Seguridad en el Desarrollo de Aplicaciones

Nuestro curso presenta un **abordaje inicial** al tema de la **seguridad de aplicaciones de software** desde una perspectiva teórico-práctica, discutiendo los tipos tipos de fallos (vulnerabilidades) de seguridad que se pueden observar en las apliaciones de software.

Se discuten desafíos para varios tipos de vulnerabilidades de software y prácticas seguras de desarrollo de software. Utilizamos herramientas de análisis estático y dinámico para la detección y corrección de dichas vulnerabilidades.

Los ejemplos, problemas, o fallas (vulnerabilidades) se presentan en syntaxis del **lenguaje C** (*típicamente conocido por ser suceptible a fallas de seguridad y a ataques*), y ocacionalmente en otros lenguajes como **Python**, **Shell**, **Java**, **OCaml**, etc.

¿Lo que vamos a Aprender?

- Principios de diseño seguro (**Saltzer and Schroeder**):
  - economy of mechanism
  - fail-safe defaults
  - complete mediation
  - open design
  - separation of privilege
  - least privilege
  - least common mechanism
- Cómo los principios de diseño seguro se evidencian en el desarrollo de aplicaciones de software?
- Técnicas de validación de datos de entrada (**input validation**)
- Cómo las vulnerabilidades son registradas (**CVE, CWE**)
- Tipos de ataques informáticos:
  - Cross domain attacks
  - command injection
  - SQL injection
  - Clickjacking
- Memory vulnerabilities:
  - buffer overflow
  - heap overflow
  - Integer overflow
  - pointer overwrites
- Técnicas de análisis estático y dinámico para prevención de problemas de seguridad.

  - model-checking
  - symbolic execution
  - concolic execution
  - taint analysis
- Introducción a técnicas criptográficas.

Metodología

Parte sincrónica

- Slides
- Discusión en clase
- Demostración de algún concepto, error o ataque

Parte asincrónica

- Vídeos sobre material complementario ó técnica
- Demostración de programas ó aplicaciones software
- Tutorial de aprendizaje
- Talleres
  - Lectura de un artículo (preguntas y respuestas)
  - Dado un programa en C, identificar tipos de errores
  - CVE, CWE

Evaluación

# taller	%	Tema	Fecha-Publicación	Fecha Entrega
1	10%	Seguridad de Chromium	22-04-2025	28-04-2025
2	10%	Validación de datos de entrada	29-04-2025	05-05-2025
3	10%	Analizar un CVE de prioridad alta	06-05-2025	12-05-2025
4	10%	Lectura: Make Least Privilege a Right	13-05-2025	19-05-2025
5	10%	Presentación tarea 3	20-05-2025	26-05-2025
6	10%	Práctico: fallas de seguridad en C/C++	27-05-2025	02-06-2025
7	10%	Práctico: fallas de seguridad en C/C++	03-06-2025	09-06-2025
8	10%	Memory corruption	10-06-2025	16-06-2025
9	20%	Criptografía	17-06-2025	23-06-2025

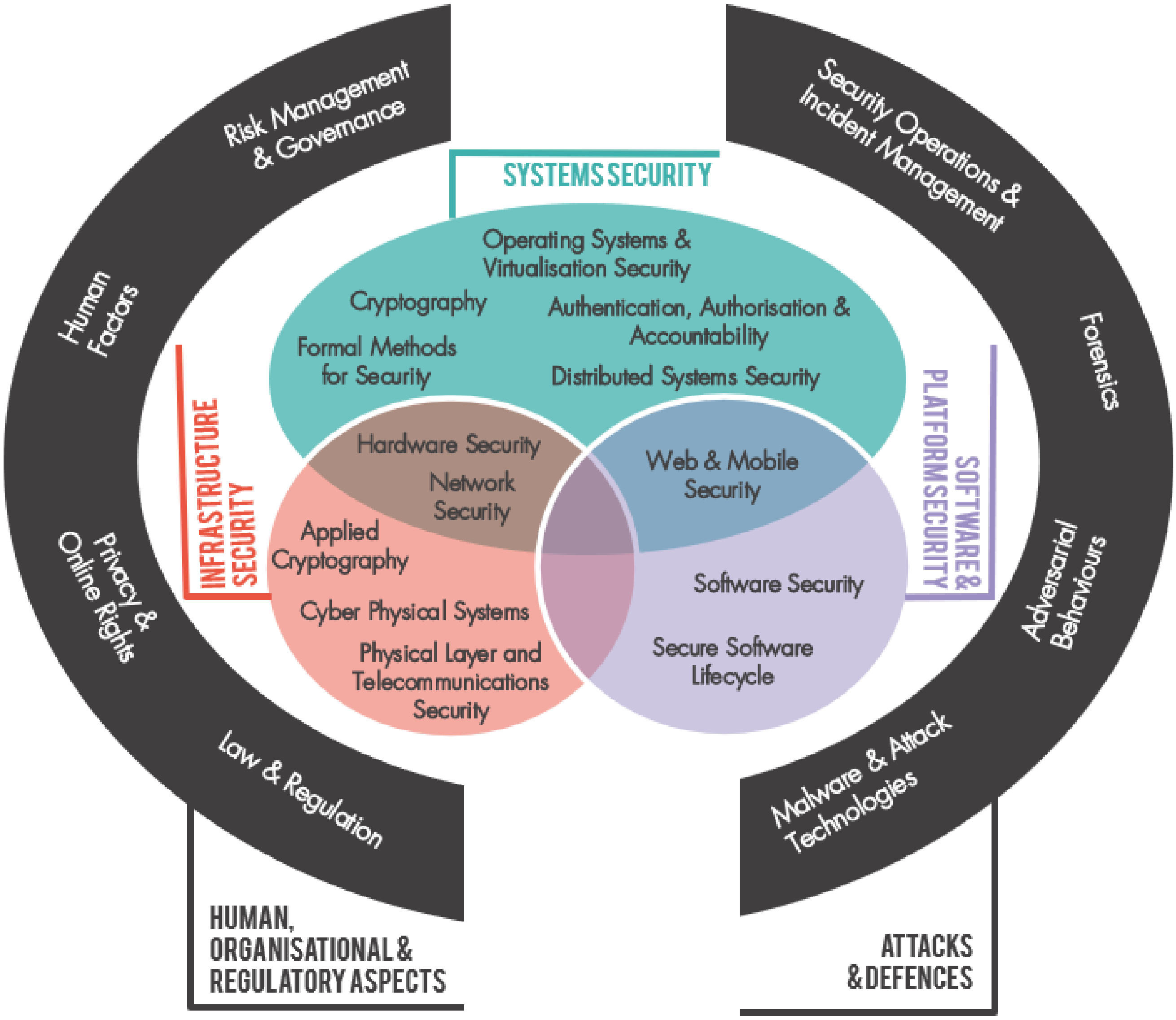
Software Security Landscape

Software Security Landscape

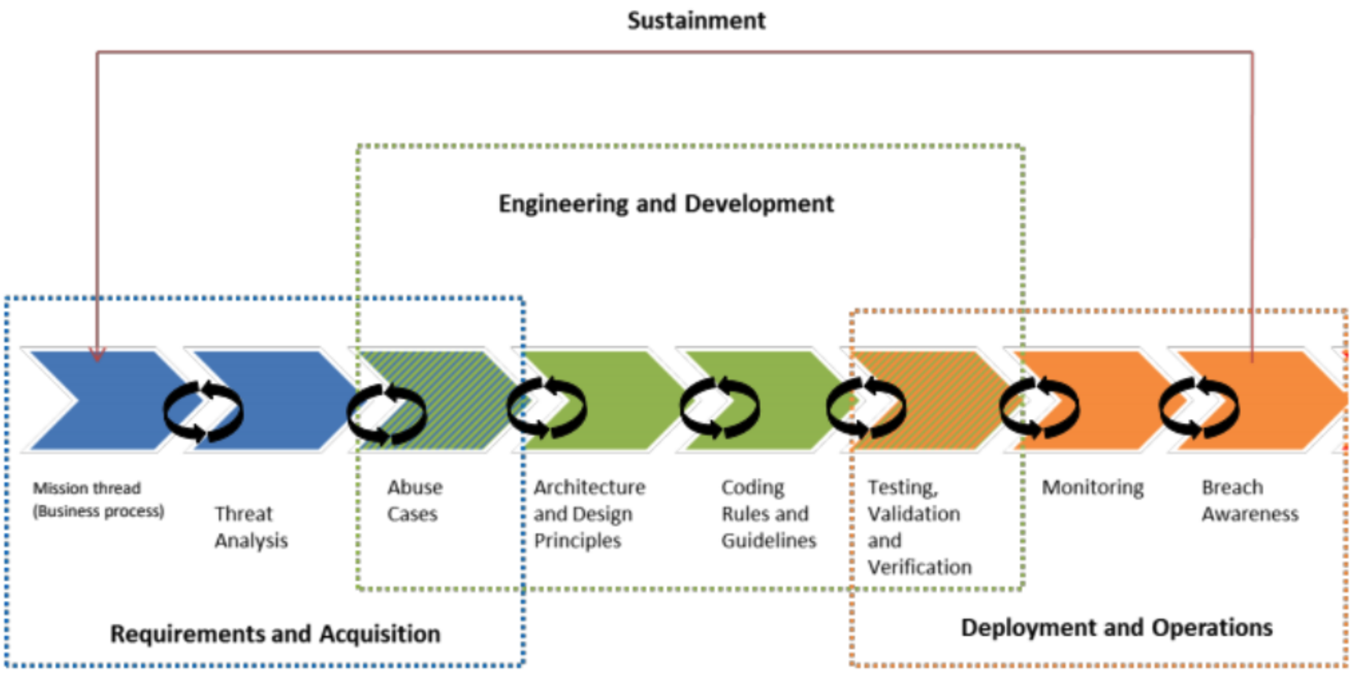
- software security issues
- cost of security vulnerabilities
- the range of security flaws
- causes of insecure software
- secure design principles
- classifying and recording vulnerabilities

Security Body of Knowledge

Source: cybok.org



Security in Software Lifecycle



Our focus: Design and development Source: Mark Sherman, CERT / SEI Webinar

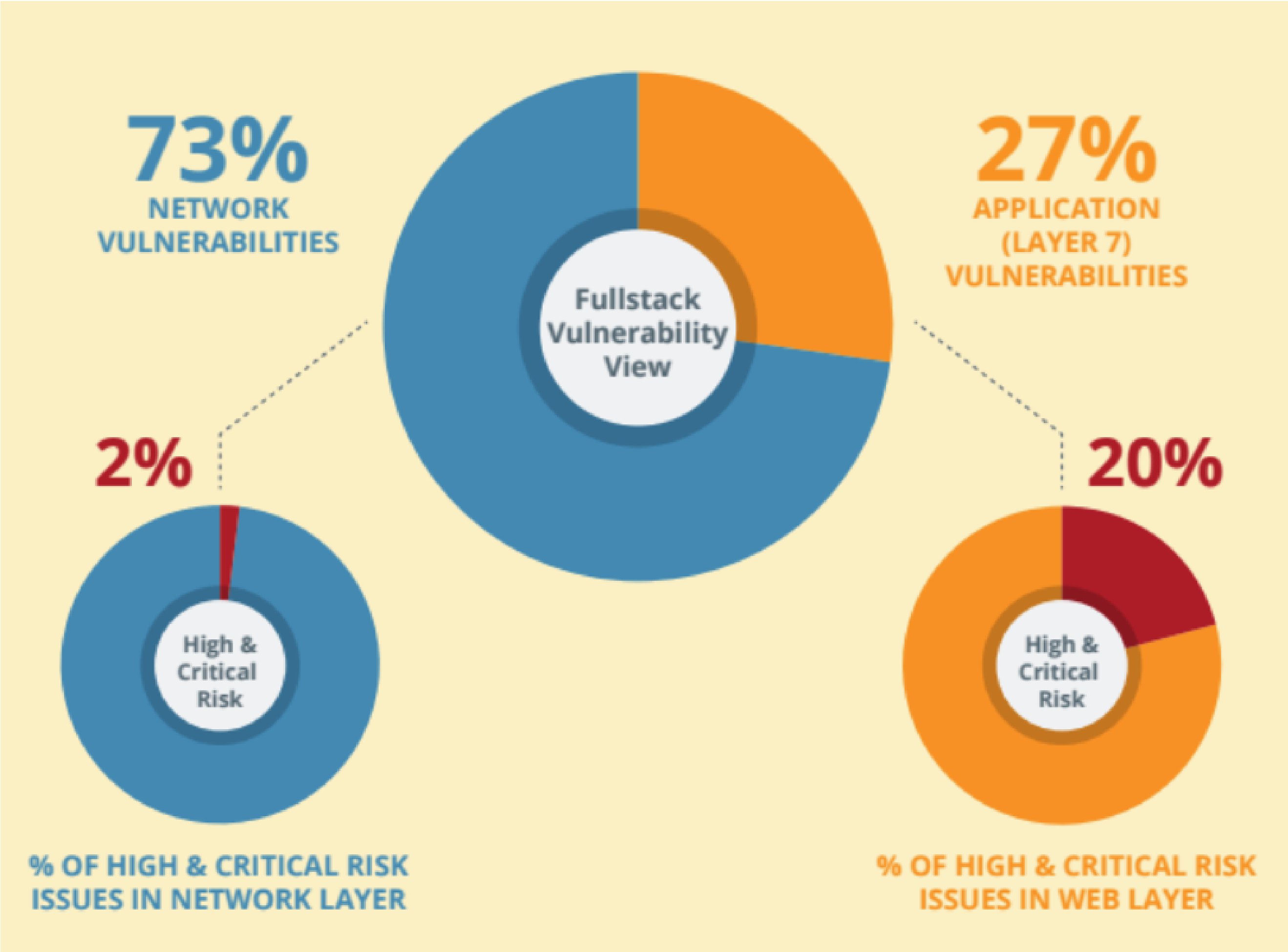
---

Software Security Issues

- Understand vulnerabilities and causes
- Prevent them by writing secure software
- **Learn**:
  - Sound principles
  - Specific techniques and coding patterns
- Tools for detection, analysis, and testing
- **Goal**: Software engineering for security

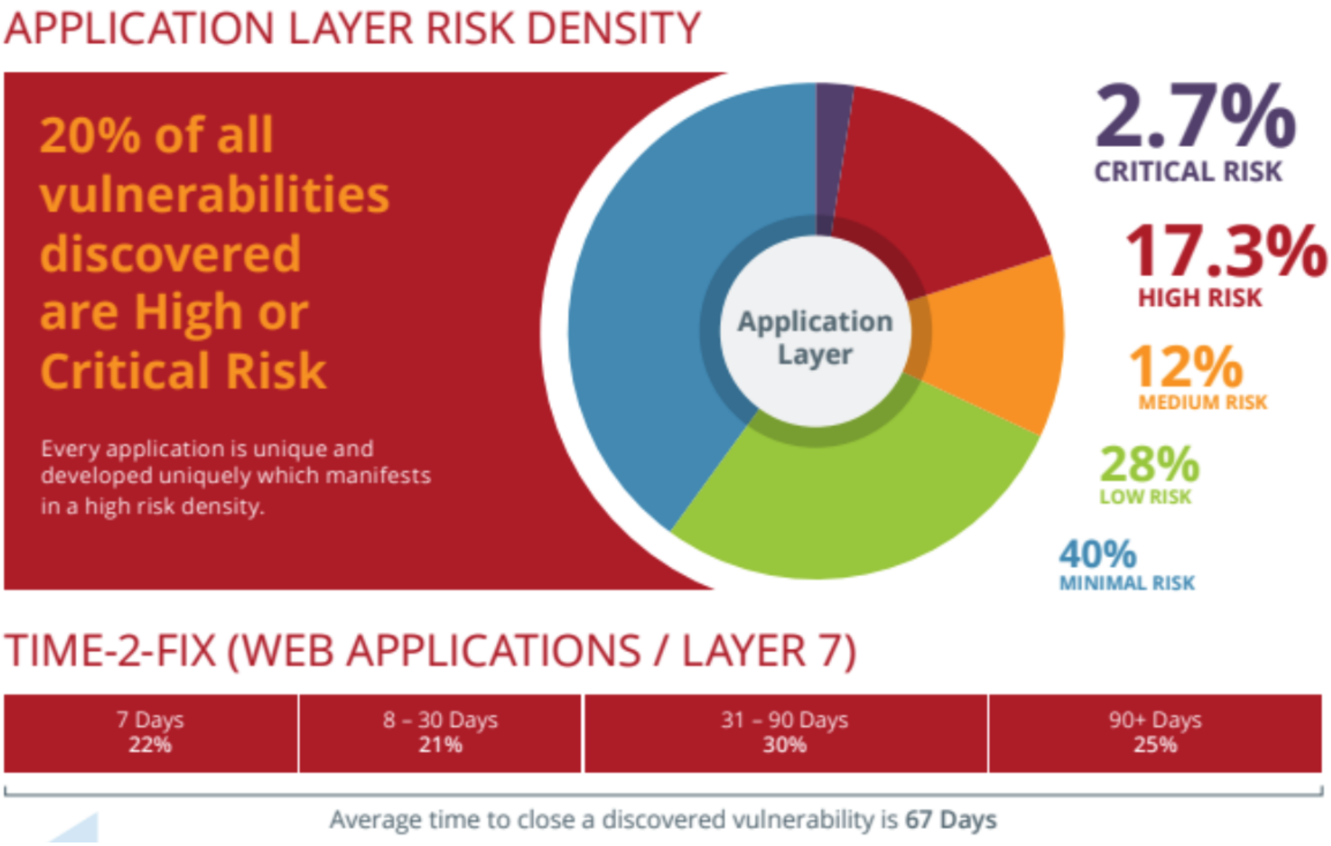
Location of Vulnerabilities

Web Applications vs. Network Layer



Source: Edgescan 2018 Vulnerability Statistics

Vulnerabilities by Risk



Vulnerabilities by Type (Web)

Cost of Security Vulnerabilities

How Long To Fix?

Statistics: Dan Cornell, RSAConf. 2012

- **9.6 minutes** for stored XSS
- **16.2 minutes** for reflected XSS
- **84 minutes**: stored/reflected XSS (total)
- **It doesn't matter**: Bigger picture
- **All of the above**

Remediation: Not Just Bug Fix!

- Need to:
  - Set up development environment
  - Fix the vulnerability

  - Confirm the fix
  - Do functional testing
  - Deploy
  - **+ Overhead**
- Total cost is much more than just the actual fix!

Cost of Cybercrime

- **McAfee report**: $600 billion (2018)
- **Accenture report, 2017 (250 companies):**
  - $11.7 million per company
  - 130 successful breaches per company!
  - Both figures increased about **25%** in one year
- **50 days average**: Resolve insider attack
- **23 days average**: Resolve ransomware

Direct Costs of Incidents

	SMB		Enterprise	
	Proportion of business incurring this expense	Typical losses	Proportion of business incurring this expense	Typical losses
Professional services	88%	$11K	88%	$84K
Lost business opportunities	32%	$16k	29%	$203K
Down-time	34%	$66K	30%	$1.4M
Total expected typical damage	$38K		$551K	

Survey of **5500 companies, 2015**
Source: Kaspersky Lab: IT Security Risks Special Report, 2015

Indirect Costs of Incidents

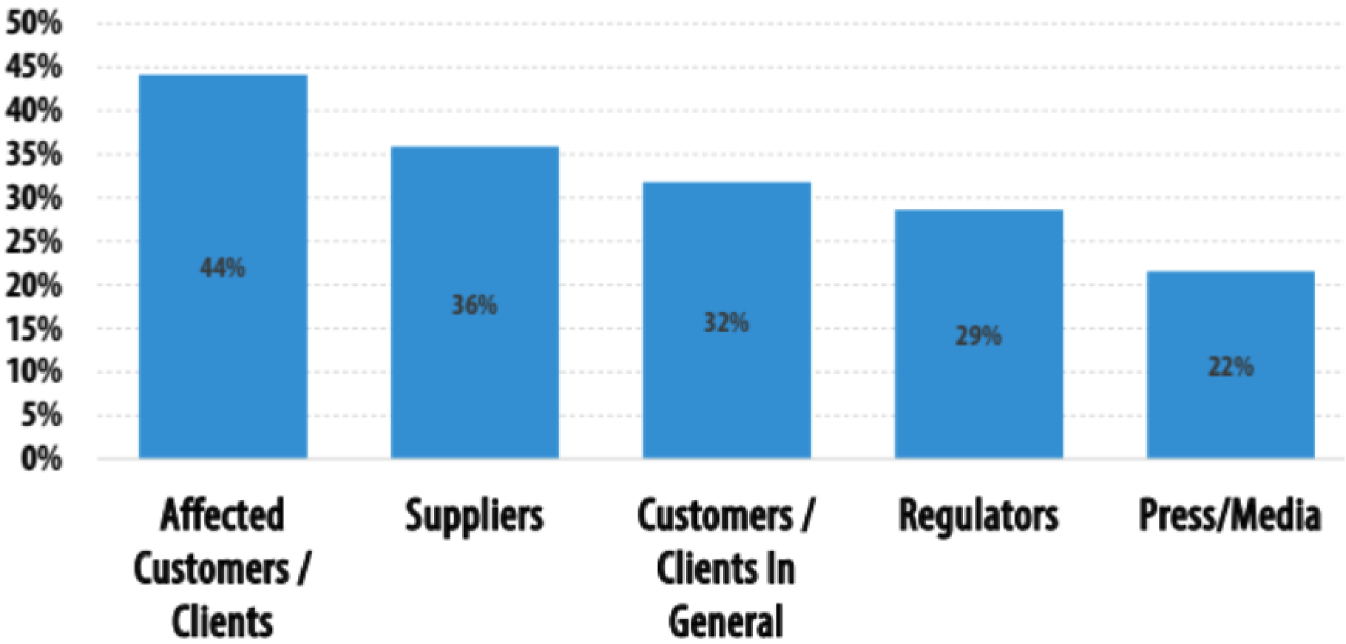
	SMB		Enterprise	
	Proportion of business incurring this expense	Typical losses	Proportion of business incurring this expense	Typical losses
Staffing	41%	$5.5K	40%	$52K
Training	47%	$5k	53%	$33K
Systems	54%	$7K	54%	$75K
Total expected indirect spend	$8K		$69K	

Expenses to prevent incidents in the future

Source: Kaspersky Lab: IT Security Risks Special Report, 2015

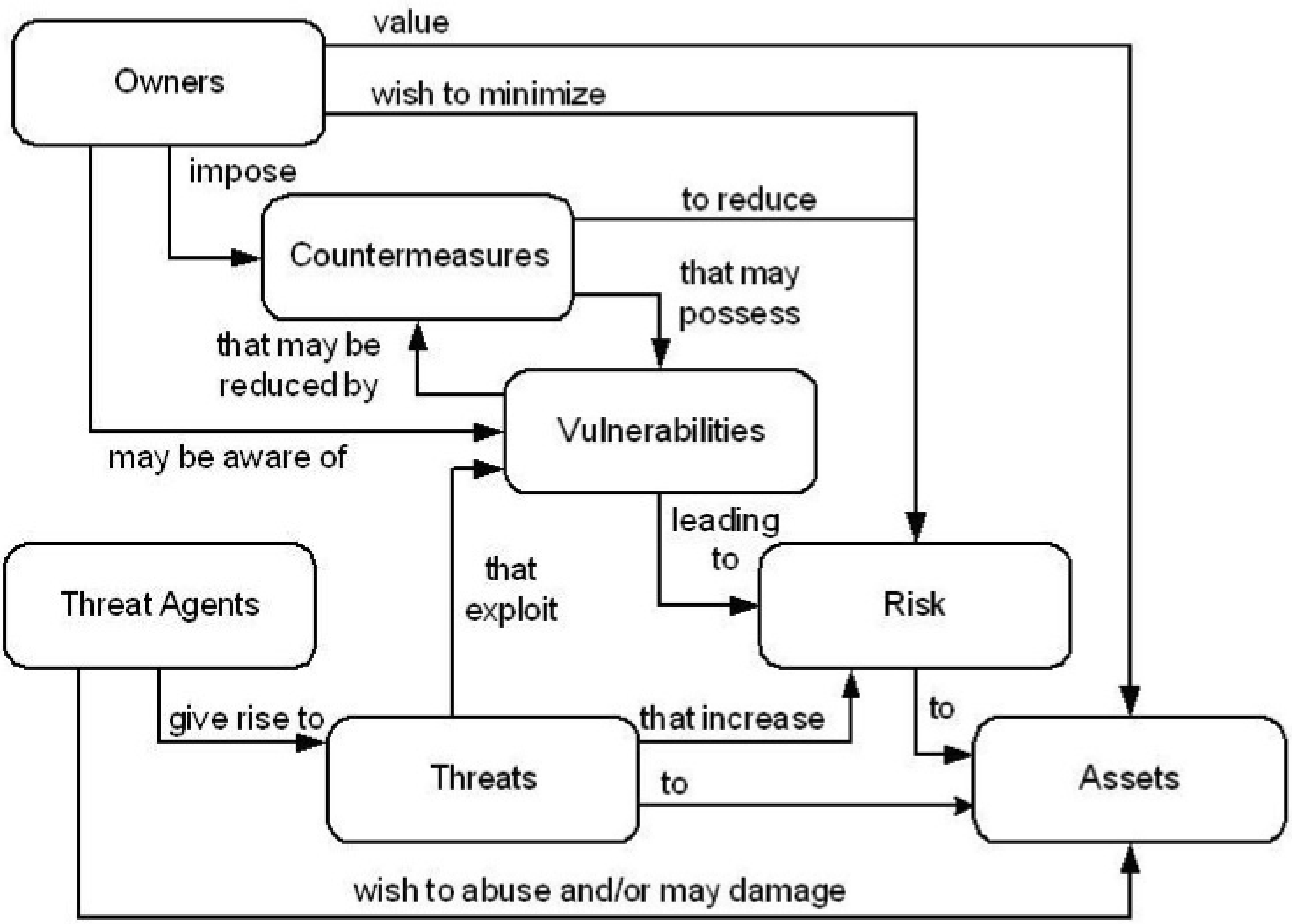
Disclosure and Reputation

- Need to disclose breaches to:
  - Stakeholders
  - Authorities
  - Public



The Range of Security Flaws

Vulnerabilities in the Picture



From **ISO 15408 standard**

What is a Vulnerability?

- A weakness that can be exploited by an attacker to perform unauthorized actions
- **Unauthorized** = Violating security policy
- Very broad definition
- For software, we focus on a few common classes of vulnerabilities
  - 5 major vulnerabilities
  - See **Software Security Knowledge Area (Cyber Security Body of Knowledge)**

(i.) Memory Corruption Vulnerabilities

- Also called **memory management vulnerabilities**
- Found in imperative languages where **allocation/deallocation** is the programmer's responsibility
  - **Spatial vulnerability** (out of bounds)
  - **Temporal vulnerability** (memory no longer in use)
- **Data-only attacks**
- **Code corruption (code injection)**
- **Control-flow hijack**: Without injection
- **Information leak**

(ii.) Structured Output Generation (Code Injection)

- Dynamic construction of output in some syntax (**SQL query**, **HTML** page)
- If unchecked construction from untrusted input → Data interpretable as commands (SQL injection, command/script injection)
- Problematic cases:
    - Sublanguages with different syntax (**JavaScript** embedded in **HTML**)
    - Multi-phase processing (stored and higher-order injection)

(iii.) Race Condition (in concurrency)

- Concurrency may lead to nondeterminism
- Attacker may control ordering/timing of concurrent actors, violating security goal
- **Time-of-check to time-of-use**: Invalidate a condition assumed to have been checked (**Complete mediation**)
- **File system race condition**: Privileged access by an unauthorized user
- **Corruption of web session state** by incorrectly synchronized threads

(iv.) API Vulnerabilities

- Communication interface between components (e.g., program and library)
- Any API has **conditions for correct use**: **Contract** that must be observed
- **Violating contract** brings system into an error state that may be exploited
- **Common issue**: Misuse of cryptographic APIs (flexibility makes correct use harder)

(v.) Side-channel Vulnerabilities

- Conveys information about program execution through **implementation details below the software abstraction level** (e.g., power, timing, microarchitecture state)
- **Covert channel**: Attacker also controls the program communicating via the side channel
- **Usual attack**: Information leak
- Can also result in **fault injection attacks** (driving hardware outside normal range)

Conclusion: Flaws as Contract Violations

- Vulnerabilities are flaws that can **lead to violating security policies** (i.e., a particular kind of specification/contract)
- Not necessarily a one-to-one connection
- **Formalizing security objectives** in detail can help both **prevention** and **detection** of vulnerabilities

---

Causes of Insecure Software

Some Quality Stats

- **90% of security incidents** due to **exploits against software defects** (DHS)
- **75% of software vendor applications** fail to comply with **OWASP Top 10 list**

[Owasp Top 10]

Some Insecurity Causes

- **Insecure Coding Practices**:
  - 30% of companies don't scan for vulnerabilities during development
- **Threat Landscape is Shifting**
- **Reuse of Vulnerable Components/Code**
- **Programming Language Idiosyncrasies**
  - Languages are selected by **application type**, without considering security
  - **Source**: Veracode Report, 2016
  - **Recent**: Veracode Report, 2025

Classification of Causes

1. **Technical factors**
2. **Psychological factors**
3. **Economic factors**

Technical Factors

- **Complexity of software products** increases the **attack surface**
- **Reuse of old code/libraries** (not designed for security, often not scrutinized)
- **Combination of components + incorrect assumptions** may introduce new flaws
- **Difficult software characteristics**: Parallel, multi-user, nondeterministic, etc.

Psychological Factors

- **Mental models**: Humans check for issues they **expect**, but fail to see new ones
- **Asymmetry**: An attacker only has to exploit **one flaw**, while a defender must secure **everything**
- **Assumptions**: Developers assume software is **used as intended**, but attackers **misuse it**

Economic Factors

- **Tight deadlines**
  - Limited time/resources for testing
- **Security is just one feature**
- **Legacy software**: Can't rewrite
- **Democratization**: Mobile app development
- **Closed-source vs. Open-source** (peer-reviewed vs. internal review)

Vulnerability perceptions

Why do apps have vulnerable code?

1. Developers don't understand secure coding practices
2. Poor coding by application developers
3. Use of legacy libraries and databases
4. Development tools and technologies with inherent bugs

Secure Design Principles

Before Coding Comes Design

- Insecurity often stems from implementation details.
- Design flaws can be even more critical.
- Revisit and consistently evaluate secure design principles.
- Basic principles are timeless.

Saltzer & Schroeder, **The Protection of Information in Computer Systems**, 1975

- 10 principles

- PDF

---

1. Economy of Mechanism

- Keep it simple
- Reduces

- likelihood of errors
- complexity of scenarios
- potential for misunderstanding

- Allows for reasoning through inspection (potentially using formal methods).

2. Fail-safe Defaults

- Base access on permission, not exclusion (whitelisting, not blacklisting)
- Argue why access should be granted, not the opposite
- On crash/fail, default to secure behavior
- Mistakes will be detected (access denied)
- Failure by allowing unauthorized access would go undetected!

3. Complete Mediation

- Check everything, every time.
- Don’t rely on prior checks—object/permissions/environment may change.
- Require system-wide access control.
- Update mechanisms with authority changes (e.g., revocation).

4. Open Design

- no security through obscurity .
- Based on Kerckhoff’s principle :
    - a cryptosystem should be secure even if everything about it, except the key, is public knowledge.
    - Keep mechanisms public; keep keys secret.
- Enables scrutiny, formal proofs, and realistic expectations.

5. Separation of Privilege

- Require more than one check for access.
- Prevents single point of failure.
- Common in workflows (bank loans, patient data, etc.).
- Example: two-factor authentication.

6. Least Privilege

- Grant minimum rights for the task .
- Reduces exposure and post-compromise damage.
- Example: sandboxing.
- Minimizes required analysis scope post-incident.

7. Least Common Mechanism

- If possible, two different functionalities should not share a common (specially critical) mechanism.
- Every shared mechanism is a potential for security compromise
- **Shared code**: extra features interaction, invalid assumptions.
- **Shared data**: mutual influence, incorrect assumptions, information flow.

8. Psychological Acceptability

- Usability is critical.
- Align security with user mental models.
- Prevent misuse (e.g., requirements on password change).
- Design helpful UI for pop-ups/messages.

Tygar: why johnny can't encrypt

Shen: why johnny still can't encrypt

9. Work Factor

- not to design an idealized system, but to consider the strength of an attacker
- Weigh attacker effort vs. protection level.
- No system is completely secure, evaluate risk-cost tradeoff.
- Example: offline vs. online password attacks.

10. Compromise Recording

- Security is not bulletprof: install auditing mechansisms.
- Detect what you can’t prevent.
- Install appropiate audit/logging.
- Easier and cheaper than full prevention.

Other Principles

- **Weakest Link:** Overall system is as secure as its weakest point.
- **Defense in Depth**: a single breach will not compromise a systems
- **Security by Design** (not as an afterthought unlike the Internete)

Classifying and Recording Vulnerabilities

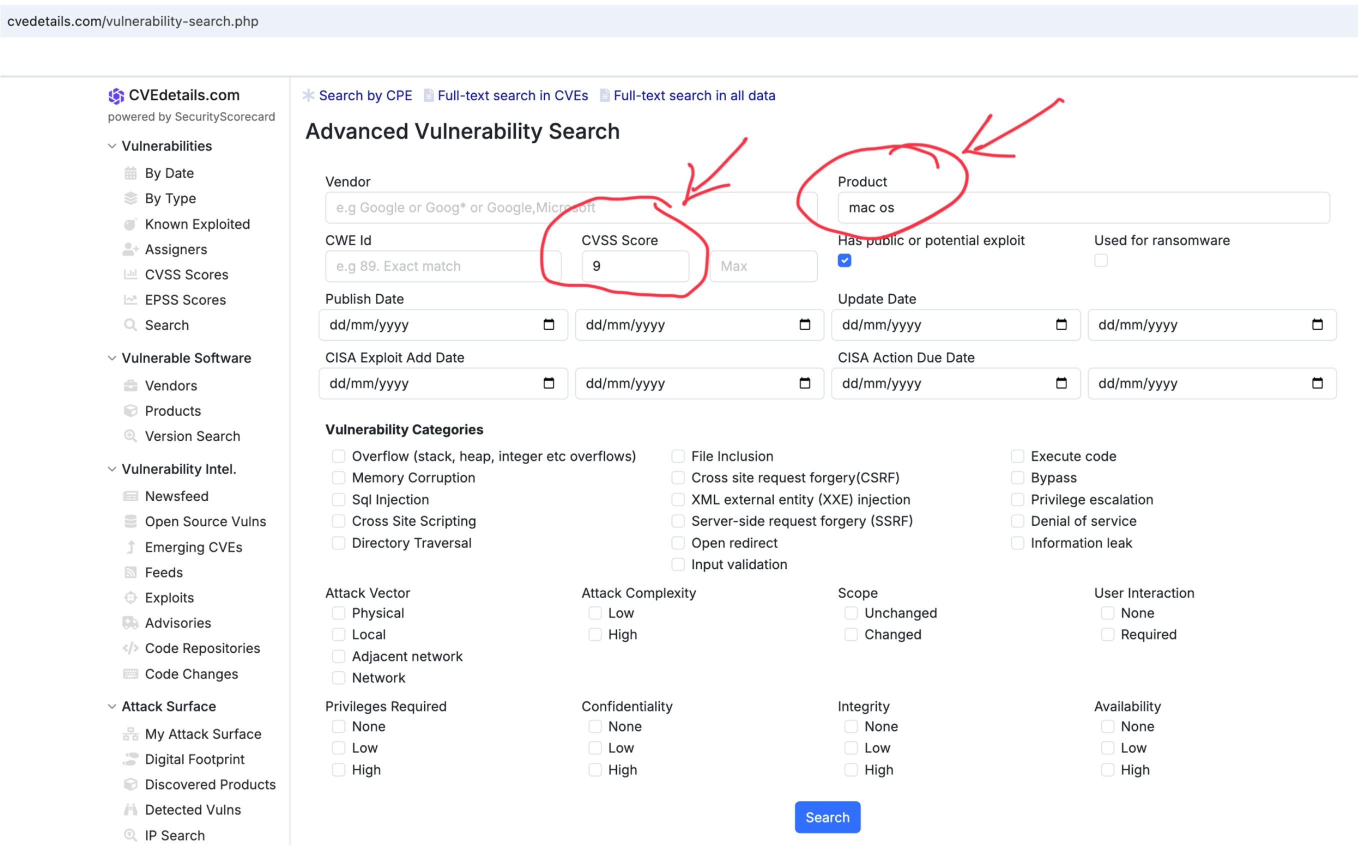
Weakness vs. Exploit

- **Vulnerability:** an exploitable instance of a weakness that can be exploited by an attacker.
- **Weakness:** classes of vulnerabilities, e.g., a bug/flaw (e.g., buffer overflow, injection) that could lead to a vulnerability.
- **Exposure:** mistake or configuration issue than can be used as an entry point.
- **Exploit:** piece of code, software, or a set of commands that takes advantage of a vulnerability or flaw .
- Useful: classify both **weakenesses** and **vulnerability** instances.

CVE (Recordings)

- Mitre CVE List
- NVD - National Vulnerability Database
- CVE Details

CVE Details



CVE Entries

- CVSS score: 0-10, it measures the technical severity of a vulnerability.

- EPSS score: 0-10, it measures how likely is the vulnerability to be exploited within the next 30 days.

CVE (Common Vulnerabilities and Exposures)

- Standardized vulnerability IDs, hosted by MITRE.
- Format: CVE-YYYY-NNNN (changed in 2014 to support more digits).
- CVEs link vendor advisories and **proof-of-concepts**.

CWE (Common Weakness Enumeration)

- MITRE-hosted list of known software flaws.
  - Mitre CWE
- Provides identification, examples, mitigation.
- CWEs can be hierarchical: e.g.,
  - 120 (classic buffer overflow),
  - 121 (stack),
  - 122 (heap).

CVE Entries

- Steps: Report → ID assignment → Disclosure → Listing.
- Example: CVE-2018-7600 (Drupal remote code execution).
- NIST's NVD provides CVSS scores and analysis.
- **Impact**: e.g. Confidentiality, Integrity, Availability, Non-Repudiation.
- Relationship with other CWEs
- Mitigation strategies

CWE - (Common Weakness Enumeration)

Examples:

CWE-807 Reliance on Untrusted Inputs in a Security Decision

CWE-22 Improper Limitation of a Pathname to a Restricted Directory (Path Traversal)

CWE-134 Uncontrolled Format String

CWE-190 Integer Overflow or Wraparound

CAPEC (Common Attack Pattern Enumeration & Classification)

- Describes attack steps in detail.
- Maps to CWEs (e.g., attack targets specific weakness).
- Not limited to software—includes hardware, physical, and social engineering attacks.
- Steps that an attacker would take to exploit a vulnerability.
- Capec Mitre

Some Representative Exploits

Chromium Upgrade (v66 → v67)

- 24 CVEs addressed:
  - 6 out-of-bounds access
  - 2 heap buffer overflows
  - 2 use-after-free
  - 2 bypasses

Chromium vs Chrome

Feature	Chromium	Google Chrome
Open source?	✅ 100% open-source	❌ Contains proprietary components
Maintained by	Google + community	Google
Includes Google branding?	❌ No logos, no auto-updates	✅ Yes (logos, auto-update, crash reports)
Built-in Flash/PDF	❌ May not be included	✅ Included
Sync with Google	❌ No	✅ Yes

Meltdown & Spectre

Vulnerabilities in the Picture

- Exploit out-of-order/speculative execution.
- Break memory isolation in modern CPUs.
- Require kernel and hardware updates.

YouTube: Meltdown and Spectre

Detecting and Reporting Vulnerabilities

The Business of Bugs

- Zero-day vulnerabilities = valuable assets.
- Markets:
  - **White market**: responsible disclosure.
  - **Grey market**: gov/law enforcement buyers.
  - **Black market**: underground economy.

Disclosure Models

- **Responsible disclosure**: coordinate with vendor.
- **Full disclosure**: public info forces faster fixes.
- **Non-disclosure**: private use or NDA-bound sharing.

Bug Bounty Programs

- Encourage ethical reporting, offer rewards.
- Examples:
  - **Zero-Day Initiative**
  - **Bugcrowd** (400+ programs)

Risks of Full Disclosure

- Image: HP Cyber Risk Report 2016.
- Pressures vendors, but may expose users before patching.
