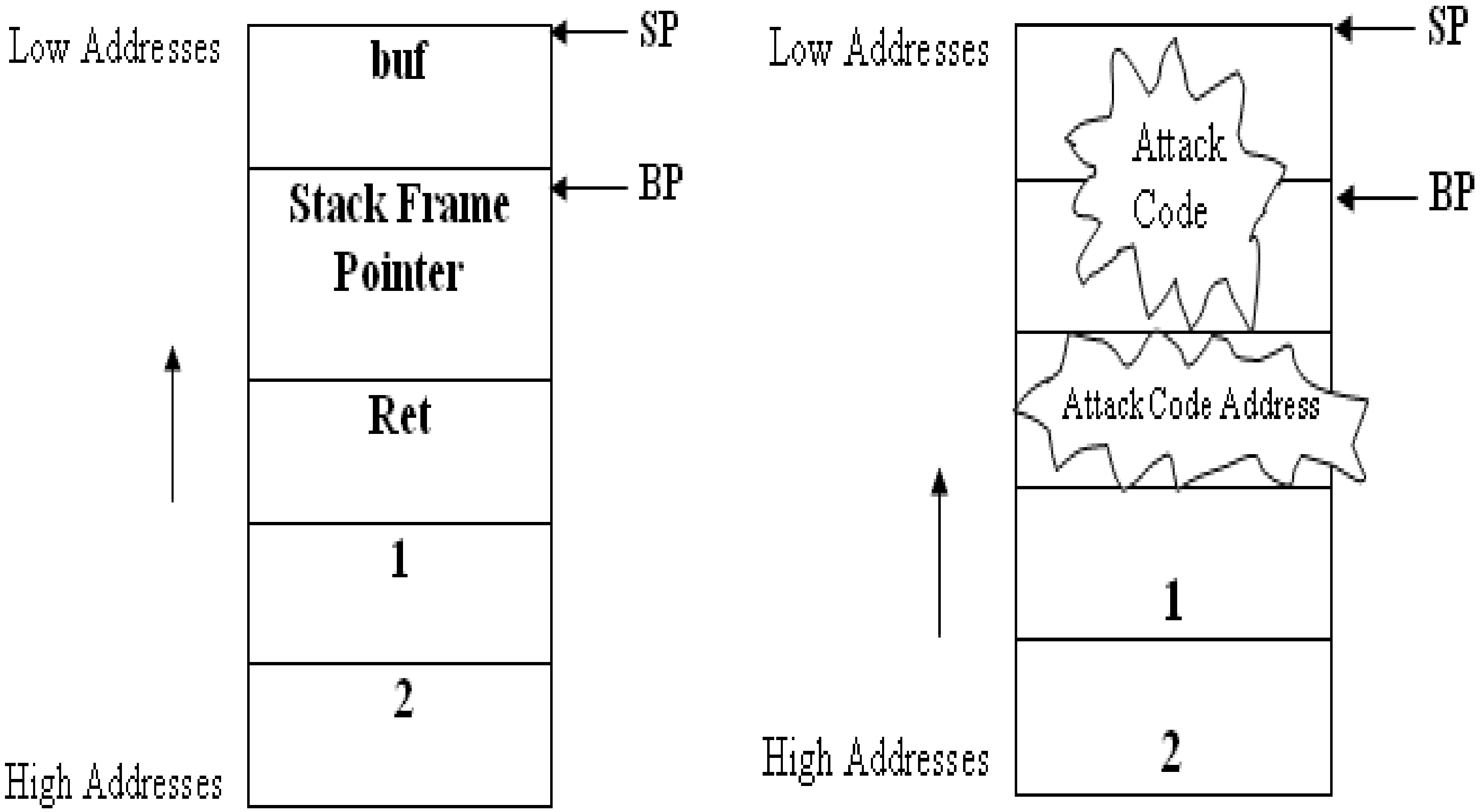
```
int main()
{
        int buf[10];
        buf[20]=15;
        return(0);
}
```

Buffer Overflow

- In C and C++, there is no automatic bound checking on the buffer, which allows the user to write past the buffer

```
int main(… arguments…)
{
        char buf[10];
        scanf("%s", buf);
        ……..
        return(0);
}
```

Stack Based Buffer Overflow Attack Generation



Buffer Overflow

- **Buffer Overflow** is a **memory usage vulnerability** and is the most common form of security vulnerability

- Buffer Overflow occurs anytime the program writes **more information** into the buffer than the **space** it has **allocated** in the memory.

- Well-known **Internet worms**, including **Code Red**, **Slapper** and **Slammer**, use buffer overflow attacks to propagate and execute payloads.

Integer Overflows

- Integer overflows occur when input numerically extends beyond the space available for that data type
- Consider the situation
  - The jeweler advertises one item for $2,500.00.
  - The attacker orders 17,180 of the item
  - Costing a grand total of $327.04!
  - How did that happen?
    - $2,500.00 X 17,180 is too big number and integer overflow is occurred
- Consider another situation
  - 25,000 plus 25,000 prints -15536
  - How did that happen?
    - Because 25,000 plus 25,000 is too big number to fit in 15 bits, so the math overflows and changes the sign.

SQL Injection Attack

- SQL queries are inserted in web forms maliciously to extract sensitive information from database

- SELECT *fieldlist* FROM *table* WHERE *field* = **'anything' OR 'x'='x'**;

The Following things might result from SQL Injection

- The user could log in to the application as another user, even as an administrator.
- The user could view private information belonging to other users e.g. details of other users' profiles, their transaction details etc.
- The user could change application configuration information and the data of the other users.
- The user could modify the structure of the database; even delete tables in the application database.
- The user could take control of the database server and execute commands on it at will.

Why good people write bad code?

- Technical factors
- Psychological factors
- Real-world factors

Overall Thrust

- Identify common programming errors that lead to software vulnerabilities
- Establish standard secure coding practices
- Educate software developers
- Follow Secure Software Development Life Cycle

Secure Coding

- It is the practice of writing programs that are resistant to attack by malicious programs.

- Secure coding helps protect a user’s data from theft or corruption.

- An insecure program can provide access for an attacker to take **control of a server or a user’s computer**, resulting in anything from a **denial of service** to a single user to the **compromise of secrets**, loss of service, or **damage to the systems of thousands of users.**

- Secure coding is important for all software.

Detect and Handle Input and Output Errors

```
size_t read_at(FILE *file, long offset,
void *buf, size_t nbytes) {
  fseek(file, offset, SEEK_SET);
  return fread(buf, 1, nbytes, file);
}
```

Detect and Handle Input and Output Errors

```
size_t read_at(FILE *file, long offset,
void *buf, size_t nbytes) {
  fseek(file, offset, SEEK_SET);
  return fread(buf, 1, nbytes, file);
}
```

```
size_t read_at(FILE *file, long offset,
void *buf, size_t nbytes) {
  if (fseek(file, offset, SEEK_SET) != 0) {
    /* Indicate error to caller. */
    return 0;
  }
  return fread(buf, 1, nbytes, file);
}
```

Do not ignore values returned by functions

```
puts("foo");
```

Do not ignore values returned by functions

```
cout << "foo";
```

```
if (!(cout << "foo")) {
  /* Handle Error */
}
```

Functions should validate the parameters

```
void setfile(FILE *file) {
   myFile = file;
}
```

Functions should validate the parameters

```
void setfile(FILE *file) {
   myFile = file;
}
```

```
errno_t setfile(FILE *file) {
 if (file && !ferror(file) && !feof(file)) {
   myFile = file;
   return 0;
 }
```

Never hard code sensitive information

Noncompliant Code

```
class IPaddress {
String ipAddress = new String("172.16.254.1");
public static void main(String[] args) {
  //..
}
}
```

Never hard code sensitive information

Compliant Code

```
class IPaddress {
public static void main(String[] args) throws IOException {
char[] ipAddress = new char[100];
BufferedReader br = new BufferedReader(new InputStreamReader(
new FileInputStream("serveripaddress.txt")));
 // Reads the server IP address into the char array,
// returns the number of bytes read
  int n = br.read(ipAddress);
   // Validate server IP address
// Manually clear out the server IP address
  // immediately after use
for (int i = n - 1; i >= 0; i--) {
    ipAddress[i] = 0;
}
br.close();
}
}
```

Architectural Principles

- Principle of
  - least privilege
  - fail-safe defaults
  - economy of mechanism
  - complete mediation

Architectural Principles

- Principle of
  - Open design
  - Separation of privilege
  - Least common mechanism
  - Psychological acceptability

What is Software Security?

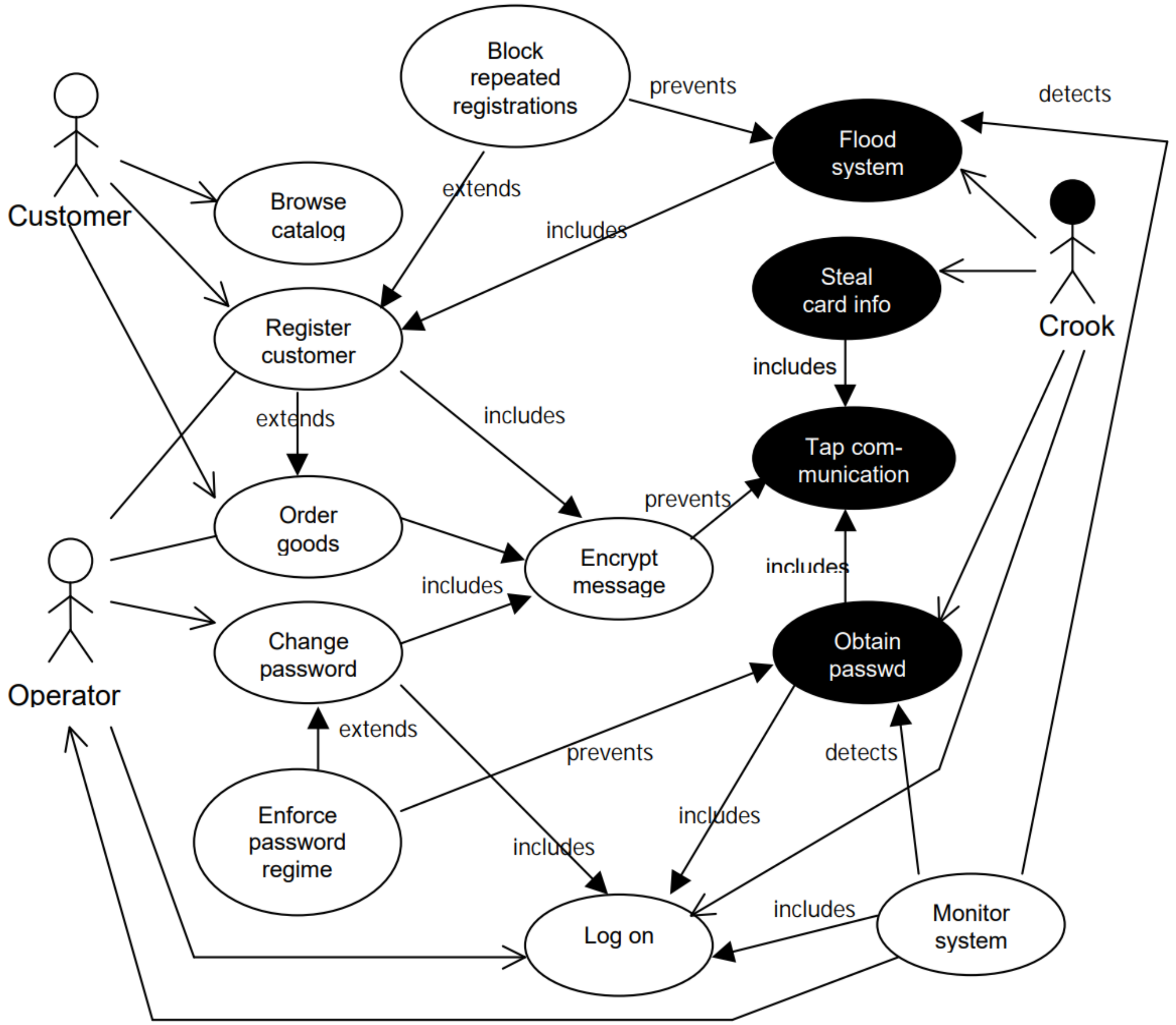
- Software security is about more than just eliminating vulnerabilities and conducting penetration tests
- Attempts to minimize the introduction of design flaws and bugs during a software development life cycle (SDLC)
- Network security mechanisms and IT infrastructure security services do not sufficiently protect application software from security risks

What is Software Security?

- Software security initiatives should follow a risk-management approach

- Project managers and software engineers need to learn to think like an attacker to address the range of functions that software should not do, and how software can better resist, tolerate, and recover when under attack.

Capturing Misuse (Abuse) Cases

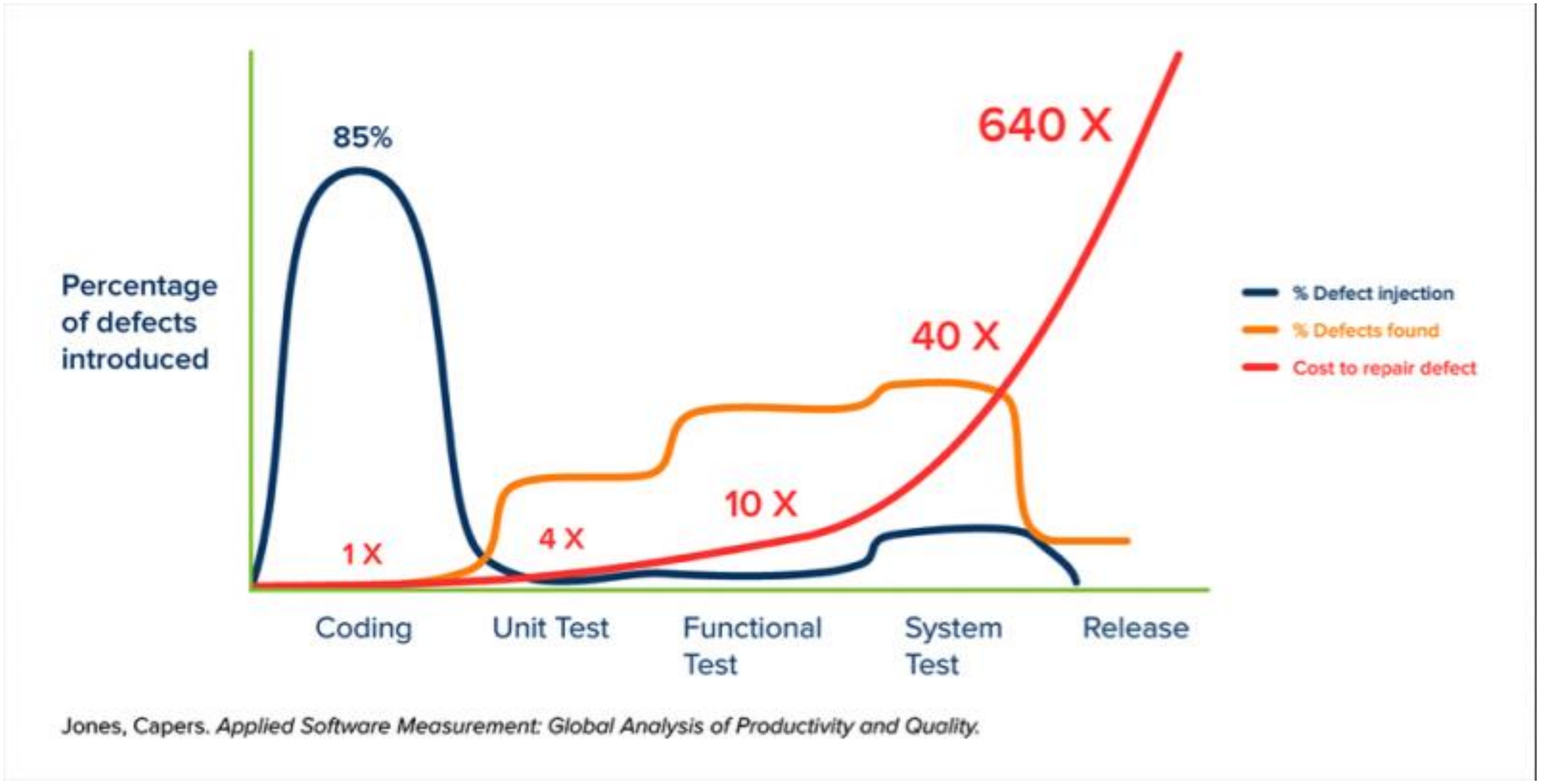


Evolution of software security

For example

1. Meet the requirements
   - some problems may arise & patch it
2. then, include patching also into the ecosystem
3. Why patching? Let us pen-test our products
4. Ok ...., Let us include using of static analysis tools when coding itself
5. But, programmers need to be aware of the problems ....then, train your programmers to know about common problems / best practices / secure coding practices etc..
6. then, think of abuse cases, and develop security tests for them
7. then, why not include security from the beginning of the development life cycle?

Why Secure SDLC?

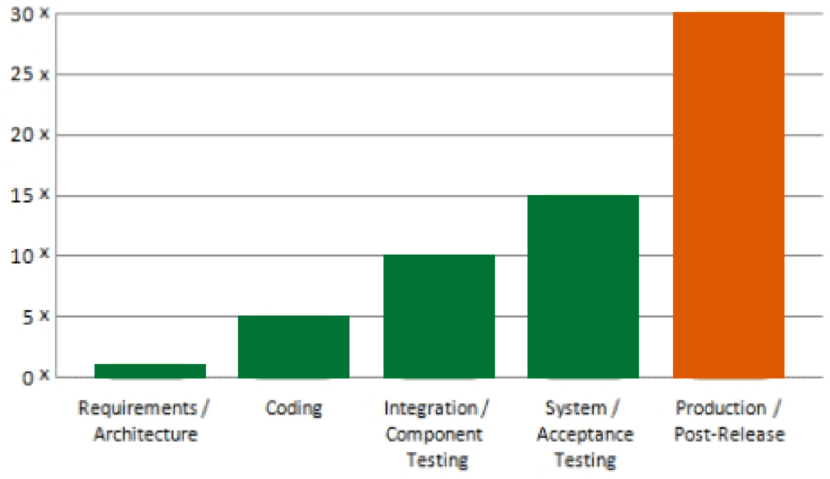


Jones, Capers. *Applied Software Measurement: Global Analysis of Productivity and Quality.*

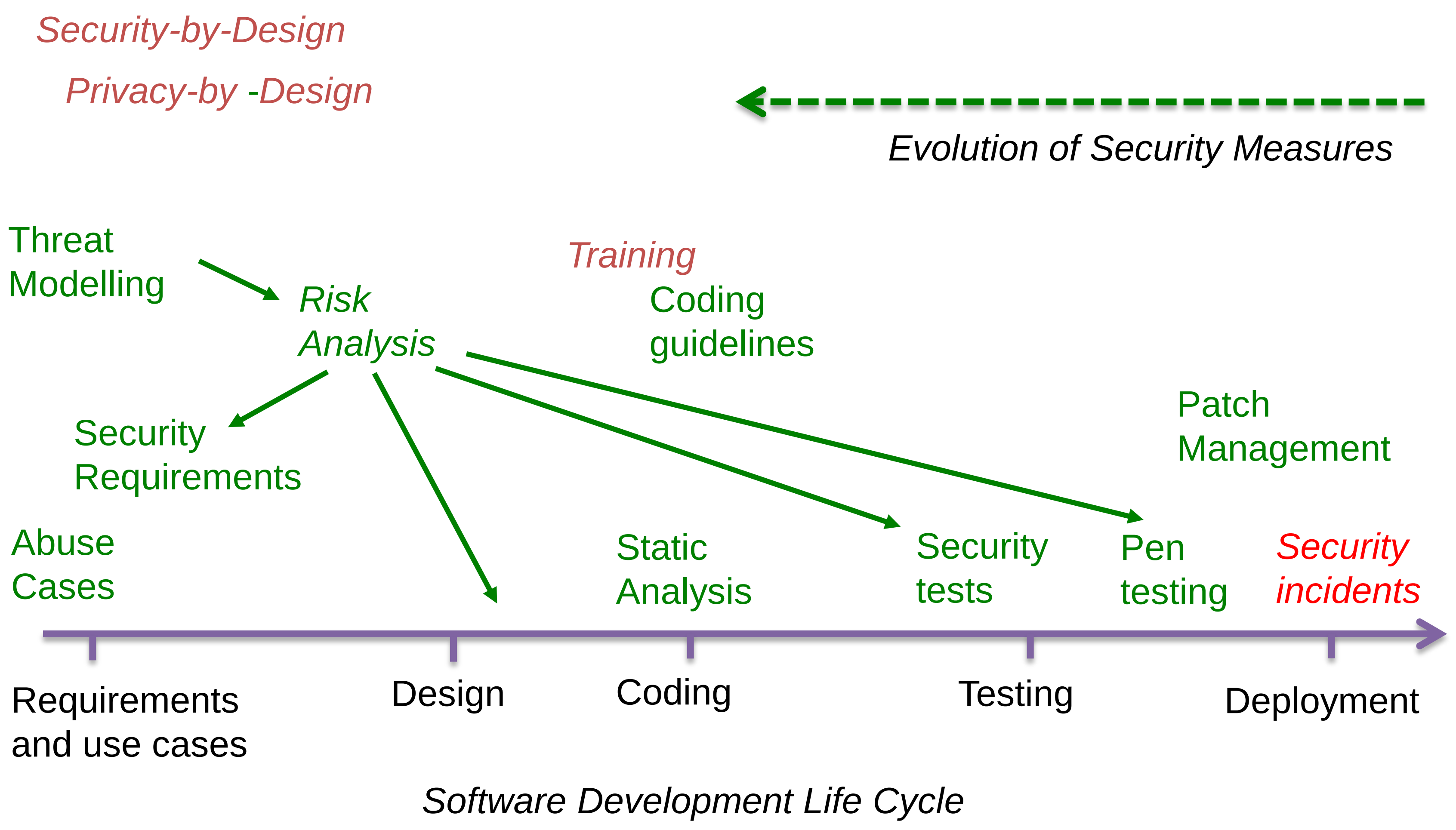
"Software is the weakest link in the security chain"

Advantages of pursuing secure SDLC

- Early detection of flaws in the system
- Cost reduction as a result of early detection and resolution of issues
- Overall reduction of intrinsic business risks for the organization



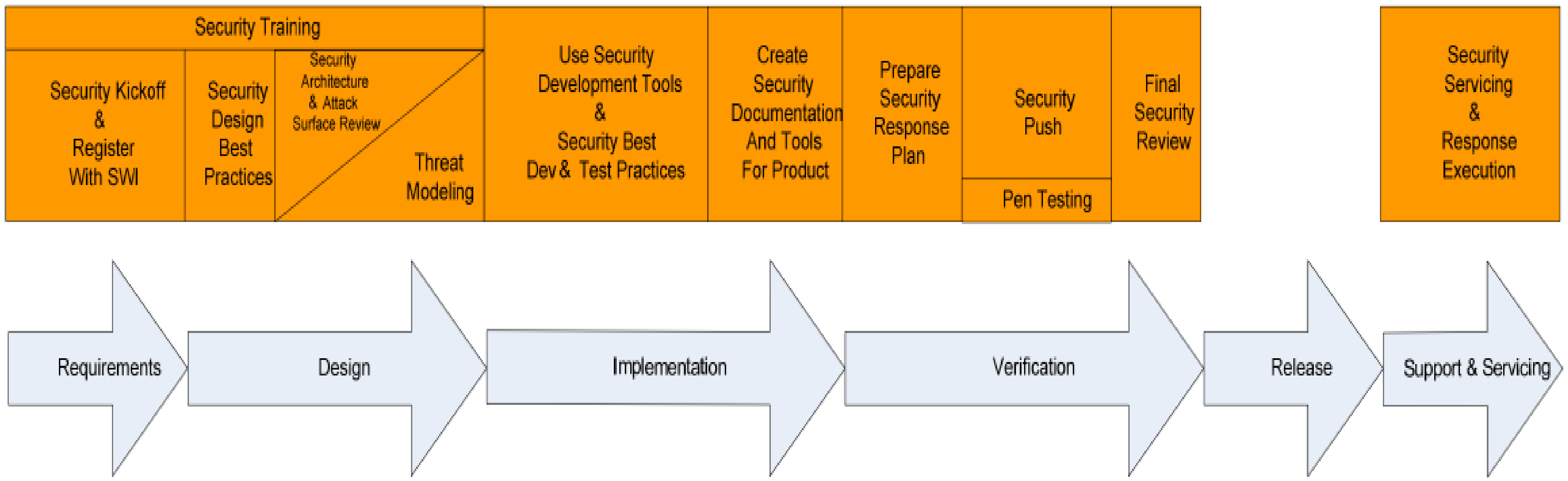
Security in Software Development Life Cycle



Trustworthy Computing Security Development Life Cycle

- **Secure by Design**
- **Secure by Default**
- **Secure in Deployment**
- **Communications**

Software Development Lifecycle Improvements to the Microsoft Development Process


Security Training
Security Kickoff & Register With SWI
Security Design Best Practices
Security Architecture & Attack Surface Review
Threat Modeling
Use Security Development Tools & Security Best Dev & Test Practices
Create Security Documentation And Tools For Product
Prepare Security Response Plan
Security Push
Pen Testing
Final Security Review
Security Servicing & Response Execution
Requirements
Design
Implementation
Verification
Release
Support & Servicing

Top 10 Secure Coding Practices

- Validate input
- Heed compiler warnings
- Architect and design for security policies
- Keep it simple
- Default deny
- Adhere to the principle of least privilege
- Sanitize data sent to other systems
- Practice defense in depth
- Use effective quality assurance techniques
- Adopt a secure coding standard

Additional Practices

- Defining security requirements and model threats

Important Steps towards Software Security

- Design of Misuse cases and Security Use Cases
- Detailed Threat Modeling & Attack Surface Analysis
- Following Architectural & Design Security principles
- Secure Coding Practices
- Static and Dynamic Analysis of Code (Source Code analysis, penetration testing etc)
- Design and Code Reviews
- Establish a process for handling vulnerabilities (patch management)

Experiences of Security Architects

- Defense in depth is better than relying on a single barrier.
- Design your systems as if your keenest adversary will be attacking them.
- Any design is only as secure as its weakest component.
- Require individual accountability.
- Build checkpoints (single points of entry) into your designs.
- Make use of the access control provided by the operating system. Don't rely on it.
- Modularize.
- Don't invoke untrusted programs from within trusted ones.
- Fail cleanly. Degrade gracefully.

Potential Research Areas

- Security analysis of software for detecting vulnerabilities
  - Static analysis
  - Runtime analysis
  - Fuzzing
- Executable inspection and analysis
- Machine Learning approach
- Containerization
- Static / Dynamic Malware Analysis
- Threat Intelligence Integration

References

- https://www.cybok.org/media/downloads/Secure_Software_Lifecycle_KA_-_draft_for_review_April_2019.pdf
- https://resources.sei.cmu.edu/asset_files/whitepaper/2013_019_001_297287.pdf
- https://subs.emis.de/LNI/Proceedings/Proceedings67/GI-Proceedings.67-3.pdf

Thank You