INF3510 Information Security

Lecture 12:
Application and Development Security

Audun Jøsang

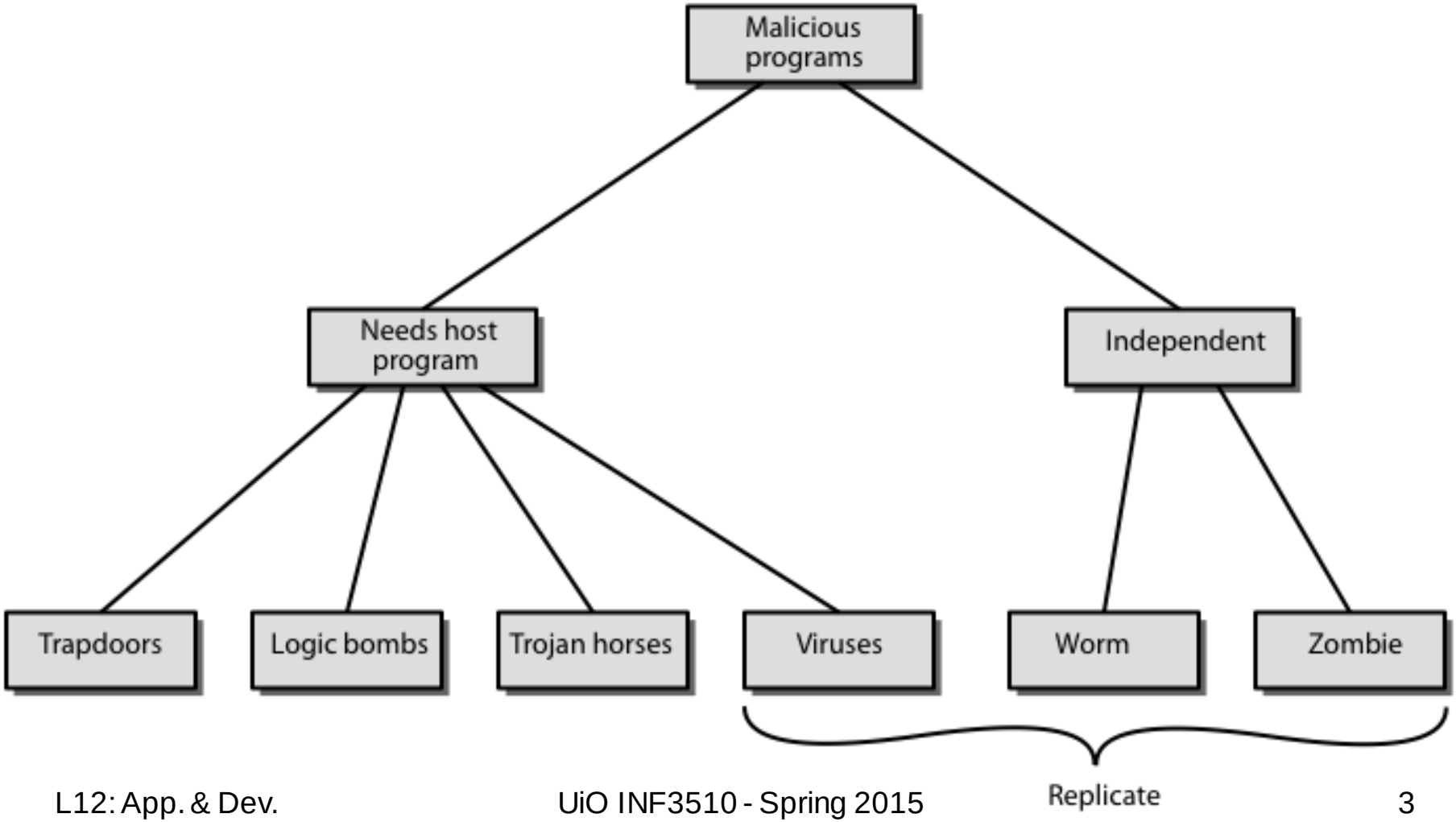
University of Oslo

Spring 2015

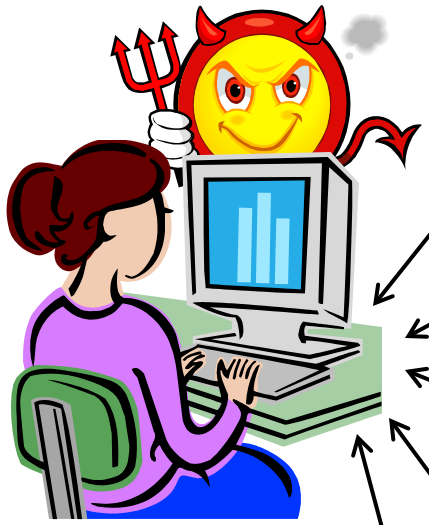
Outline

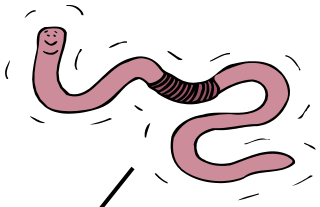
- Application Security
  - Malicious Software
  - Attacks on applications
- Software Development Security
  - Secure software development models
  - Security development maturity models

Malicious Software

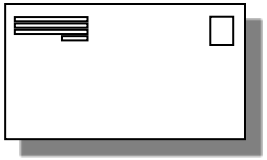


How do computers get infected ?

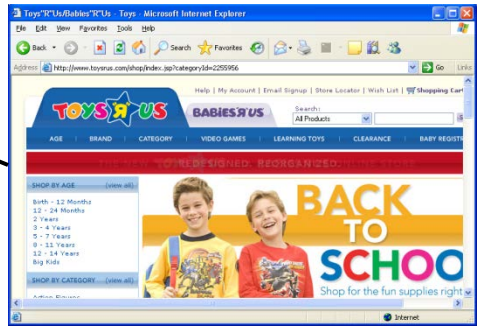




Direct attacks from the network, as worms or exploitation of application vulnerabilities such as SQL injection or buffer overflows

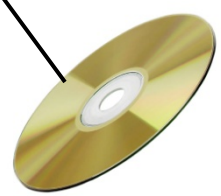


Executing a malicious attachment



Accessing a malicious or infected website or starting application from a website

Plugging in external devices



Installing infected software

Backdoor or Trapdoor

- is a secret entry point into a program,
- allows those who know access bypassing usual security procedures
- has been commonly used by developers for testing
- is a threat when left in production programs allowing exploited by attackers
- is very hard to block in O/S
- can be prevented with secure development lifecycle

Logic Bomb

- one of oldest types of malicious software
- code embedded in legitimate program
- activated when specified conditions met
  - eg presence/absence of some file
  - particular date/time
  - particular user
- causes damage when triggered
  - modify/delete files/disks, halt machine, etc

Trojan Horse

- program with hidden side-effects
- program is usually superficially attractive
  - eg game, s/w upgrade etc
- performs additional tasks when executed
  - allows attacker to indirectly gain access they do not have directly
- often used to propagate a virus/worm or to install a backdoor
- … or simply to destroy data

Mobile Code

- program/script/macro that runs unchanged
  - on heterogeneous collection of platforms
  - on large homogeneous collection (Windows)
- transmitted from remote system to local system & then executed on local system
- often to inject Trojan horse, spyware, virus, worm,
- or to perform own exploits
  - unauthorized data access, root compromise

Multiple-Threat Malware

- Malware may operate in multiple ways
- **Multipartite** virus infects in multiple ways
  - eg. multiple file types
- **Blended** attack uses multiple methods of infection or transmission
  - to maximize speed of contagion and severity
  - may include multiple types of malware
  - eg. Nimda has worm, virus, mobile code
  - can also use IM & P2P

Viruses

- piece of software that infects programs
  - modifying programs to include a copy of the virus
  - so it executes secretly when host program is run
- specific to operating system and hardware
  - taking advantage of their details and weaknesses
- a typical virus goes through phases of:
  - dormant
  - propagation
  - triggering
  - execution

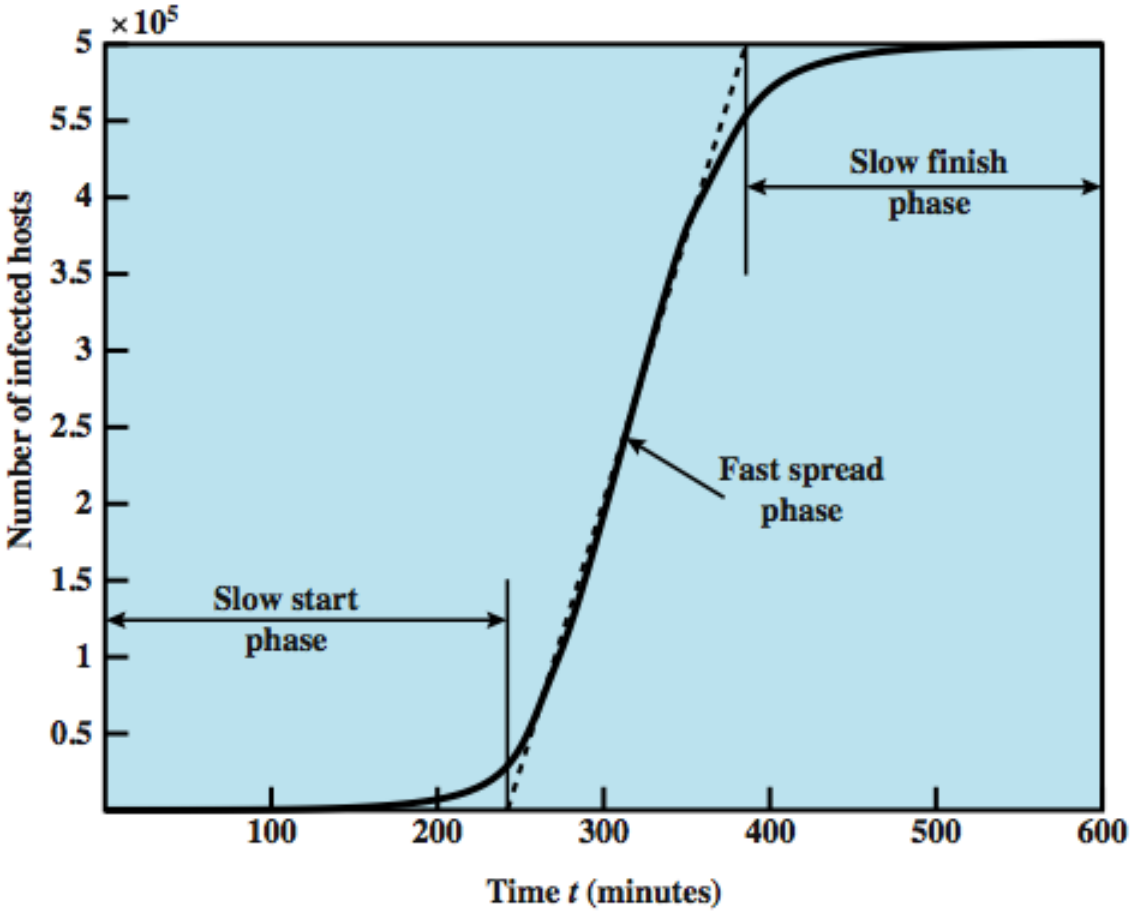
Some virus types

- Boot sector virus
- File infector virus
- Macro virus
- Encrypted virus
- Stealth virus
  - Uses techniques to hide itself
- Polymorphic virus
  - Different for every system
- Metamorphic virus
  - Different after every activation on same system

Worms

- Replicating program that propagates over net
  - using email, remote exec, remote login
- Has phases like a virus:
  - dormant, propagation, triggering, execution
  - propagation phase: searches for other systems, connects to it, copies self to it and runs
- May disguise itself as a system process
- Morris Worm, one of best know worms
  - released by Robert Morris in 1988
  - exploited vulnerabilities in UNIX systems
  - brought the whole Internet (of 1988) to standstill

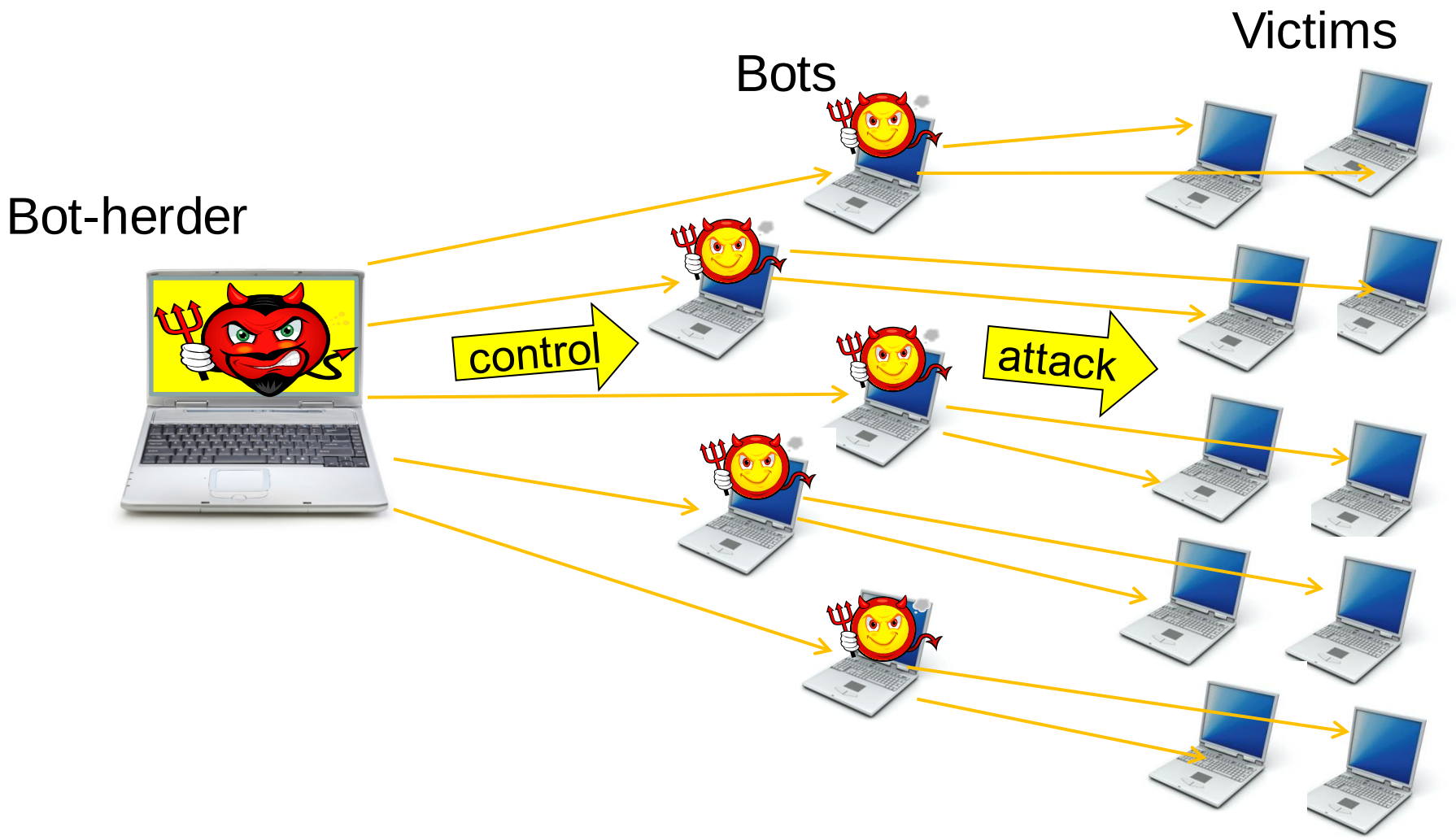
Worm Propagation Speed



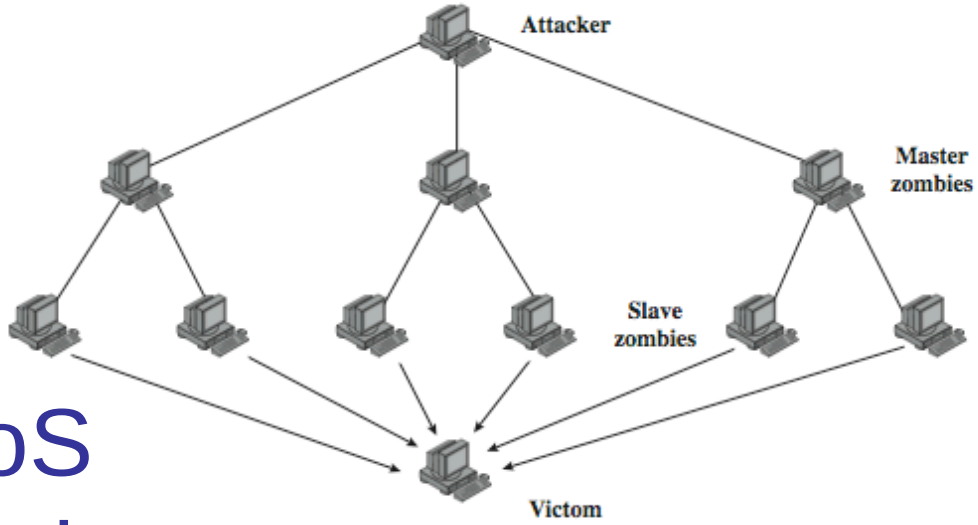
What is a botnet ?

- **A botnet** is a collection of computers infected with malicious software agents (robots) that can be controlled remotely by an attacker.
- Owners of bot computers are typically unaware of infection.
- Botnet controller is called a "bot herder" or "bot master"
- Botnets execute malicious functions in a coordinated way:
  - Send spam email
  - Collect identity information
  - Denial of service attacks
  - Create more bots
- A botnet is typically named after the malware used to infect
- Multiple botnets can use the same malware, but can still be operated by different criminal groups

Botnet Architecture

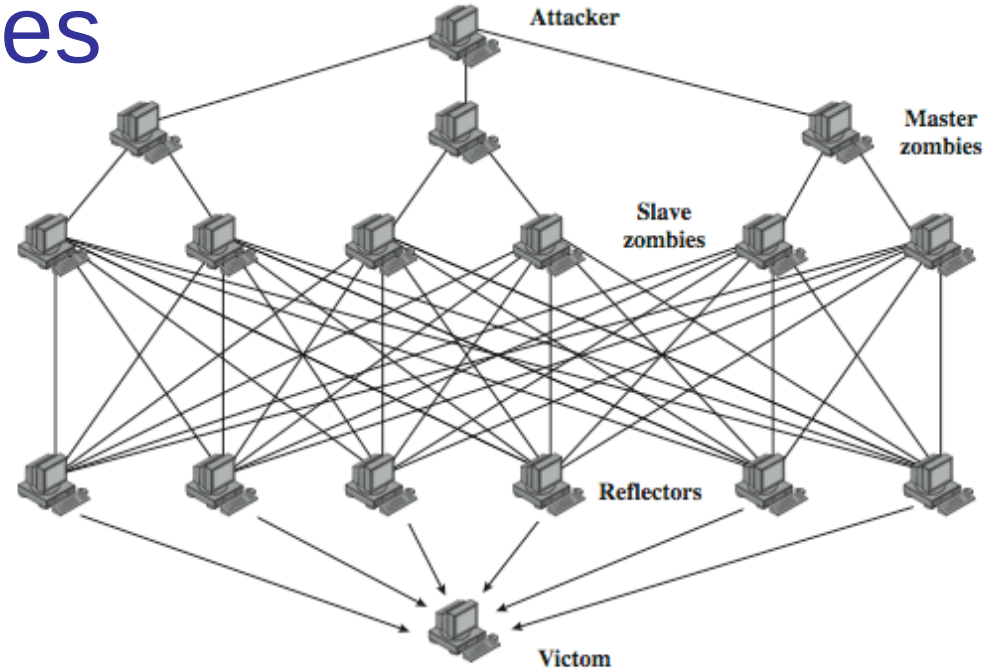


DDoS Flood Types



(a) Direct DDoS Attack

- Direct attack
  - Bots send traffic with own or spoofed sender address to victim



(b) Reflector DDoS Attack

- Reflected attack
  - Bots send traffic to innocent hosts with victim address as sender address. Innicent host become part of attack by replying to victim.

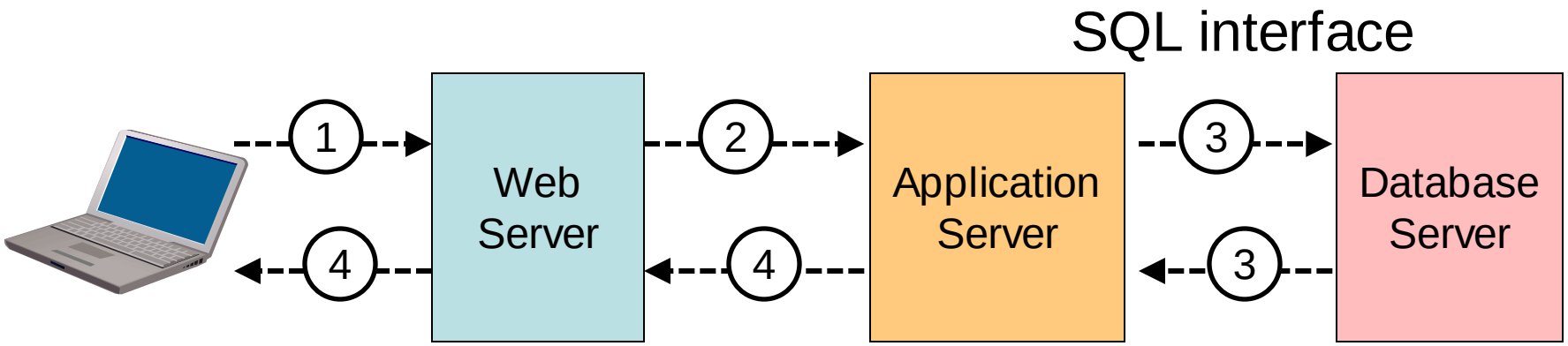
What is SQL?

- Structured Query Language: interface to relational database systems.
- Allows for insert, update, delete, and retrieval of data in a database.
- ANSI, ISO Standard, used extensively in web applications.
- Example:

```
select ProductName from products where
ProductID = 40;
```

SQL at back-end of websites

1. Take input from a web-form via HTTP methods such as POST or GET, and pass it to a server-side application.
2. Application process opens connection to SQL database.
3. Query database with SQL and retrieve reply.
4. Process SQL reply and send results back to user.



What is SQL Injection?

- Misinterpretation of data input to database system
  - Attacker disguises SQL commands as data-input
  - Disguised SQL commands = 'injected' SQL commands
- With SQL injection, an attacker can get complete control of database
  - no matter how well the system is patched,
  - no matter how well the firewall is configured,
- Vulnerability exists when web application fails to sanitize data input before sending to it database
- Flaw is in web application, not in SQL database.

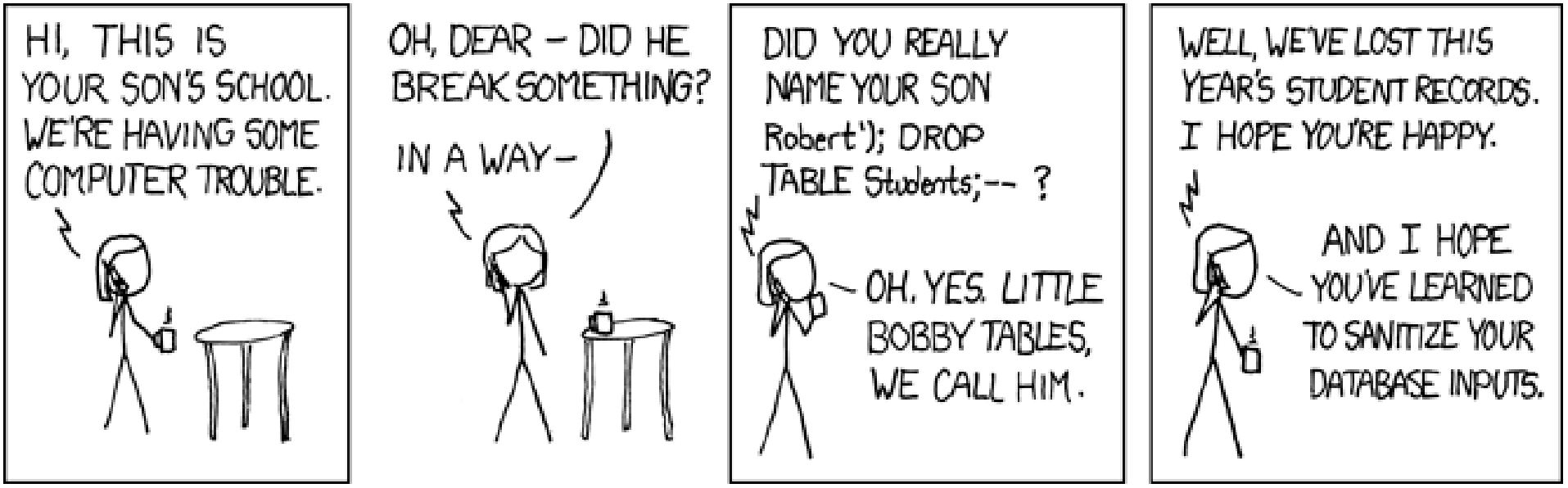
What is SQL Injection?

- For example, if user input is "**40 or 1 = 1**"

```
select ProductName from products where
  ProductID = 40 or 1 = 1
```

- 1=1 is always TRUE so the "where" clause will always be satisfied, even if ProductID ≠ 40.
- All product records will be returned.
- Data leak.

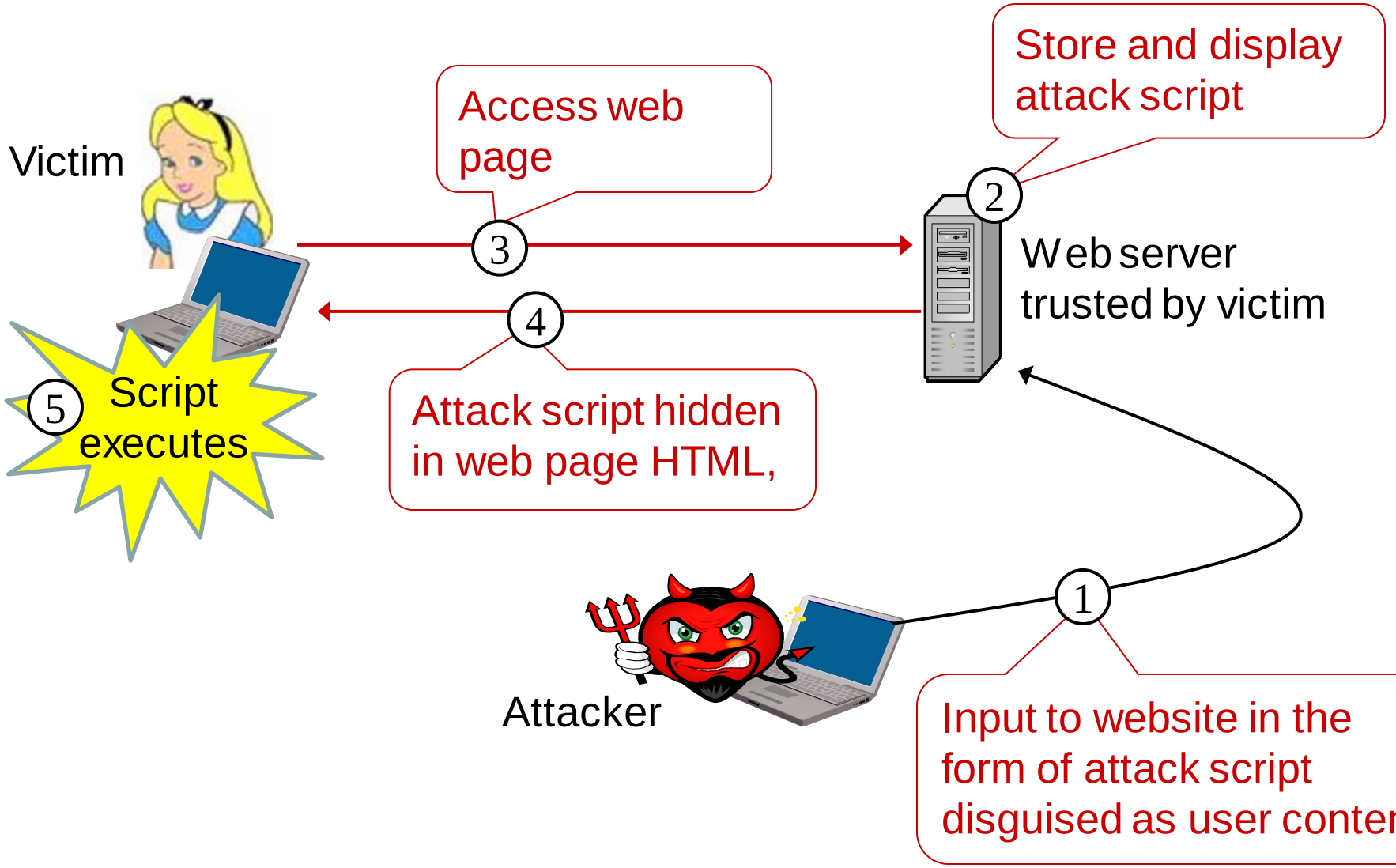
XKCD – Little Bobby tables



Prevention of SQL Injection

- **Check and filter user input.**
  - Length limit on input (most attacks depend on long query strings).
  - Different types of inputs have a specific language and syntax associated with them, i.e. name, email, etc
  - Do not allow suspicious keywords (DROP, INSERT, SELECT, SHUTDOWN) as name for example.
  - Try to bind variables to specific types.

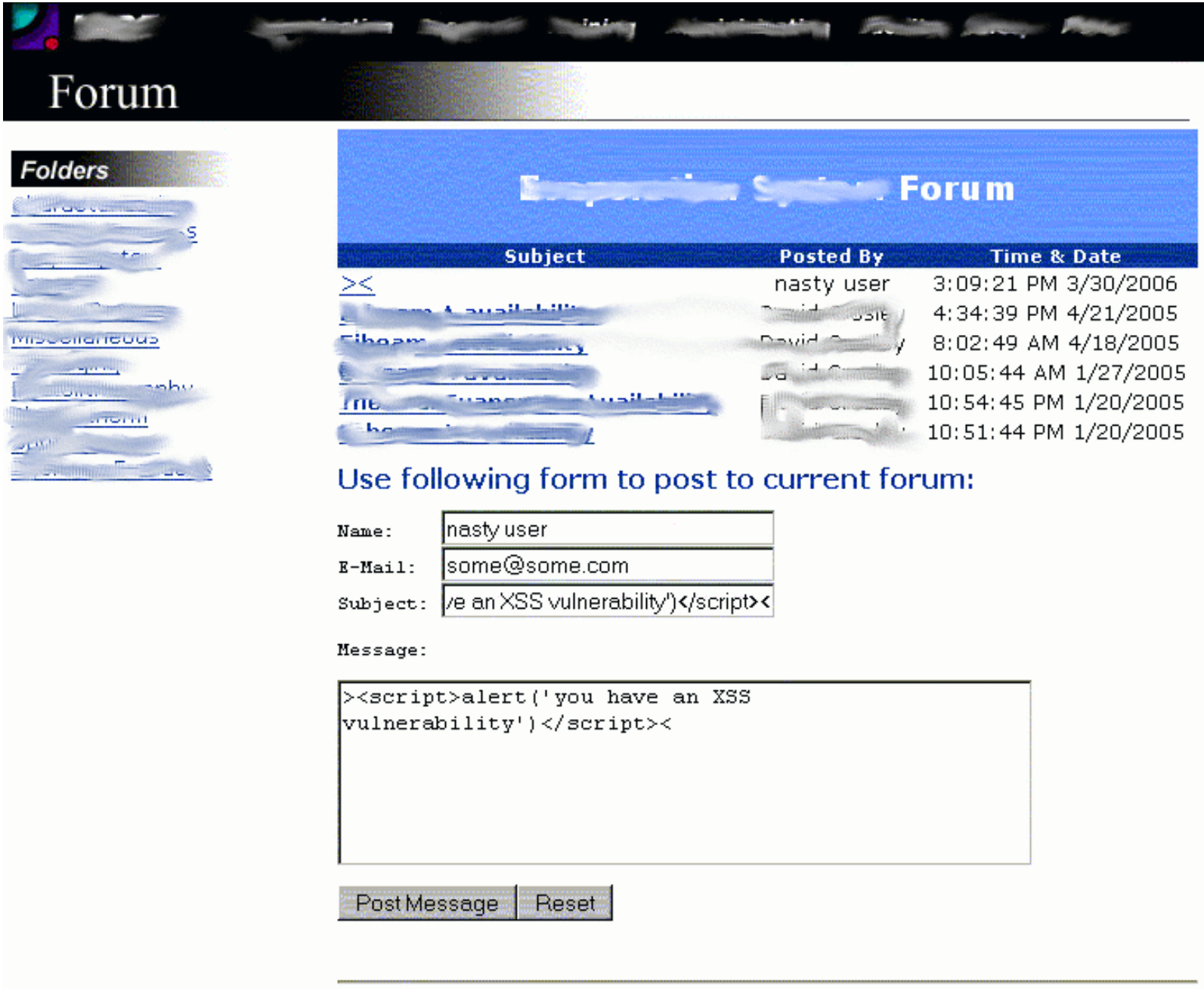
Stored XSS


Victim
Access web page
Store and display attack script
2
3
Web server trusted by victim
4
Attack script hidden in web page HTML,
5
Script executes
1
Attacker
Input to website in the form of attack script disguised as user content

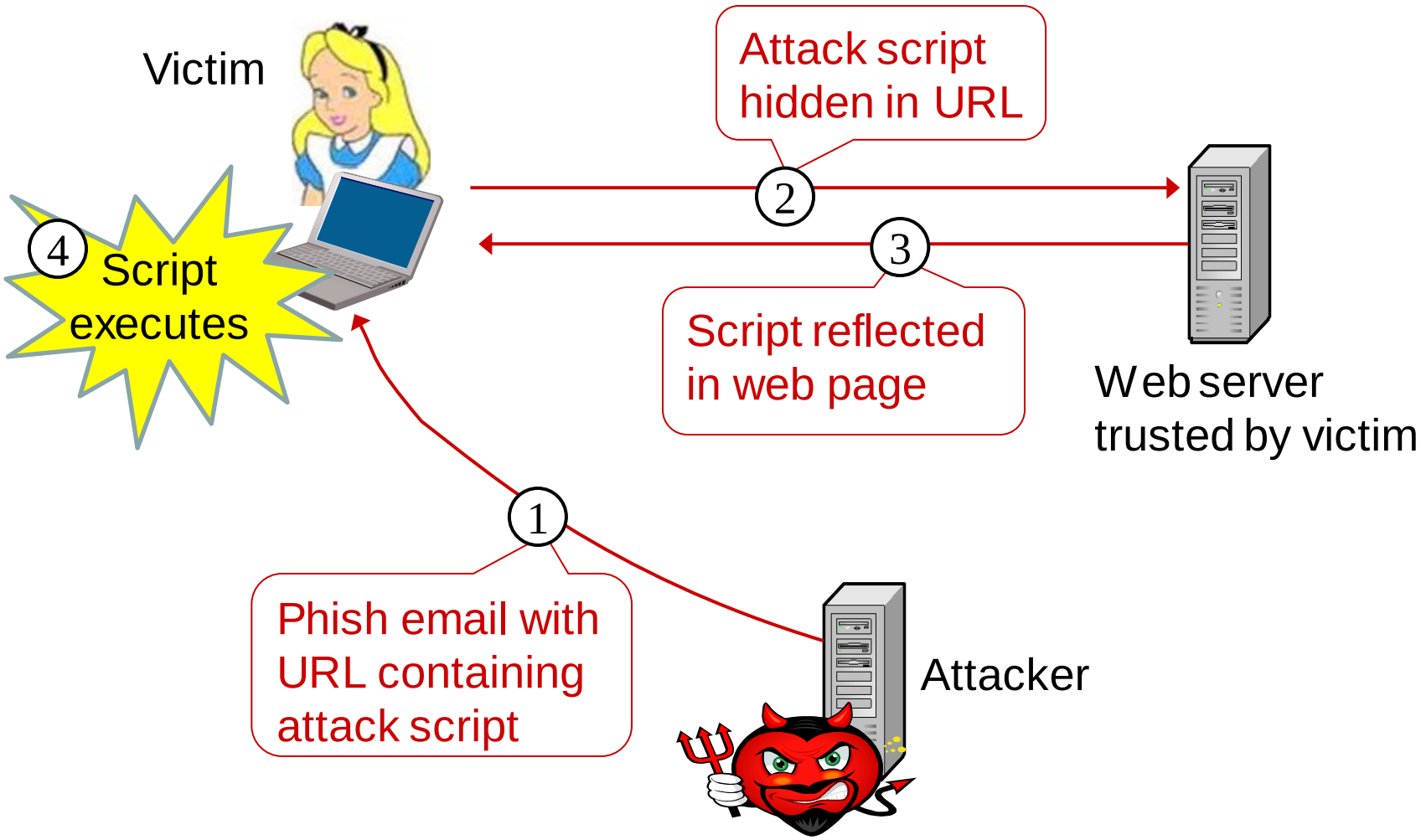
Stored XSS

- Stored, persistent, or second-order XSS.
- Data provided by users to a web application is stored persistently on server (in database, file system, …) and later displayed to users in a web page.
- Typical example: online message boards.
- Attacker uploads data containing malicious script to server.
- Every time the vulnerable web page is visited, the malicious script gets executed in client browser.
- Attacker needs to inject script just once.

XSS: Script Injection Demo



Reflected XSS



Reflected XSS

- Data provided by client is used by server-side scripts to generate results page for user.
- User tricked to click on attacker's link for attack to be launched; page contains a frame that requests page from server with script as query parameter.
- If unvalidated user data is echoed in results page (without HTML encoding), code can be injected into this page.
- Typically delivered via email, containing an innocently looking URL that contains a script.
  - E.g., search engine redisplays search string on the result page; in a search for a string that includes some HTML special characters code may be injected.

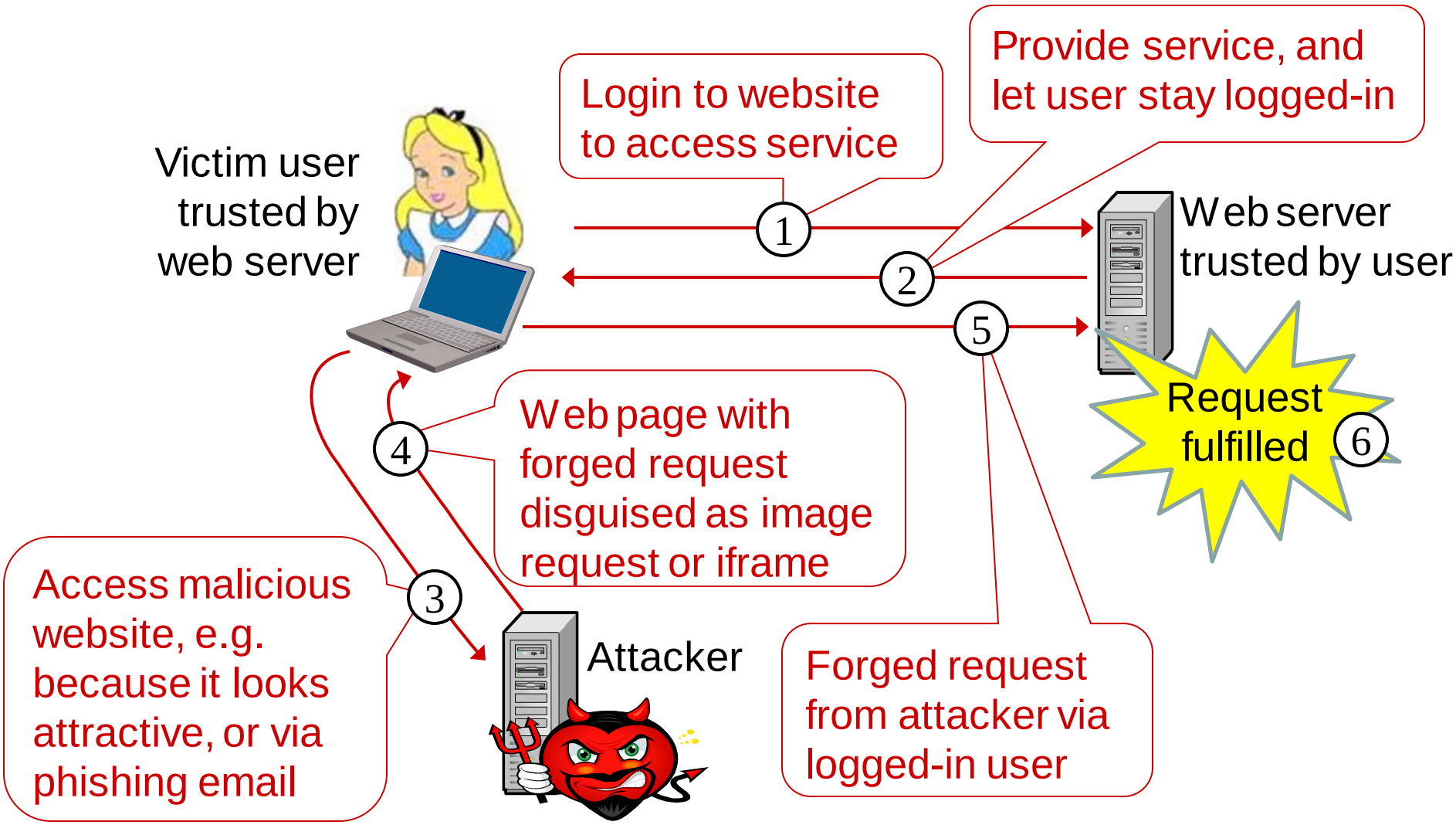
XSS – The Problem

- Ultimate cause of the attack: The client only authenticates 'the last hop' of the entire page, but not the true origin of all parts of the page.
- For example, the browser authenticates the bulletin board service but not the user who had placed a particular entry.
- If the browser cannot authenticate the origin of all its inputs, it cannot enforce a code origin policy.

Preventing SQL injection and XSS

- Hide information about Error handling
  - Error messages divulge information that can be used by hacker
  - Error messages must not reveal potentially sensitive information
- Validate all user entered parameters
  - CHECK data types and lengths
  - DISALLOW unwanted data (e.g. HTML tags, JavaScript)
  - ESCAPE questionable characters (ticks, --,semi-colon, brackets, etc.)

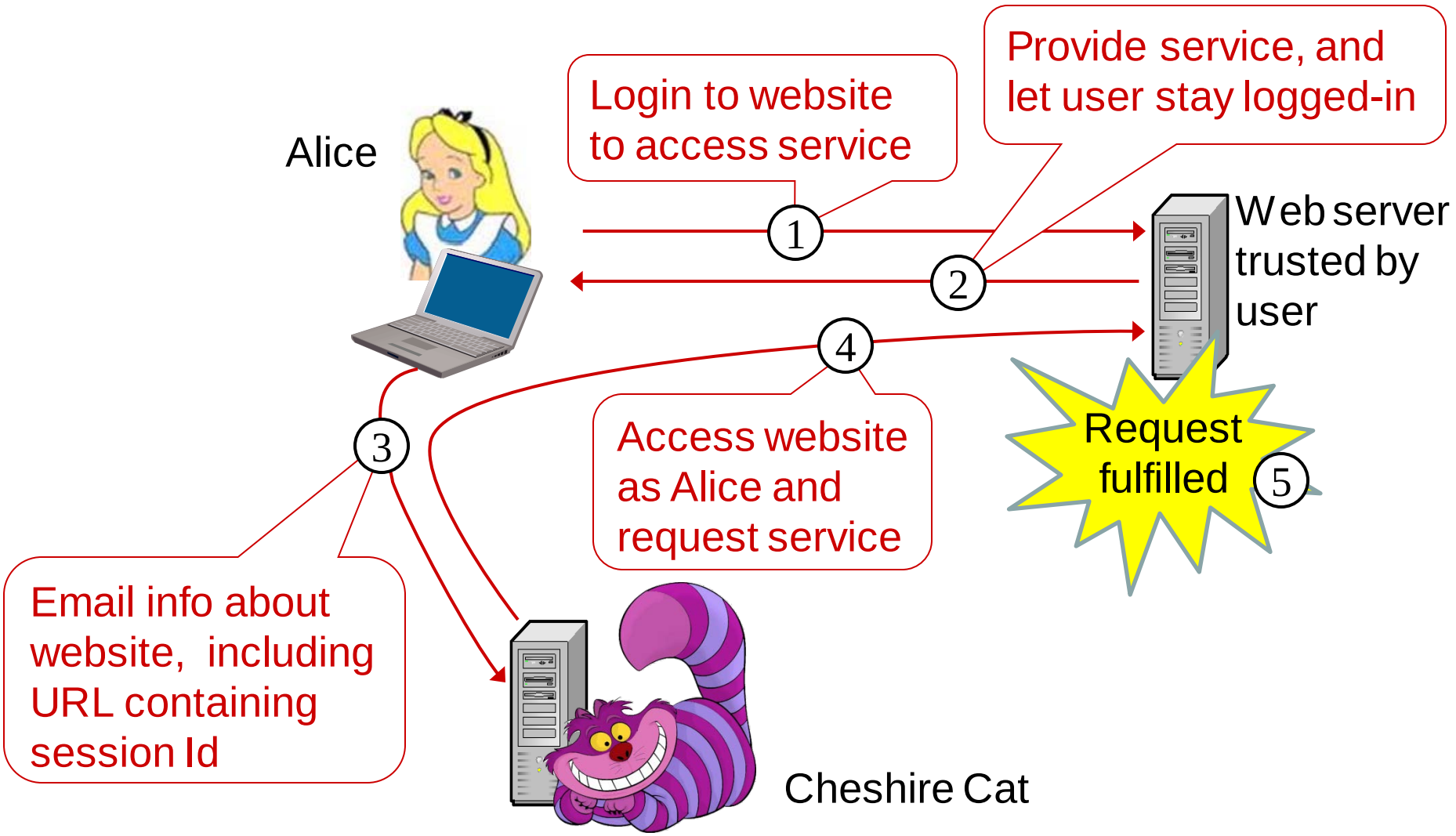
CSRF (Cross-Site Request Forgery)



CSRF – Problem and Fix

- Users stay logged-in at websites even when not using them
  - Can be exploited by attackers sending fake requests via users
- Forged HTTP requests for a specific website that requires user login are hidden on attacker's webpage in the form of fake image requests, iframes or other elements.
- Browser accesses webpage and forwards forged requests.
- Preventing CSRF usually requires the inclusion of an unpredictable reference token (e.g. a random number) with each HTTP request to websites requiring login. Request tokens should at a minimum be unique per user session.
- Because the request token is unpredictable, the attacker is unable to create a forged request that will be accepted and fulfilled by the web server.

Broken Authentication and Session Mgmt

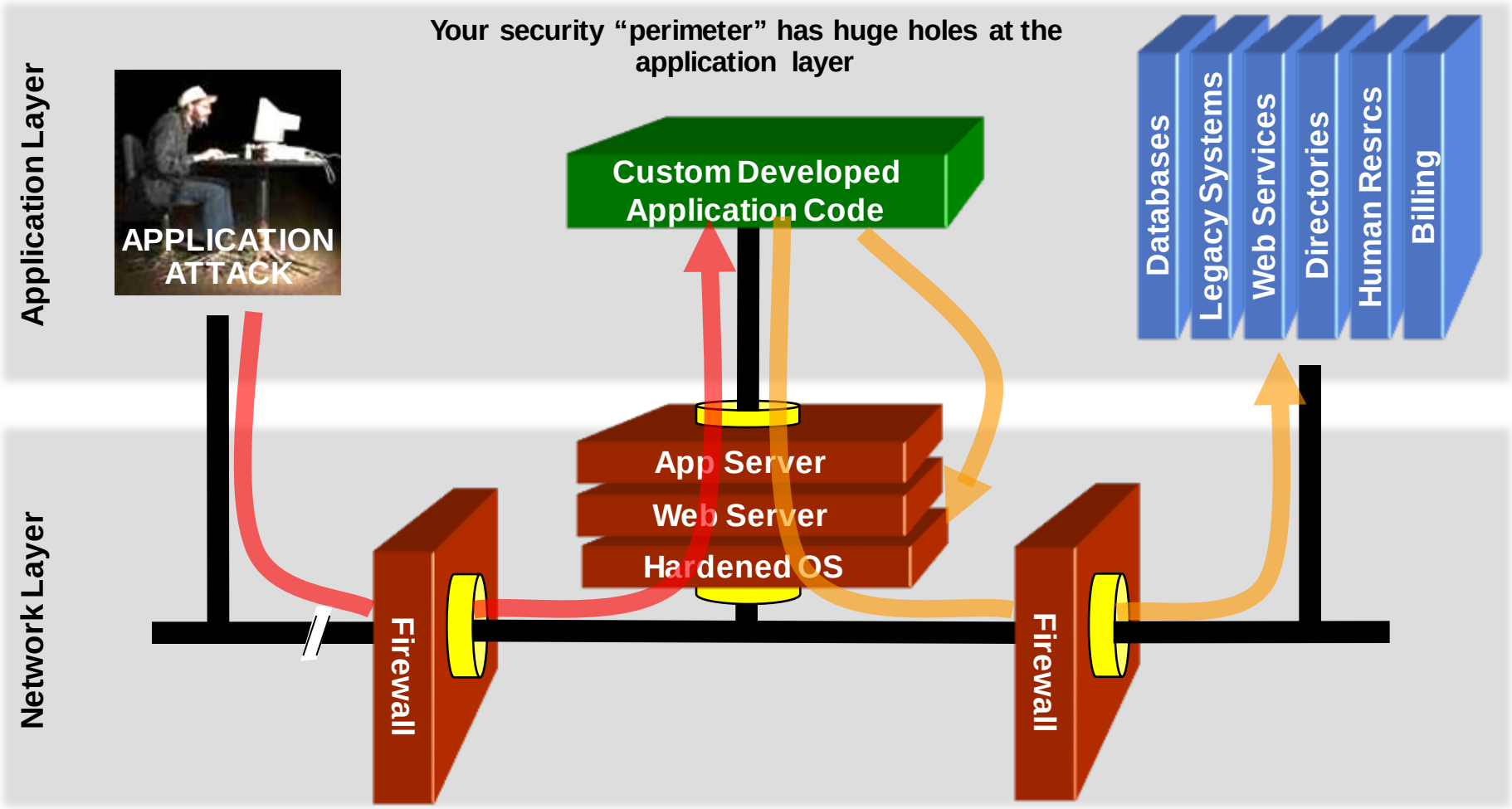


Broken Authentication and Session Mgmnt Problem and Fix

- User authentication does not necessarily provide continuous authentication assurance
  - User authentication is only at one point in time
- Easy for developers to implement session control with a simple session Id which is passed in the URL
  - Unfortunately this can be misused
- Recommendations for session Id must be followed
  - E.g friom OWASP
- Examples of controls for session Id:
  - Link session Id to e.g. IP address, TLS session Id
- .

Software Development Security

The web application security challenge



Network security (firewall, SSL, IDS, hardening) does not stop application attacks

OWASP
The Open Web Application Security Project

- Non-profit organisation
  - Local chapters in most countries, also in Norway
- OWASP promotes security awareness and security solutions for Web application development.
- OWASP Top-10 security risks identify the most critical security risks of providing online services
  - The Top 10 list also recommends relevant security solutions.
- OWASP ASVS (Application Security Verification Standard) specifies requirements for application-level security.
- Provides and maintains many free tools for scanning and security vulnerability fixing

Top-10 Web Application Risks

1. Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery (CSRF)
9. Using Components with Known Vulnerabilities
10. Unvalidated Redirects and Forwards

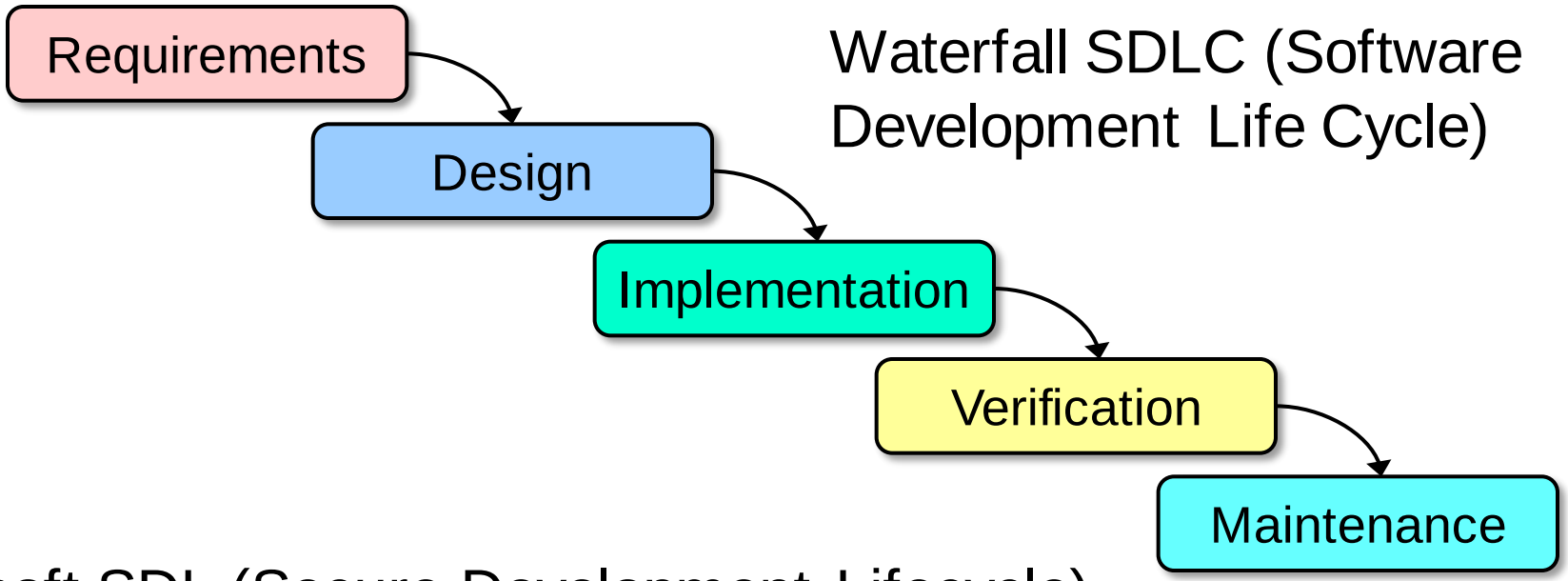
SDLC: Software Development Life Cycle

- SDLC model contains 5 basic stages:
  1. Requirements Specification
  2. Design
  3. Implementation
  4. Verification and Testing
  5. Deployment and Maintenance
- Each SDLC model organises/integrates these basic stages in a specific way
  - Waterfall
  - Agile (XP: Extreme Programming).
  - Iteration model
  - etc…

Secure SDLC

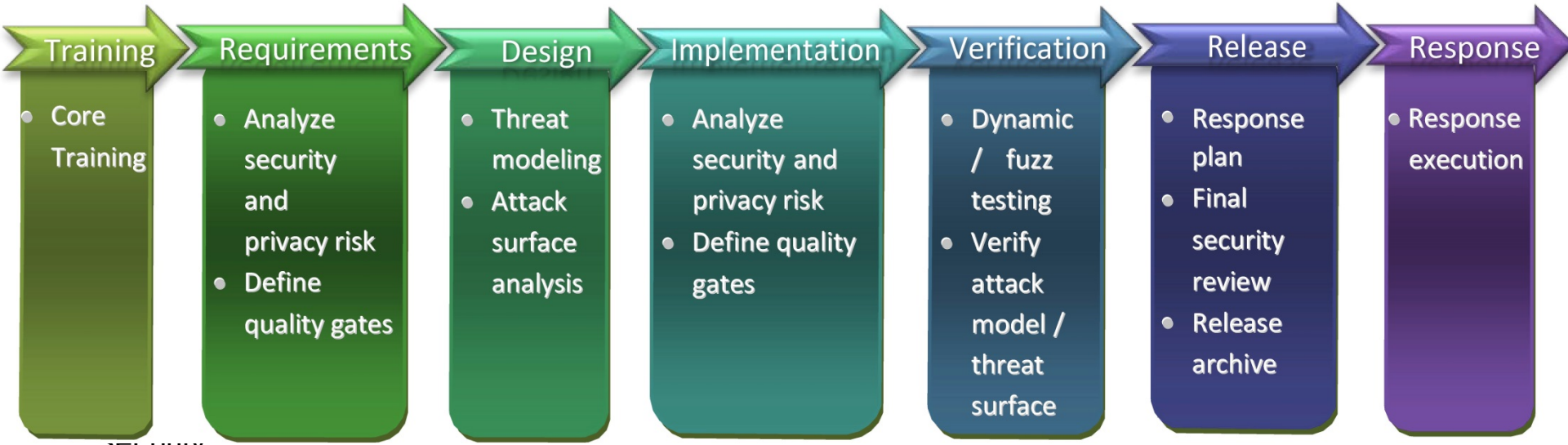
- Secure Software Development Life Cycle
  - Used along with traditional/current software development lifecycle methods in order to ensure that security is considered during the SDLC.
- Three essential elements of secure SDLC
  1. Include security related tasks in each stage of the SDLC
  2. Security education for system engineers
  3. Metrics and accountability to assess security of system

Waterfall and Secure Waterfall



Waterfall SDLC (Software Development Life Cycle)

Microsoft SDL (Secure Development Lifecycle)



Security

SDL Security Training

- New employees typically do not arrive with ability to develop secure software
  - Security training as a part of New Employee Orientation
  - Specialised security training for technical staff
  - Update and fresh up security skills annually
- Universities without adequate IT-security training are part of the problem of software insecurity
  - What about UiO ?
  - IFI has weak focus on IT-security training
    - No mandatory courses in security
    - No practical security development training
    - No pen-testing training
    - No digital forensics training
  - UiO must become part of the solution !

SDL Requirements Phase

- Opportunity to consider security at the outset
- Consider having Security Buddy for development projects
- Development team identifies security requirements
- Security Buddy reviews product plan, makes recommendations, ensures adequate security resources
- Security Buddy assesses security milestones and exit criteria

- The Security Buddy stays with the project through the Final Security Review

SDL Design

- Design stage
  - Define and document security architecture
  - Identify security critical components (“trusted base”)
  - Identify design techniques (e.g., layering, managed code, least privilege, attack surface minimization)
  - Document attack surface and limit through default settings
  - Create threat models (e.g., identify assets, interfaces, threats, risk) and mitigate threats through countermeasures
  - Identify specialized test tools
  - Define supplemental ship criteria due to unique product issues (e.g., cross-site scripting tests)
  - Confer with Security Buddy on questions
- Exit criteria: Design review complete and signed off by development team *and* Security Buddy

SDL Development

- Apply coding and testing standards (e.g., safe string handling)
- Apply fuzz testing tools (structured invalid inputs to network protocol and file parsers)
- Apply static code analysis tools (to find, e.g., buffer overruns, integer overruns, uninitialized variables)
- Conduct code reviews

SDL Verification

- Software functionality complete and enters Beta
- Test both new and possible legacy code
- Security push:
  - Provides an opportunity to focus on security
    - Code reviews (especially legacy/unchanged code)
    - Penetration and other security testing
    - Review design, architecture, threat models in light of new threats
  - Security push is not a substitute for security work during development

SDL Release

- Final Security Review (FSR)
  - Additional penetration testing, possibly by outside contractors to supplement security team
  - Analysis of any newly reported vulnerabilities affecting libraries used
  - FSR results: If the FSR finds a pattern of remaining vulnerabilities, the proper response is not just to fix the vulnerabilities found, but to revisit the earlier phases and take pointed actions to address root causes (e.g., improve training, enhance tools)
- Make security response plan

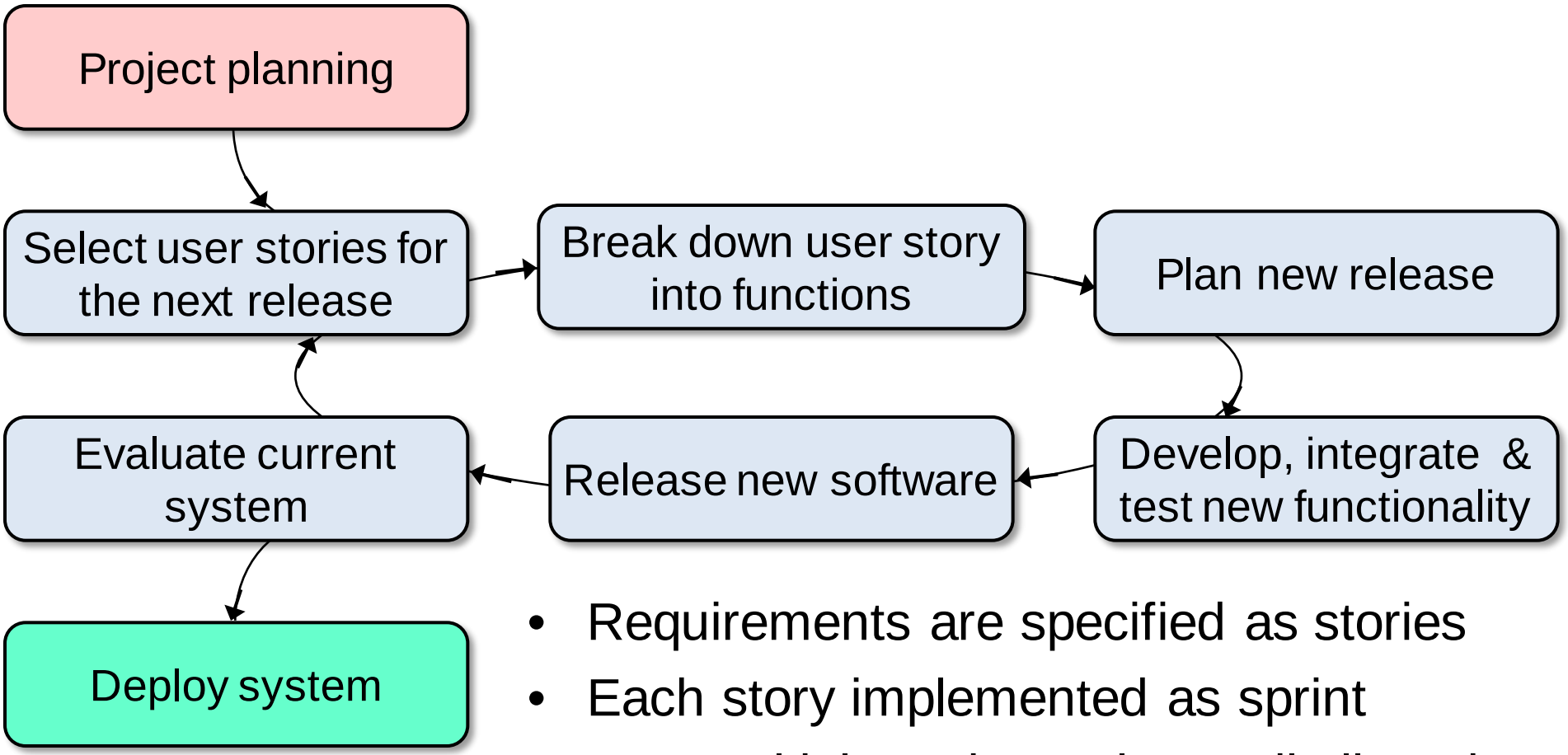
SDL Response Phase

- Sustained engineering teams for security
- Patch management
- Post mortems and feedback to the SDL

Fuzzing

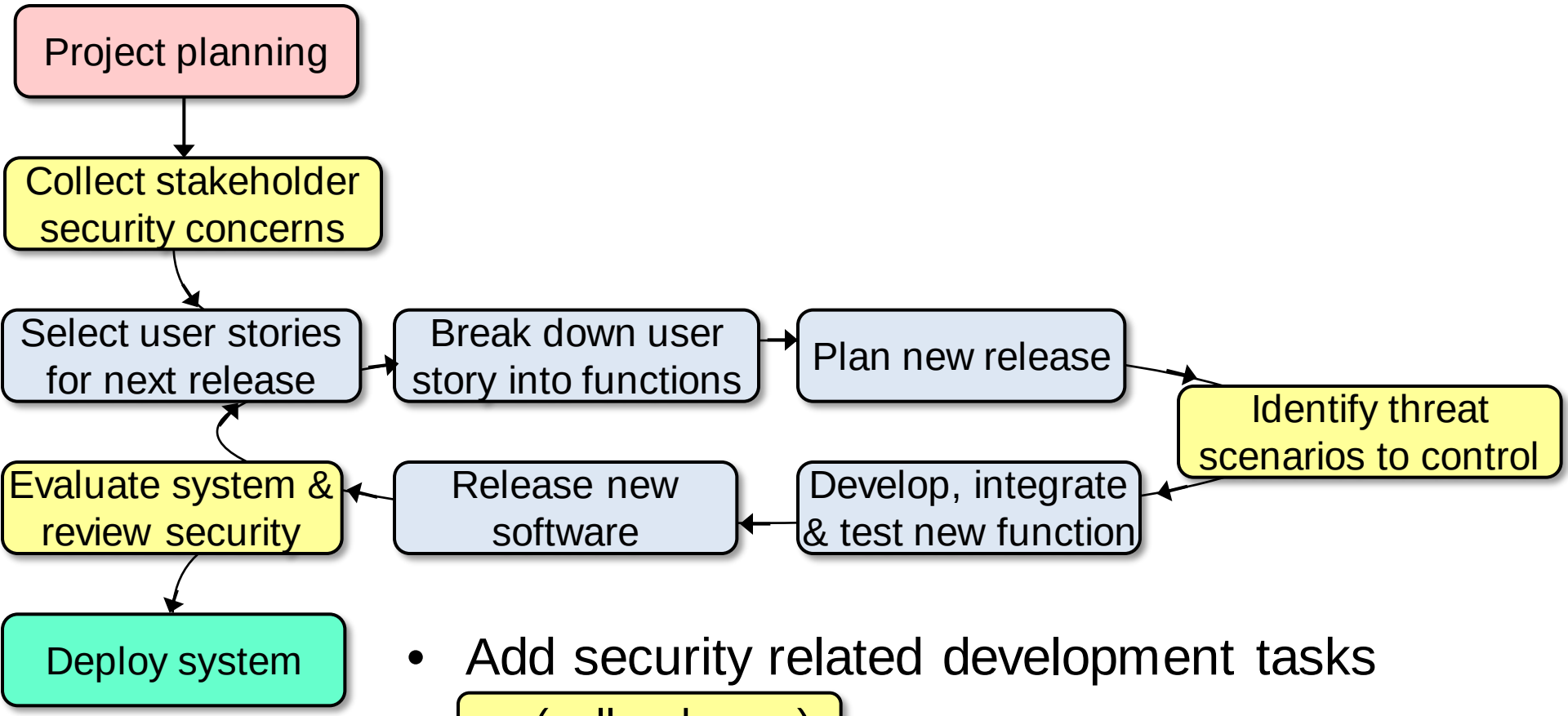
- Malformed input should be handled in a consistent way by software and systems
  - Should be rejected with/without appropriate error message
- … but malformed input often leads to system crash due to software bugs
- Fuzzing is to generate many forms of malformed input and then to analyse resulting system crashes
  - The software location of a crash points to the location of the bug
- Some crashes can be exploited by attackers
  - Then the bug is a security vulnerability
- Developers and attackers use fuzzing to find vulnerabilities
- Infinitely many different malformed inputs
  - Impossible to test them all $\Rightarrow$ impossible to find all vulnerabilities

Agile Software Development



- Requirements are specified as stories
- Each story implemented as sprint
- Run multiple sprint cycles until all stories are implemented

Secure Agile Software Development

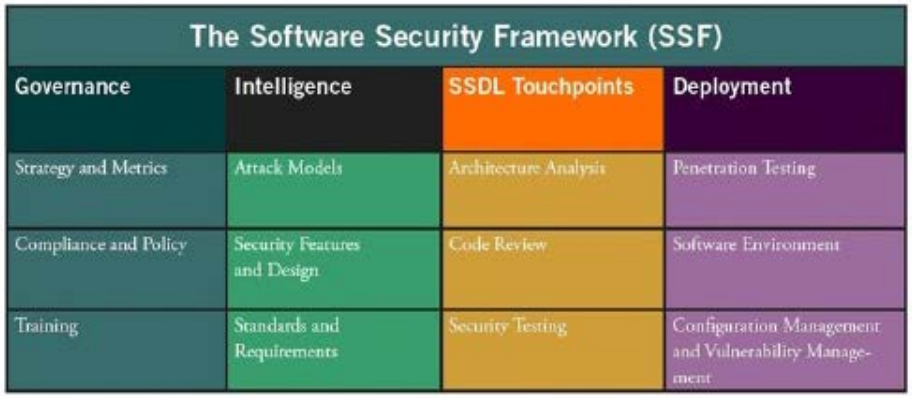


- Add security related development tasks
  - (yellow boxes)
- Secure agile SDLC is challenging
- Security necessarily makes SDLC less agile

Software Security Maturity Models

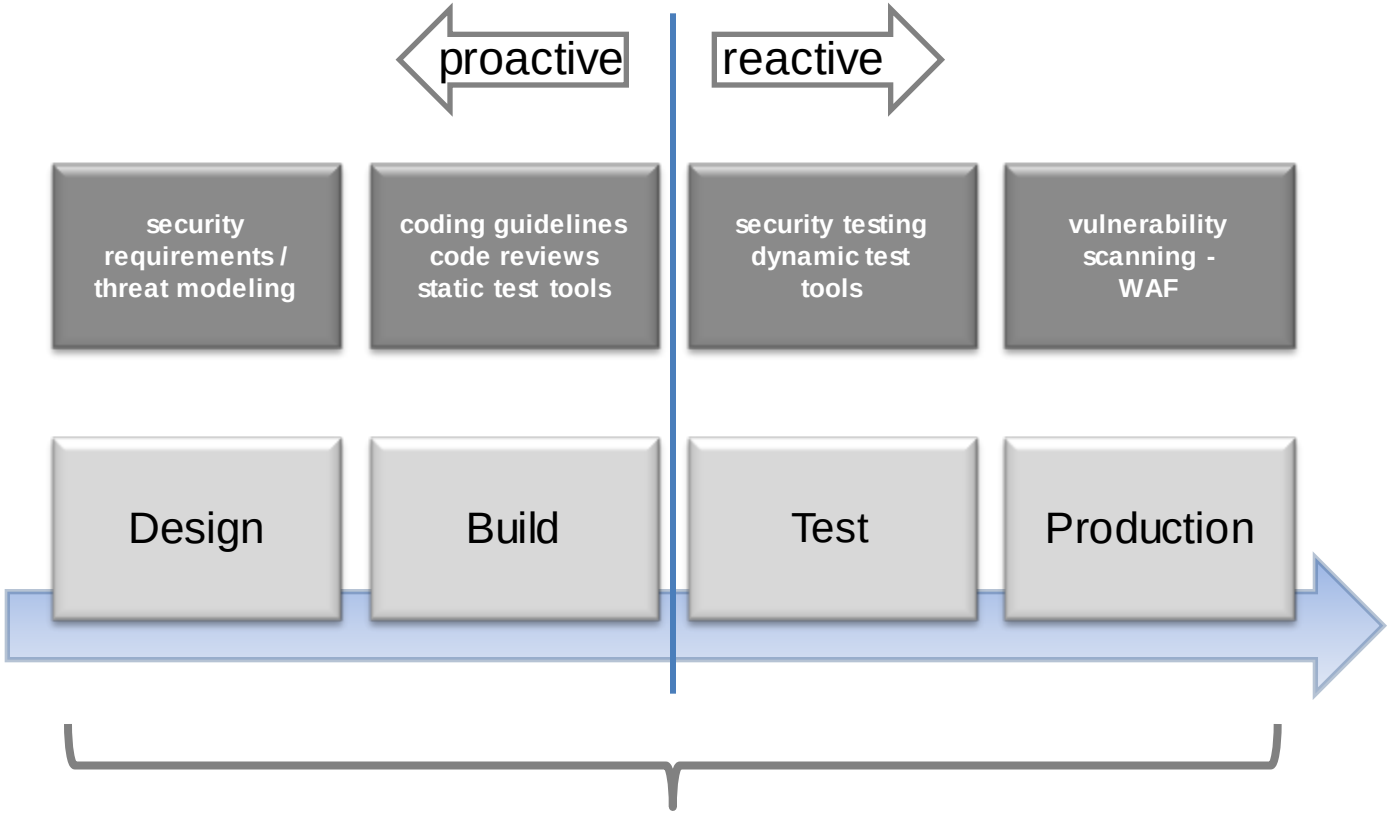
- OpenSAMM
  - Software Assurance Maturity Model
  - Promoted by OWASP

- BSIMM
  - Build Security In Maturity Model



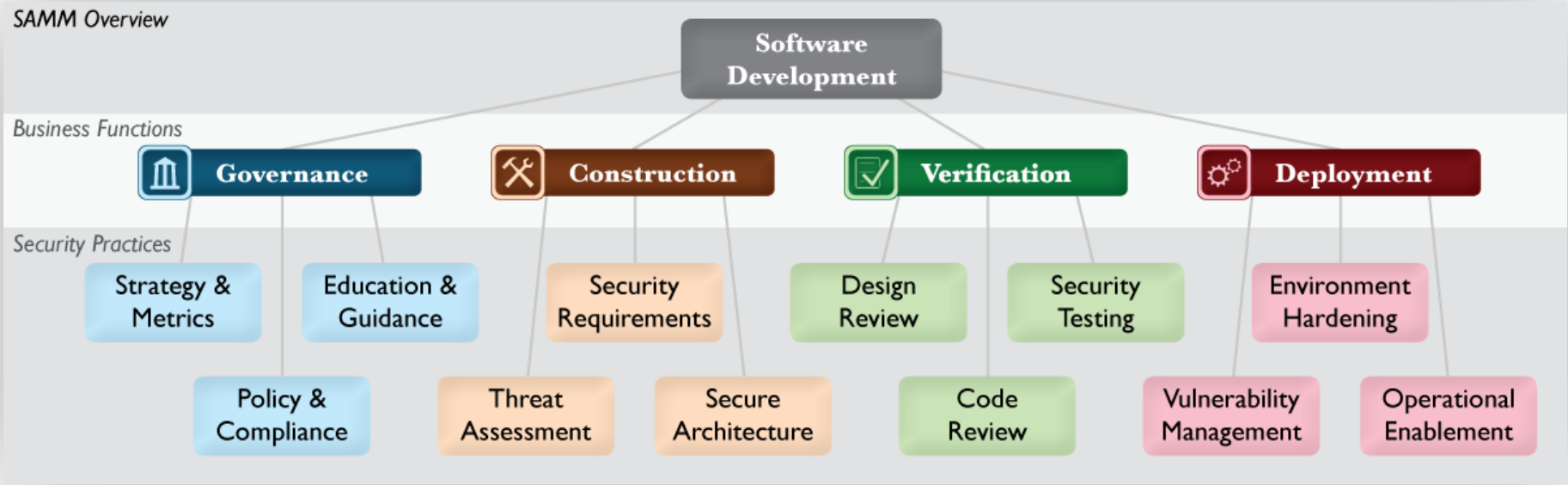
OWASP 16

Open SAMM
Software Assurance Maturity Model



SAMM Security Practices

- From each of the Business Functions, 3 Security Practices are defined
- The Security Practices cover all areas relevant to software security assurance
- Each one is a ‘silo’ for improvement



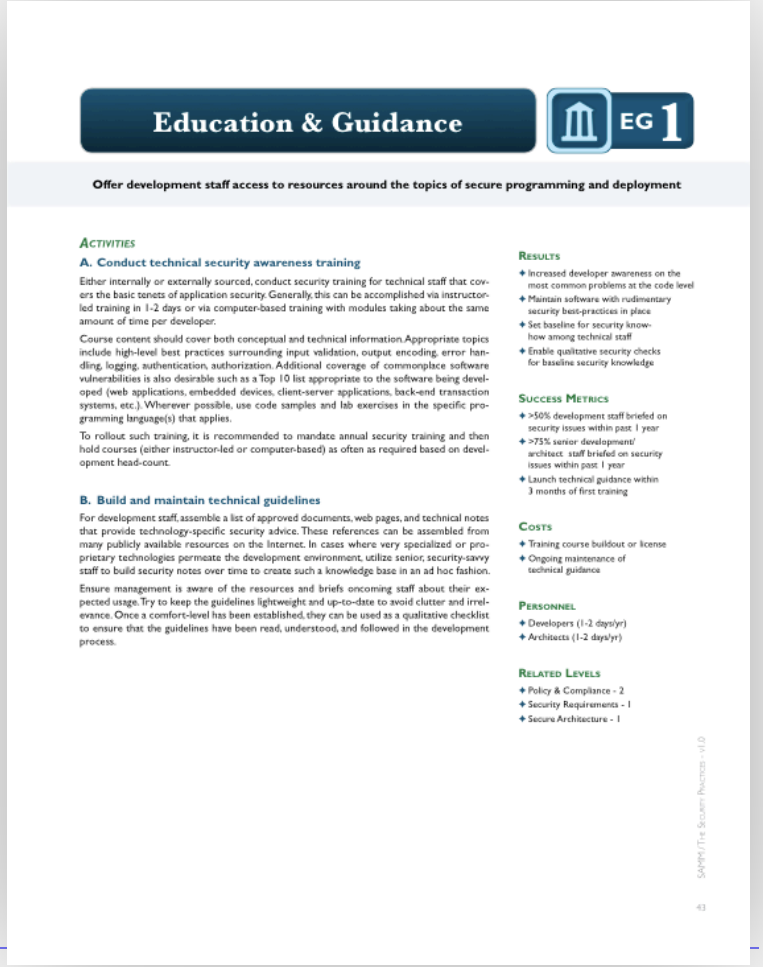
Under each Security Practice

- Three successive Objectives under each Practice define how it can be improved over time
  - This establishes a notion of a Level at which an organization fulfills a given Practice

- The three Levels for a Practice generally correspond to:
  - (0: Implicit starting point with the Practice unfulfilled)
  - 1: Initial understanding and ad hoc provision of the Practice
  - 2: Increase efficiency and/or effectiveness of the Practice
  - 3: Comprehensive mastery of the Practice at scale

Per Level, SAMM defines...

- Objective
- Activities
- Results
- Success Metrics
- Costs
- Personnel
- Related Levels



Education & Guidance

EG 1

Offer development staff access to resources around the topics of secure programming and deployment

Activities

A. Conduct technical security awareness training

Either internally or externally sourced, conduct security training for technical staff that covers the basic tenets of application security. Generally, this can be accomplished via instructor-led training in 1-2 days or via computer-based training with modules taking about the same amount of time per developer.

Course content should cover both conceptual and technical information. Appropriate topics include high-level best practices surrounding input validation, output encoding, error handling, logging, authentication, authorization. Additional coverage of commonplace software vulnerabilities is also desirable such as a Top 10 list appropriate to the software being developed (web applications, embedded devices, client-server applications, back-end transaction systems, etc.). Wherever possible, use code samples and lab exercises in the specific programming language(s) that applies.

To rollout such training, it is recommended to mandate annual security training and then hold courses (either instructor-led or computer-based) as often as required based on development head-count.

B. Build and maintain technical guidelines

For development staff, assemble a list of approved documents, web pages, and technical notes that provide technology-specific security advice. These references can be assembled from many publicly available resources on the Internet. In cases where very specialized or proprietary technologies permeate the development environment, utilize senior, security-savvy staff to build security notes over time to create such a knowledge base in an ad hoc fashion.

Ensure management is aware of the resources and briefs oncoming staff about their expected usage. Try to keep the guidelines lightweight and up-to-date to avoid clutter and irrelevance. Once a comfort-level has been established, they can be used as a qualitative checklist to ensure that the guidelines have been read, understood, and followed in the development process.

Results

- Increased developer awareness on the most common problems at the code level
- Maintain software with rudimentary security best-practices in place
- Set baseline for security know-how among technical staff
- Enable qualitative security checks for baseline security knowledge

Success Metrics

- >50% development staff briefed on security issues within past 1 year
- >75% senior development/ architect staff briefed on security issues within past 1 year
- Launch technical guidance within 3 months of first training

Costs

- Training course buildout or license
- Ongoing maintenance of technical guidance

Personnel

- Developers (1-2 days/yr)
- Architects (1-2 days/yr)

Related Levels

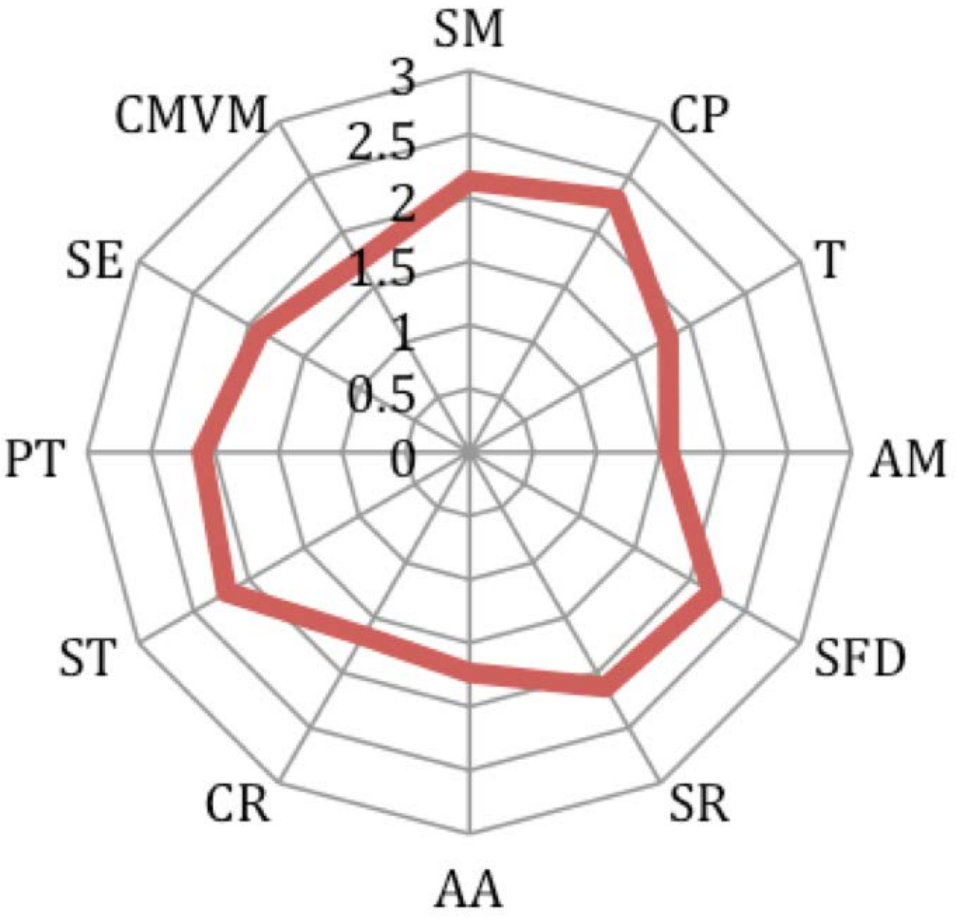
- Policy & Compliance - 2
- Security Requirements - 1
- Secure Architecture - 1

BSIMM SSF

The Software Security Framework (SSF)			
Governance	**Intelligence**	**SSDL Touchpoints**	**Deployment**
Strategy and Metrics	Attack Models	Architecture Analysis	Penetration Testing
Compliance and Policy	Security Features and Design	Code Review	Software Environment
Training	Standards and Requirements	Security Testing	Configuration Management and Vulnerability Management

- 4 main domains.
  i) Governance, ii) Intelligence, iii) SSDL, iv) Deployment.
- 12 separate practices (3 per domain)
- 112 activities spread over the 12 practices

BSIMM Radar Map Maturity Chart



- 12 practices
- Score for each practice
- 4 levels
- ↑ size ⇒ ↑ maturity

Windows 10 Security

- Next and last lecture
- Monday 27 April 2015
- **Time 10:30h – 12:00h**
- Guest lecturer :Ole Tom Seierstad (Microsoft)

Try to addend.

Will be interesting !

End of Lecture