Software Security Course

An Introduction to Software Security

Dimitrios A. Glynos
{ *daglyn at unipi.gr* }

Department of Informatics
University of Piraeus

Part I

What is software security?

Software Security

- Software Security: A set of practices that help protect a software's ***assets*** from attackers
- What can we consider as ***assets*** in a piece of software?
  - The software's functionality
  - The software's proprietary algorithms and data (e.g. training set)
  - The software's system data
  - The software's user data
  - The software's development environment
  - The software's supply chain
  - ...
- Sometimes an asset is not under the complete control of the vendor (e.g. AWS shared responsibility model[1] for deployment environment)

[1] https://aws.amazon.com/compliance/shared-responsibility-model/

Where does security stand in the world of software?

- Secure Software $\rightarrow$ Quality Software $\rightarrow$ Dependable Software
- Security may bring features to a software project (privacy, accountability, attestation, etc.)
- Software security practices are quickly becoming a **formal requirement** in all types of software development projects (i.e. not just the critical ones) [2] [3] [4] [5]

[2] Automotive ISO-SAE-21434 cybersecurity requirements
[3] Medical Device Pre-market cybersecurity requirements
[4] Directive for USA government contractors
[5] EU Radio Equipment Directive

The C I A triad

Since Software Security lies in the realms of *Information Security* we are mostly concerned with matters of:

- **C**onfidentiality
- **I**ntegrity
- **A**vailability

Software Security

Software Security mainly deals with:

- The identification of *security defects*
- The implementation of *security controls*
- The management of security issues within the lifecycle of a project

Software Security Defects

Software Security Defects appear in:

- The architecture or protocol(s) of a project
- The implementation of a project (security bugs)
- The release format and distribution of a project's resources
  - Default configuration [6]
  - Packaging issues
  - Supply chain issues

[6] Patches are not usually shipped to address security defects that the end users may introduce via bad configuration

Example types of security bugs

- Missing Access Control
- Buffer overflow
- Dangling Pointer dereference bugs (Use-after-free, Double free, NULL pointer dereference)
- Unsafe Deserialization
- Injection (XSS, Command Injection, SQL Injection, Prompt Injection etc.)
- Race Condition
- Parser Differentials
- Business Logic errors
- ...

Impact of exploitation

To a System

- Denial of service
- Disclosure of sensitive information
- Memory corruption
- Code execution
- Privilege escalation
- Circumvention of business logic
- ...

To an Organization

- Financial losses
- IP theft
- Legal action
- Reputation damage
- ...

To an Individual

- Financial losses
- Identity theft
- Property theft
- Incrimination
- Reputation damage
- ...

Software Vulnerabilities

- Security bugs are also called **”software vulnerabilities”** because their exploitation by attackers may lead to the compromise of the confidentiality, integrity and/or availability of resources (services, systems, data, etc.)
- A piece of software that exploits a vulnerability is called an **”exploit”**
  - A piece of software that demonstrates a vulnerability is called a **”proof-of-concept” (or PoC)**
- A **”zero-day attack”** is an attack that exploits a previously unknown vulnerability
  - The software vendor either did not know this vulnerability existed or did not have the necessary time to address it

Recorded damages from vulnerability exploitation

Damages connected to software vulnerability exploitation

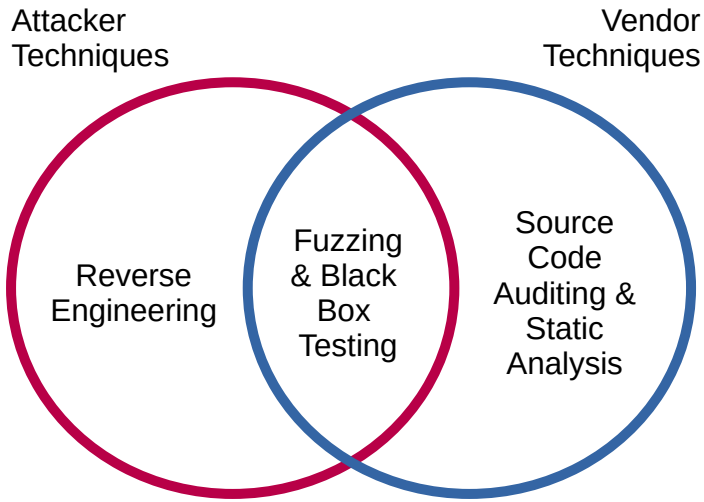
- **Morris/Internet worm, 1988**
  Exploitation of sendmail / finger / rsh (and weak passwords!),
  6000 hosts infected, $100,000–10,000,000 estimated damages
- **Stuxnet worm, 2010**
  Exploitation of 4 vulnerabilities in Microsoft Windows,
  targeted SCADA systems, damaged Iran's uranium enrichment program
- **RSA SecurID breach, 2011**
  Exploitation of 0-day bug in Adobe Flash,
  $66 million in damages
- **SolarWinds breach, 2020**
  Attack to software distribution point led to data breaches in federal systems
- **Solana Wormhole Attack, 2022**
  0-day attack to Solana ETH signatures,
  $326 million lost

The 0-day industry

- Even if the vendors / community did their best to protect users..
  - Not everyone will play nice: a growing market behind 0-day exploits
- 2005: WMF vulnerability sold by *individual* in black market for $4000
  - Response: Zero-day initiatives from various infosec vendors (acquisition and disclosure of vulnerabilities coming from independent researchers)
- 2012: Forbes on 0-day market and $250,000 sell of iOS exploit by *broker*
  - Response: Bug bounties for researchers by major software vendors (Microsoft, Apple, Google, Facebook, ...)
- 2015: French 0-day *supplier* VUPEN is forced to shutdown due to Wassenaar export controls, becomes US-based Zerodium
- 2016: details about the US Government (USG) Vulnerabilities Equity Program (VEP) are made public
  - *"the discovery of vulnerabilities may present competing equities for USG* ***offensive*** *and* ***defensive*** *mission interests"*.
- 2023: Appeals court sides with Corellium (Vulnerability Research SaaS Platform) in Apple Copyright Case

Can we win the 0-day race?

- *The Wolves of Vuln Street - The First System Dynamics Model of the 0day Market*[7]
- A joint study between HackerOne, MIT and Harvard
- Important outcome: *Creating incentives for tools and techniques that support (black box) vulnerability discovery is a more efficient way for defenders to drain the offensive stockpile of 0-day vulnerabilities.*



[7] https://www.hackerone.com/vulnerability-management/wolves-vuln-street-first-system-dynamics-model-0day-market

Why do we care?

Leaving security defects unpatched undermines

- the quality of a software product
- the trust users place in a system
- the importance of the information being processed

Can we find all of these defects automatically?

- The short answer is ”NO”.
- We need all the help we can get
  - People
  - Processes
  - Technology

Security: An ongoing process

- There's no such thing as 100% secure software
- Security is a process, not a goal

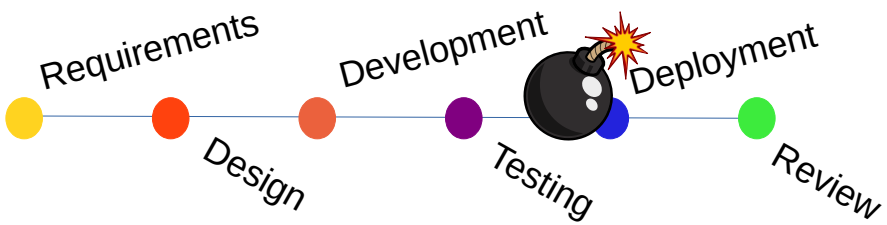
Part II

Security in the SDLC

Revelations from 60+ years of software engineering

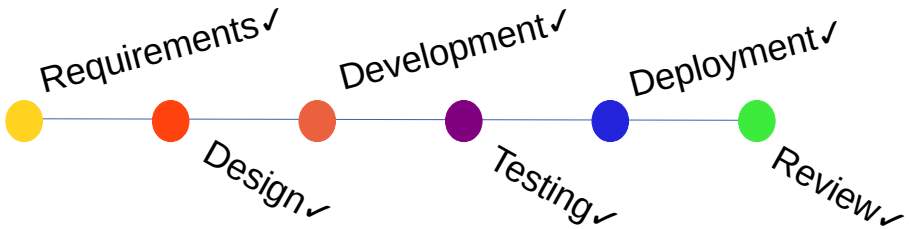
Security is often considered too late in the product lifecycle

- Danger due to **complexity**: may be difficult to fix an architectural flaw
- Danger due to **increased cost**: may be too costly to fix something outside of the development plan
- Danger due to **bad timing**: a security fix may delay a planned release creating business risks
- Danger due to **IP theft**: important data about a product may leak to competitors
- Danger due to **legal action**: users may sue the vendor due to insufficient measures taken in protecting their assets
- Danger due to **reputation damage**: the vendor / provider might not recover from its handling of a breach



Secure Software Development Lifecycle (S-SDLC)

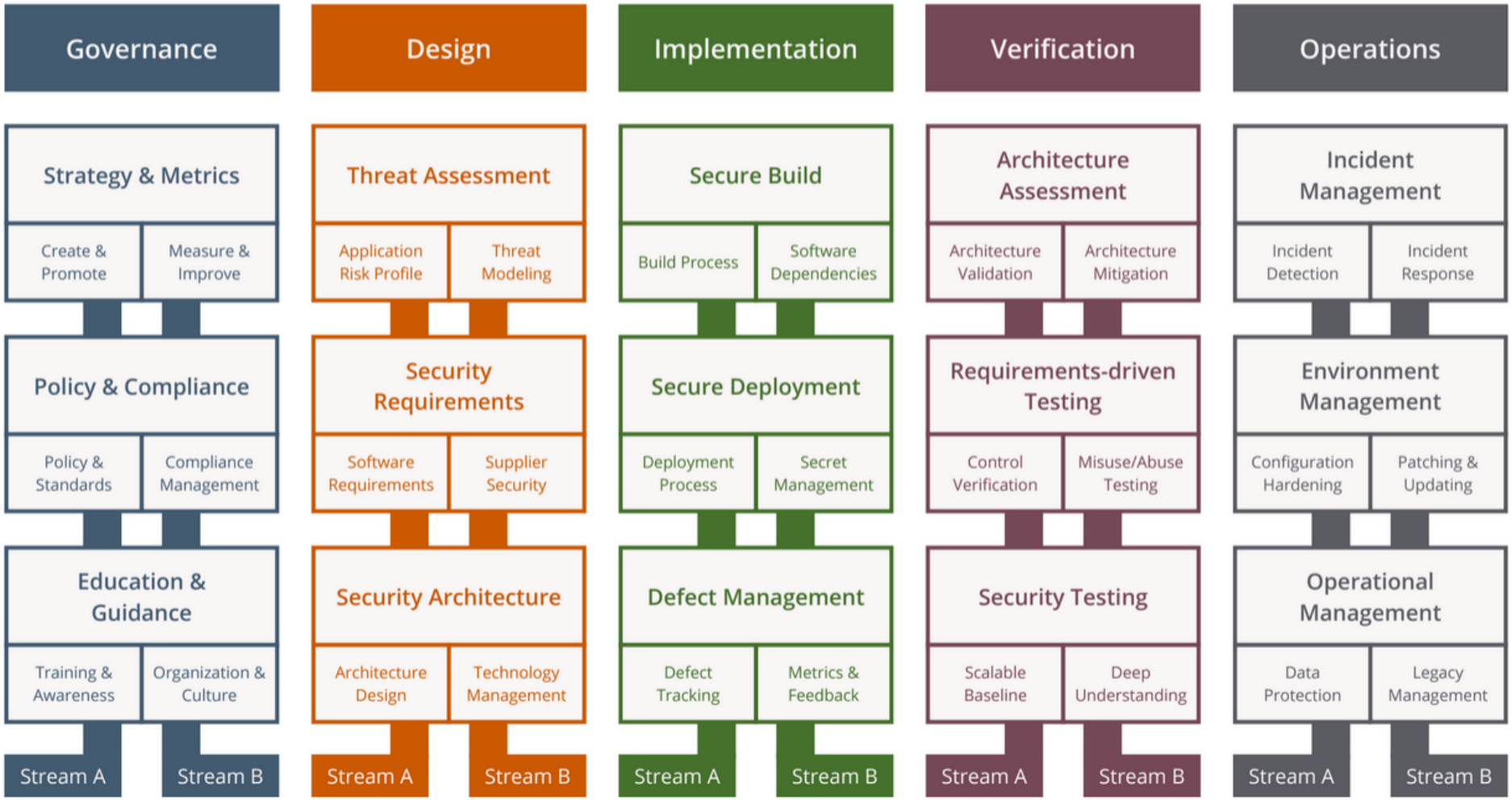
- To neutralize such project risks, a Secure SDLC (or S-SDLC) framework is applied that runs security processes throughout all phases of the SDLC
- Example S-SDLC Frameworks
  - Microsoft SDL[8]
  - OWASP SAMM[9]
  - NIST SSDF[10]
- The security processes make sure that the phase deliverable is also fine from the perspective of security



[8] https://www.microsoft.com/en-us/securityengineering/sdl/
[9] https://owaspsamm.org
[10] https://csrc.nist.gov/projects/ssdf

OWASP Software Assurance Maturity Model (SAMM)

SAMM is organized in 5 domains



Comments on SAMM

- Developed by OWASP
- Has common roots with another industry framework (BSIMM[11])
- It's easy to map practices with SDLC phases
- Code Audit is part of the "Deep understanding" practice and only applicable to high-risk components
- SAMM provides self-assessment spreadsheets[12] for organizations to measure their security posture

[11] https://www.synopsys.com/software-integrity/software-security-services/bsimm-maturity-model.html

[12] https://owaspsamm.org/assessment/

Roles and Responsibilities

- A Secure SDLC is typically managed by a ***Product Security Officer (PSO)***
- Security procedures are typically executed by ***Security Engineers***
- The ***Data Protection Officer (DPO)*** examines the data flows to make sure that personal and other sensitive information is handled correctly (see GDPR[13])
- The ***Chief Information Security Officer (CISO)*** is typically responsible for maintaining a Secure Development Environment (see ISO27001[14])

[13] https://eur-lex.europa.eu/eli/reg/2016/679/oj

[14] https://www.isms.online/iso-27001/annex-a/8-25-secure-development-life-cycle-2022/

DevSecOps

- Security expertise comes at too costly a price for most businesses
- To lower the costs but maintain an acceptable level of security, some security procedures could be infused in an automated manner within the DevOps practices of an Agile flow
- This practice is called Dev**Sec**Ops
- DevOps engineers will employ automated means to:
  - Identify the product components
  - Isolate (where possible) product components
  - Standardize security features of product components
  - Identify known vulnerabilities in the product components
  - Identify previously unknown vulnerabilities in the product components
  - Apply security patches to the product
  - Govern security attributes of the project (e.g. identities, encryption keys etc.)
  - Monitor the security of the product

Part III

Methods for the identification of security issues in software

Terminology: Threat vs Vulnerability

- A *Threat* is the possible danger of a malicious actor exercising a particular offensive technique to attack a project asset
  - e.g. An attacker will draw secrets from the database through an SQL injection attack
  - For a list of offensive techniques see the "Common Attack Pattern Enumeration and Classification Project" (CAPEC[15])
- A *Vulnerability* is a threat that can be realized due to the existence of a security defect
  - e.g. An attacker can draw secrets from the XYZ database due to an SQL injection bug in versions $\leq$ 2.5.0

[15] https://capec.mitre.org/

Terminology: White box vs Black box methodology

- White box examination
  - Bugs discovered by studying the implementation/specifications of the system
- Black box examination
  - Bugs discovered by studying the behaviour of the system

Terminology: Static vs Dynamic Analysis

- Static analysis
  - Analysis of software at rest
- Dynamic analysis
  - Analysis of software at execution time

Method #1: Threat Modeling

- *Threat Modeling* is an analytical process that enumerates the *threats* that a software system (or component) is exposed to, due to its nature of operation
- It is carried out early on in the Requirements Analysis / Design phases of the target feature
- Once developers recognize the inherent threats, they build security controls to **proactively** eliminate vulnerabilities
- This allows for better planning of security controls
- Can be performed by non-experts (e.g. by developers) as a table-top exercise using a Threat Generation methodology, such as STRIDE[16]
- Threats can be prioritized using a risk assessment model, such as DREAD[17]

[16] https://en.wikipedia.org/wiki/STRIDE_(security)

[17] https://en.wikipedia.org/wiki/DREAD_(risk_assessment_model)

Threat Modeling

- Threat Modeling steps
  1. Collect documentation and feedback on important flows and assets
  2. Map assets, flows, trust boundaries and actors
  3. Generate Threats
  4. (Optionally) suggest mitigations and prioritize threats
- There are many tools available for Threat Modeling
  - Microsoft Threat Modeling Tool[18]
  - OWASP Threat Dragon[19]
  - OWASP Pythonic Threat Modeling [20]

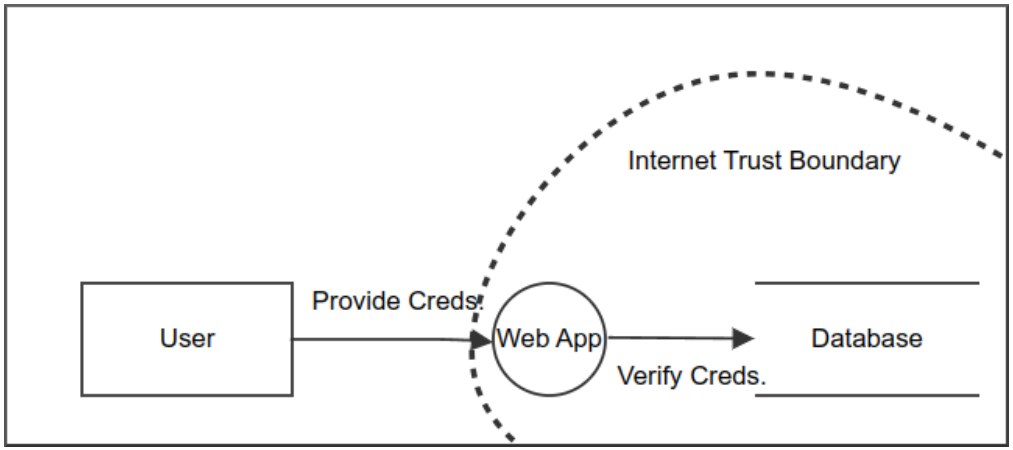
[18] https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool

[19] https://owasp.org/www-project-threat-dragon/

[20] https://owasp.org/www-project-pytm/

Threat Modeling Example

Flow Diagram for Login



Actors

- User
- Unauthenticated User
- Web App Administrator

Assets

- Web App
- User Credentials
- User Browser
- Database

Example threats based on STRIDE

ID	Threat	Mitigation	**STRIDE** Category
T1	Username fingerprinting through “invalid login” oracle	Opaque error response to login	**I**nformation Disclosure
T2	Password bruteforce attack	Login throttling control	**E**levation of Privilege
T3	Denial of service attack via multiple login attempts	Login throttling control	**D**enial of Service
T4	SQL injection during username validation	Use of prepared statement with parameterized query	**T**ampering

Method #2: Design Review

- *Design Review* (*aka* Security Design Review) is a process where security experts analyze project design documents and identify potential threats affecting the product design
- The experts provide suggestions for security controls that mitigate the identified risks
- This process is an alternative to Threat Modeling, and is suitable for the Requirements Analysis / Design phase
- This is much faster than Threat Modeling, but requires security expertise
  - One may say that the experts run threat modeling in their heads, based on their experience :-)

Method #3: Software Composition Analysis (SCA)

- A process that identifies the "ingredients" of a piece of software
- Usually a White box process
- The output of this process is a *Software Bill of Materials* (SBOM)
  - Popular SBOM formats: SPDX[21] and CycloneDX[22]
  - The US require an SBOM for all software projects deployed to federal government systems[23]
- Most SCA tools also identify *known vulnerabilities* and *license compatibility* issues for the ingredients identified
  - Popular data sources for known vulnerabilities: NVD for all software, OSV for OSS projects

[21] https://spdx.dev/

[22] https://cyclonedx.org/

[23] https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

Software Composition Analysis Example

```
$ ~/apps/snyk test --file=package.json
Testing yeoman-doctor...
Tested 22 dependencies for known issues, found 1 issue,
1 vulnerable path.

Issues to fix by upgrading:
  Upgrade latest-version@3.1.0 to latest-version@6.0.0 to fix
  x Open Redirect [Medium Severity][https://security.snyk.io/vuln/SNYK-JS-GOT-2932019] in got@6.7.1
    introduced by latest-version@3.1.0 > package-json@4.0.1 > got@6.7.1

$ npm sbom --sbom-format spdx
{
  "spdxVersion": "SPDX-2.3",
  ...
  "name": "some-software-js@0.0.1",
  ...
  "packages": [
     {
      "name": "@ampproject/remapping",
      "SPDXID": "SPDXRef-Package-ampproject.remapping-2.3.0",
      "versionInfo": "2.3.0",
      ...
      "description": "Remap sequential sourcemaps through transformations to poi
nt at the original source code",
      "downloadLocation": "https://registry.npmjs.org/@ampproject/remapping/-/re
mapping-2.3.0.tgz",
      "homepage": "https://github.com/ampproject/remapping#readme",
      "licenseDeclared": "Apache-2.0",
      ...
```

SCA Troubles

- SCA is sometimes difficult to perform with accuracy when an ingredient has been “absorbed” by the project
- Example: A developer has placed an MD5 implementation found on the world-wide-web within the file `"md5.c"` of his project
  - *What is the origin of this software?*
  - *What is its version?*
  - *Is this affected by vulnerabilities?*

Method #4: Source Code Audit

- A manual line-by-line source code examination for security bugs
- A White box approach
- Also known as 'Code Review' or 'Security Code Review'
- Most complete method of locating security bugs
- High confidence in reported bugs
- Slow process

Method #5: Static Analysis Security Testing

- A Whitebox method: Variables, functions/methods, calls are all available during the analysis
- Static analysis tools may compile the source code to their own format for faster/deeper analysis
- Will not be able to follow indirections occuring at runtime (calls to methods of abstracted objects etc.)
- Ability to look for patterns[24] of language-specific vulnerabilities
- Handy for code auditors
- Tools in this space are not perfect: vulnerabilities found require (manual) verification
- Examples: Infer, PMD, Clang Static Analyzer, ...

[24] see semgrep, for a simpler language-aware 'grep' of vulnerabilities

Method #6: Dynamic Analysis Security Testing

- Audits actual execution path of program
- Can follow external input across possibly vulnerable functions
- Capable of detecting memory management errors (double free's etc.)
- May aid an auditor in indentifying a complex issue
- Examples: Valgrind, Purify, AddressSanitizer, ...

Method #7: Reverse Engineering

- A Black box method
- Trying to understand the structure of a program by looking at the binary and its behaviour during execution
- Map functions and callers
- Identify the semantics behind certain parts of the software
- Most debuggers also offer tools to aid in reverse engineering (e.g. IDA)

Method #8: Fuzz Testing (aka Fuzzing)

- The process of feeding specially crafted data (protocol data, files etc.) to a piece of software in order to expose vulnerabilities
- A black box methodology
- Capable of quickly exposing both simple and complex vulnerabilities
- Each finding (typically a program crash) requires manual verification
- An exploratory method, where campaigns may need to be time-boxed
- Protocol-agnostic fuzzers show poor code coverage statistics
- Examples: Peach Fuzzer, libfuzzer, AFL++, AFL, ...

Method #9: Vulnerability Scanning

- Use of tools to automatically *test* for the existence of a set of specific vulnerabilities in a software setup
- Tools use database of ”known” vulnerabilities
- Heuristics used to find similar vulnerabilities
- Vulnerabilities discovered require manual verification (due to false positives)
- Very popular with web application security tests
- Example tools: Nessus, Acunetix etc.

Method #10: Functional Security Testing

- Manual and tool-assisted testing of a live application (e.g. a web application API)
- Thorough testing to identify bugs in the exposed software functions
- Black box methodology
- Example tools: Burp Suite, ZAP etc.

Method #11: Packaging tests

- Pre-release Quality Assurance testing
- Checks through manual and automated means, whether the software has been built / configured correctly before it is passed to the users
- Checks whether all security controls have been applied correctly (digital signatures, tamper protection etc.)
- Checks whether any sensitive information is present in the released format
- Example Tools: MobSF for mobile apps

Part IV

Popular Types of Software Security Assessments

Software Security Assessments

- A *Software Security Assessment* is a project whose goal is to identify security vulnerabilities in the software and/or deployment environment of a product
- Notable types of assessments are:
  - Source Code Auditing
  - Application Security Testing
  - Penetration Testing

Source Code Auditing

- Its goal is to cover as much of the codebase as possible through a manual security review
- Although complete code coverage may be requested through manual code review methods, security engineers are also free to employ:
  - Static Analysis Security Testing tools
  - Dynamic Analysis Security Testing tools
  - Functional Security Testing tools
  - Fuzz Testing and other exploratory methods
- Auditor produces report with:
  - Description of each security bug and root cause analysis, with reference to the exact point where the bug was found
  - Possible impact and rating
  - Proposed fix
- Example report: Lisk SDK 6.1 Sapphire, NFT and PoA modules Code Audit by Trail Of Bits

Application Security Testing

- Its goal is to expose all possible vulnerabilities in an application
- One may consider this as a test *in breadth*
- Methods of testing (white box / black box) depend on the details provided
- The software may be setup in a testing environment
- Tester produces report:
  - With a description of each vulnerability
  - With a rating of each vulnerability's severity
  - With suggestions for risk mitigation
- Access to different user roles may be required
- Example report: Pomerium Security Testing by Cure53

Penetration Testing

- An emulated attack towards infrastructure belonging to an organization
- Highlights which organization assets will be compromised in an attack and which are the more likely attack paths
- Exploits vulnerabilities found in this environment
- This is essentially a test *in depth*, within the context of an organization
- Requires access to the live environment
- Report includes:
  - Vulnerabilities identified during the attack and their related risks
  - Vulnerabilities exploited during the attack
  - Possible attack paths and compromised assets
- Internal / External
- Black box / White box
- Example report: DinoBank Penetration Testing by CPTC

Part V

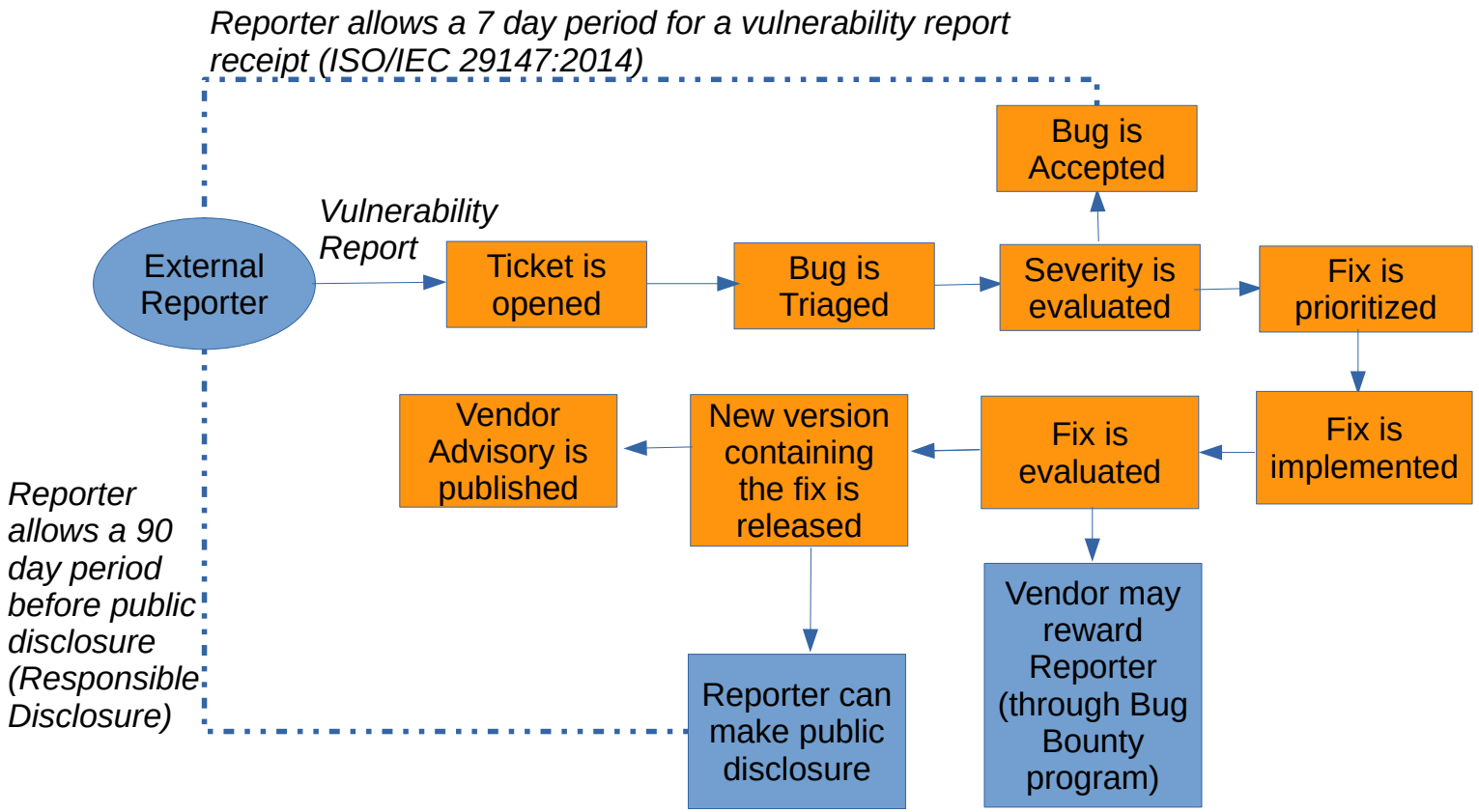
Handling Software Vulnerabilities

Who finds security defects?

An ecosystem of reporters

- The software developers
- The systems administrators
- The security engineering team of the software vendor
- Information security companies
- Independent security researchers
- The software users

Vulnerability Timeline



Note: steps marked in orange are the same for internal reporters as well.

Special Case: A security control becomes obsolete

- Sometimes security mechanisms become obsolete because of new developments in research, hardware or software
  - e.g. a weak hashing scheme (MD5) is used to check the integrity of a certificate
- Management must allocate resources to examine alternatives
- Changes may need to be made to security documentation of the product
- It is good practice to let users know which updates change security controls

Security Bug Disclosure

- Security bugs affect software that may be used by millions of users
- Security bugs are disclosed publicly (as *advisories*) in order to aid in coordinated patching procedures
- Where are security bugs disclosed?
  - Full Disclosure Mailing List[25]
  - OSS Security Mailing list[26] (for bugs concerning open source projects)
  - Open Cloud Vulnerability & Security Issue Database[27] (for cloud provider vulnerabilities)

[25] https://seclists.org/fulldisclosure/

[26] https://www.openwall.com/lists/oss-security/

[27] https://www.cloudvulndb.org/

CWE: An Ontology of Security Vulnerability Types

- CWE (Common Weakness Enumeration) is an online database of vulnerability types
- e.g. **CWE-121:** Stack-based Buffer Overflow
- Available online at `https://cwe.mitre.org/`
- Can be used in a security advisory to better classify the bug in question

CVE: A registry of Vulnerabilities

- CVE (Common Vulnerabilities Enumeration) is a numbering scheme that acts as a registry for software vulnerabilities
- e.g. **CVE-2014-1270:** WebKit, as used in Apple Safari before 6.1.2 and 7.x before 7.0.2, allows remote attackers to execute arbitrary code or cause a denial of service (memory corruption and application crash) via a crafted web site
- Can be used to track which security bugs are patched in a software update
- CVE Program Website: `https://www.cve.org/`

CVE numbering (where vendor meets the reporter)

- CVE numbers are issued by CNAs (CVE Numbering Authorities)
- Main CNA is MITRE (some vendors act a CNA for their products)
- Once a bug has been acknowledged, the bug reporter may ask for a CVE number
- CVE number is shared with vendor
- Vendor publishes software update and advisory with a link to the CVE number
- Reporter may publish advisory with a link to the CVE number

An example (vendor) advisory

```
Subject: CVE-2024-27439: Apache Wicket: Possible bypass of CSRF protection
Severity: moderate

Affected versions:
- Apache Wicket 9.1.0 through 9.16.0
- Apache Wicket 10.0.0-M1 before 10.0.0

Description:

An error in the evaluation of the fetch metadata headers could allow a bypass of the CSRF protection in
Apache Wicket. This issue affects Apache Wicket: from 9.1.0 through 9.16.0, and the milestone releases
for the 10.0 series. Apache Wicket 8.x does not support CSRF protection
via the fetch metadata headers and as such is not affected.

Users are recommended to upgrade to version 9.17.0 or 10.0.0, which fixes the issue.

Credit:
Jo Theunis (finder)

References:
https://wicket.apache.org/
https://www.cve.org/CVERecord?id=CVE-2024-27439
```

Part VI

Conclusion

Recap: Important Concepts

- Software Assets, Threats and Vulnerabilities
- Proof-of-concept, Exploit and 0-day attacks
- Secure SDLC
- PSO, Sec. Engineer, DPO, CISO
- Threat Modeling, Software Composition Analysis, Source Code Auditing, Application Security Testing, Fuzz Testing
- DevSecOps
- CAPEC, CWE, and CVE

Further reading material

- *"This is how they tell me the world ends: The Cyber Weapons Arms Race"*
- *"Agile Application Security"*
- *"Threat Modeling: Designing for Security"*
- *"Securing DevOps: Security in the Cloud"*

Part VII

About this course

Course Goals

- Ability to apply security-related best practices in software development
- Ability to identify security bugs in software
- Ability to demonstrate a security bug
- Ability to assess the risk associated with a bug
- Ability to manage vulnerabilities within a product lifecycle
- Acquaintance with latest research on software security topics

Coursework

- Four (4) exercises requiring both personal and team effort
  - 40% of exercise marks will also be redeemable through successful publishing of CVEs
- No exams

Prerequisites

- C
- Java
- Linux
- Not strictly required, but useful skills
  - Microsoft Windows
  - Ability to use a scripting language (Python, UNIX shell, Powershell, VBScript, etc.)
  - x86 assembly skills

Course Outline

1. Software security principles & SDLC integration
2. Security bugs related to the environment of execution
3. Memory related bugs
4. Web Application Security
5. Web Application Security Workshop (Extra!)
6. Dealing with parsing and other file-related bugs
7. Workshop on black-box vulnerability research
8. Avoiding bugs in cryptographic mechanisms
9. Mobile App Security
10. Static Analysis with CodeWeTrust
11. DevSecOps principles

Questions?