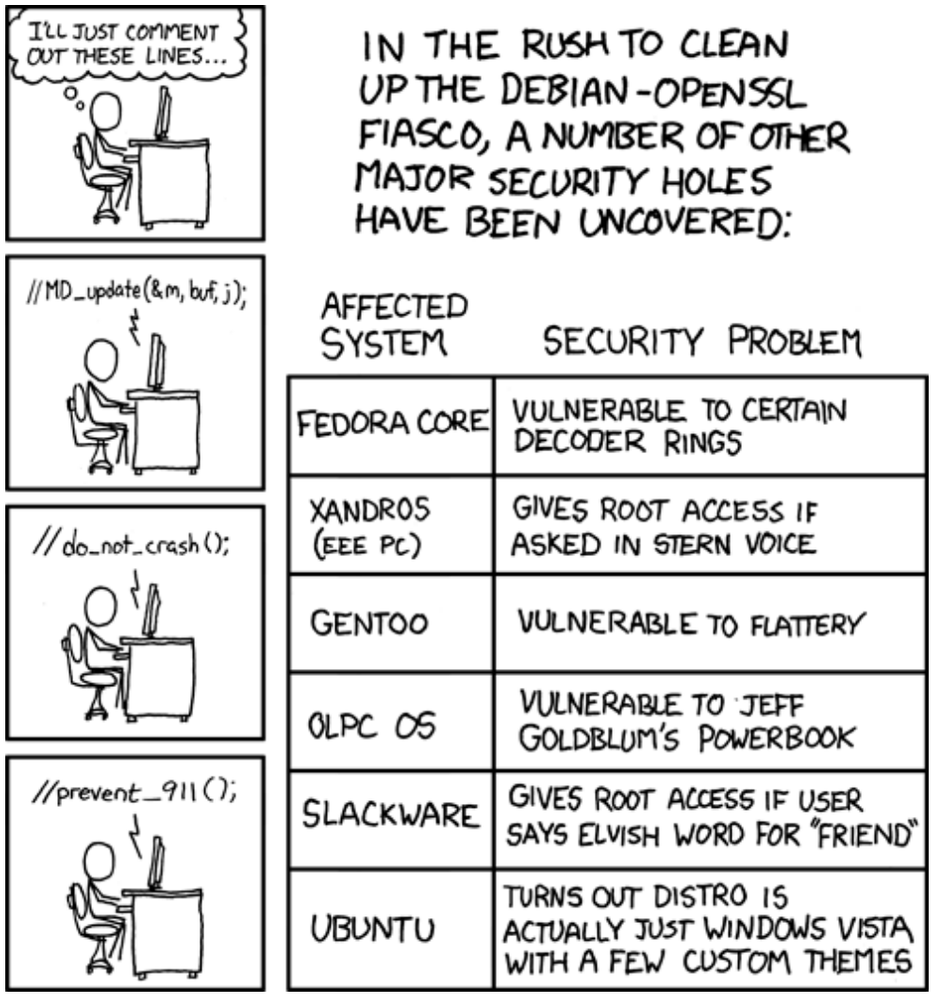
Security and Software Engineering - An Introduction -

Questões de Segurança em Engenharia de Software (QSES)
Mestrado em Segurança Informática
Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto

Eduardo R. B. Marques, edrdo@dcc.fc.up.pt

Motivation for this course



xkcd: "Security holes"

- "*The **disconnect between security and development has ultimately produced software development efforts that lack any sort of contemporary understanding of technical security risks**. Today's complex and highly connected computing environments trigger myriad security concerns, so by blowing off the idea of security entirely, software builders virtually guarantee that their creations will have way too many security weaknesses that could-and should have been avoided.*", in "*Bridging the gap between software development and information security*", KR van Wyk, G McGraw, IEEE Security and Privacy, 2005

- *"[this] "penetrate and patch" approach is not working: unpatched systems remain vulnerable, and even when they are the patched there are probably other latent vulnerabilities that remain. "Penetrate and patch" also doesn't address the new vulnerabilities that are introduced as the software evolves. **"So we need shift our mentality to building security in: We should aim to build software that is free of vulnerabilities (or far more likely to be free of them) right from the start."***, in "From 'Penetrate and Patch' to 'Building Security In'" by Michael Hicks, PL Enthusiast blog, 2015

Basic notions

Security ?

- A generic succinct definition:
  - “***Achieving some goal in the presence of an adversary.***” Computer Systems Security course @ MIT
- This means:
  - Having a **security policy** that defines the **security goals**
  - Estimated what the ***adversary*** might do & the associated risks — **threat modeling & risk assessment**
  - Develop a system in line the security goals and a threat model / risk assessment.
- **Q:** What can go wrong?

Things go wrong when …

- A system, in particular its software, is *vulnerable* regarding:
  - Security features (e.g., MITRE's database for openssl)
  - Bugs in standard code that lead to unintended functionality (e.g. buffer overflows, command injection, …)
- Security policy / threat modeling / risk analysis are inadequate:
  - Weak password recovery questions (e.g. "Sarah Palin's email hack")
  - Combined account info leakage (e.g. "The Epic Hacking of Mat Honan and Our Identity Challenge")
  - Trust all SSL Certificate Authorities (CAs) e.g. DigiNoar, Comodo attacks.
  - Assume that achines disconnected from the Internet are secure, e.g. the Stuxnet worm was originally injected in Iran's nuclear facilities via USB pen drives.

Security policy — typical goals

- The **“CIA triad”** (of information security):
  - **Confidentiality**: data is not made available or disclosed to ***unauthorized*** parties.
  - **Integrity**: data cannot be modified in an unauthorized or undetected manner.
  - **Availability**: data must be available when needed.
- Other important goals
  - **Non-repudiation**: data and operations over data should be traceable and verifiable - who did what and when?
  - **Privacy** *(!= confidentiality)*: data is subject to rights and obligations by all (authorized) parties that have access to it.

Threat modeling example / STRIDE

- **Threat model**: assumptions about what an attacker can do, i.e., assessment of possible threats.
- **STRIDE** is a popular threat model, e.g. used by the OpenStack consortium, and originally conceived by Microsoft, to classify threats in distinct categories:
  - **Spoofing** - impersonation of an entity (host, service, person, …)
  - **Tampering** - unauthorized data change / disruption.
  - **Repudiation** - unrecorded actions, even if they should be.
  - **Information disclosure**: extraction of information that should not be available
  - **Denial of service**: system becomes unavailable.
  - **Escalation**: elevation of privileges beyond what is expected.

STRIDE — examples

Threat #1 A malicious user views or tampers with personal profile data en route from the Web server to the client or from the client to the Web server. (Tampering with data/Information disclosure)
Threat #2 A malicious user views or tampers with personal profile data en route from the Web server to the COM component or from the component to the Web server. (Tampering with data/Information disclosure)
Threat #3 A malicious user accesses or tampers with the profile data directly in the database. (Tampering with data/Information disclosure)
Threat #4 A malicious user views the Lightweight Directory Access Protocol (LDAP) authentication packets and learns how to reply to them so that he can act "on behalf of" the user. (Spoofing identity/Information disclosure/Elevation of privilege [if the authentication data used is that of an administrator])

- Source: “Applying STRIDE - Commerce Server 2002”, Microsoft

Vulnerability classes

- The **CWE** (Common Weakness Enumeration) database
  - A community-curated classification of categories of software weaknesses
  - Example: CWE-77 - Command Injection
- For convenience, the CWE entries may be viewed for according to:
  - a taxonomy like 7PK (The Seven Pernicious Kingdoms)
    - “By organizing these errors into a simple taxonomy, we can teach developers to recognize categories of problems that lead to vulnerabilities and identify existing errors as they build software.” [original 7PF paper]
  - … or by relevance in a certain domain like the OWASP Top 10
    - ““focuses on identifying the most serious web application security risks for a broad array of organizations”

7PK view of the CWE database (fragment)

700 - Seven Pernicious Kingdoms

- 7PK - Security Features - *(254)*
- 7PK - Time and State - *(361)*
- 7PK - Errors - *(388)*
- 7PK - Input Validation and Representation - *(1005)*
  - Improper Input Validation - *(20)*
  - Improper Neutralization of Special Elements used in a Command ('Command Injection') - *(77)*
  - Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') - *(79)*
  - Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') - *(89)*
  - Improper Control of Resource Identifiers ('Resource Injection') - *(99)*
- 7PK - API Abuse - *(227)*
- 7PK - Code Quality - *(398)*
- 7PK - Encapsulation - *(485)*
- 7PK - Environment - *(2)*

OWASP Top 10 (fragment)

OWASP Top 10 - 2017

A1:2017-Injection

A2:2017-Broken Authentication

A3:2017-Sensitive Data Exposure

A4:2017-XML External Entities (XXE) [NEW]

A5:2017-Broken Access Control [Merged]

A6:2017-Security Misconfiguration

A7:2017-Cross-Site Scripting (XSS)

A8:2017-Insecure Deserialization [NEW, Community]

A9:2017-Using Components with Known Vulnerabilities

A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

The CVE database

- The **CVE** (Common Vulnerabilities & Exposures) database
  - Exposures / vulnerabilities found in concrete software
  - Example: CVE-2018-1000802 — command injection vulnerability in CPython
- In line with the CWE/CVE spirit and terminology:
  - **Vulnerability**: flawed computational logic in a system that compromises security.
  - **Exposure**: misconfiguration/code in a system that does not directly compromise security but may aid in exposing a system to attack.
  - **Weakness**: general characterization of a set of vulnerabilities in different
- Most often we will use the term vulnerability even if referring to exposure or weakness in the strict sense.

Risk assessment — DREAD

- Risk assessment deals with quantifying the risk posed by identified vulnerabilities and the associated threats.
- The **DREAD** model — assess risk 0-10 in regard to :
  - **Damage potential** - how much damage will be caused?
  - **Reproducibility** - how easy is it to reproduce the attack?
  - **Exploitability** - how difficult is it to launch an attack?
  - **Affected users** - how many users are affected?
  - **Discoverability** - how easy is it to discover the vulnerability?
  - ***Risk level* =** average of all 5 factors.

DREAD example

OSSA 2014-038

(from openstack.org)

- Title: List instances by IP results in DoS of nova-network
- Link: https://bugs.launchpad.net/ossa/+bug/1358583
- Importance Assigned: Medium

Summary

On a customer install which has approximately 500 VMs in the system, running the following will hang:
nova list --ip 199

What will happen afterwards is that the nova-network process will stop responding for a while, a trace shows that it's receiving a huge amount of data.

Dread Score

Potential for: Denial of Service		
Category	**Score**	**Rationale**
Damage	5	Significant Disruption to API/Management
Reproducability	8	Code path is easily understood, condition exists as standard
Exploitability	8	Exploitable with cURL, Authentication Required
Affected Users	4	All Nova API users
Discoverability	10	Discoverability always assumed to be 10
DREAD SCORE: 35/5 = 7 Critical		

Risk assessment — CVSS

- **Common Vulnerability Scoring System (CVSS)**
  - A framework by used to score the risk level of vulnerabilities in the CVE database.
- **CVSS scores**
  - **Base score:** relates to the intrinsic characteristics of a vulnerability, usually taken as the main reference for risk level.
  - **Temporal:** changeable over the lifetime of a vulnerability, accounts for possible remediation & evolution of characteristics in time. **[*Q: What do you think is the lifetime of a vulnerability?]***
  - **Environmental** score: accounts for environmental aspects, typically from a deployment perspective.

CVSS — base score metrics

Source: [CVSS V3 calculator]

Exploitability Metrics

Attack Vector (AV)*

Network (AV:N)	Adjacent Network (AV:A)	Local (AV:L)	Physical (AV:P)

Attack Complexity (AC)*

Low (AC:L)	High (AC:H)

Privileges Required (PR)*

None (PR:N)	Low (PR:L)	High (PR:H)

User Interaction (UI)*

None (UI:N)	Required (UI:R)

Scope (S)*

Unchanged (S:U)	Changed (S:C)

Impact Metrics

Confidentiality Impact (C)*

None (C:N)	Low (C:L)	High (C:H)

Integrity Impact (I)*

None (I:N)	Low (I:L)	High (I:H)

Availability Impact (A)*

None (A:N)	Low (A:L)	High (A:H)

CVSS v3 Equations

The CVSS v3.0 equations are defined below.

Base

The Base Score is a function of the Impact and Exploitability sub score equations. Where the Base score is defined as,

$$
\begin{array}{ll}
\textit{If (Impact sub score <= 0)} & \textit{0 else,} \\
\textit{Scope Unchanged}_4 & \textit{Roundup}(\textit{Minimum}[(\textit{Impact} + \textit{Exploitability}), 10]) \\
\textit{Scope Changed} & \textit{Roundup}(\textit{Minimum}[1.08 \times (\textit{Impact} + \textit{Exploitability}), 10])
\end{array}
$$

and the Impact sub score (ISC) is defined as,

$$
\begin{array}{l}
\textit{Scope Unchanged}\ 6.42 \times \textit{ISC}_{\text{Base}} \\
\textit{Scope Changed}\ 7.52 \times [\textit{ISC}_{\textit{Base}} - 0.029] - 3.25 \times [\textit{ISC}_{\textit{Base}} - 0.02]^{15}
\end{array}
$$

Common misconceptions and guiding principles

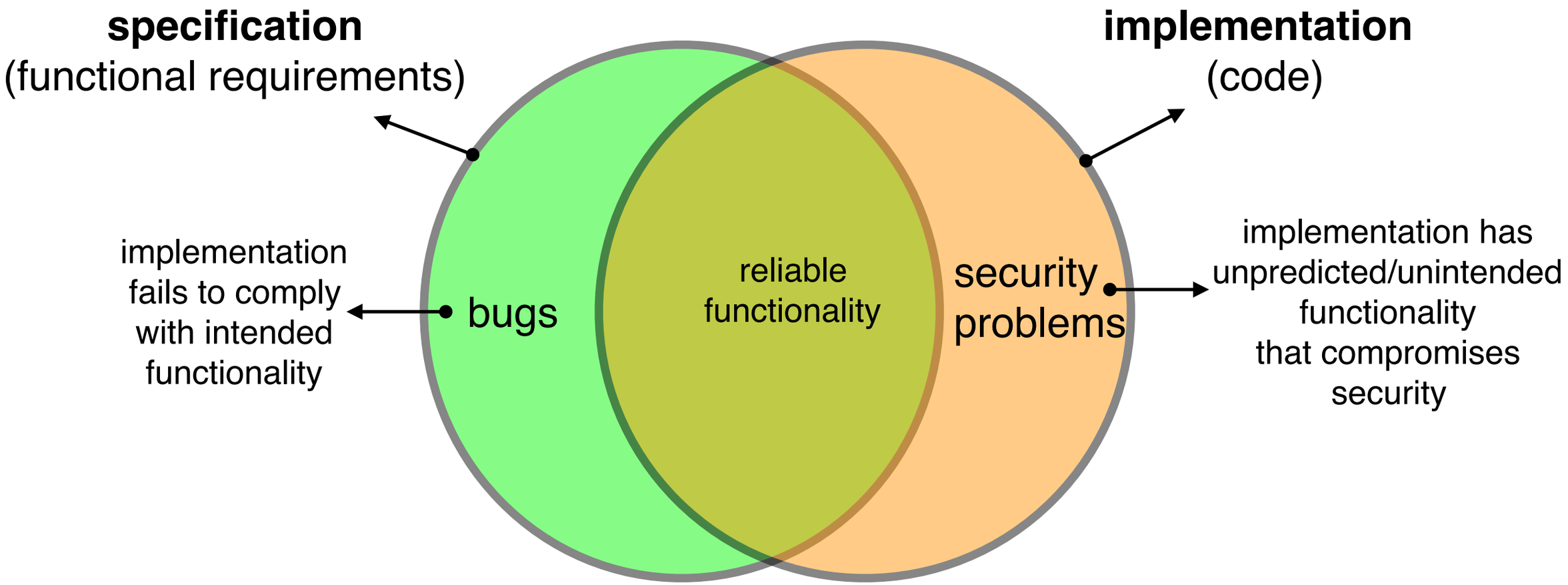
What do you think?

- Our software is perfectly secure, because:
  - It works so reliably, meeting all functional requirements.
  - It has all the appropriate security features.
  - The possible use/deployments are well known and under control.
  - We anticipated all threats.
  - The code is closed-source and the binary releases are obfuscated. Our cryptography mechanisms are also secret.
  - For security, we conduct extensive pen-testing at the end of the software development cycle.
  - We just released a patch that fixed all the security issues, and it is protected against every known exploit.

What do you think?

- Let us discuss why all of these are serious misconceptions, and introduce guiding principles:
  - It works so reliably, meeting all functional requirements.
    - ✦ **Reliability != Security**
  - It has all the appropriate security features.
    - ✦ **Security Features != Secure Features**
  - The possible use/deployments are well known and under control.
    - ✦ **“The Trinity of Trouble”**
  - We anticipated all threats.
    - ✦ **“The attacker’s advantage and the defender’s dilemma"**
  - The code is closed-source and the binary releases are obfuscated. Our cryptography mechanisms are also secret.
    - ✦ **Security by design vs security by obscurity**
  - For security, we conduct extensive pen-testing at the end of the software development cycle.
    - ✦ **SLDC and “the touchpoint model”**
  - We just released a patch that fixed all the security issues, and it is protected against every known exploit.
    - ✦ **Window of vulnerability, zero-day exploits**

Reliability != Security



- ***"Reliable software does what it is supposed to do. Secure software does what is supposed to do, and nothing else."*** — Ivan Arce, cited by Chess and West
- It's now only how the software is supposed to be ***used*** … but also how it can be ***abused*** !
- Functional requirements are driven by ***use cases***.
- Security requirements are also driven by ***abuse cases***, a concept we will came back to later.

Reliability != Security (2)

```
<html>
 <head>
  <title>PHP Hello program</title>
 </head>
 <body>
 <?php echo ‘Hello ‘ . $_GET[“X”]’; ?>
 </body>
</html>
```

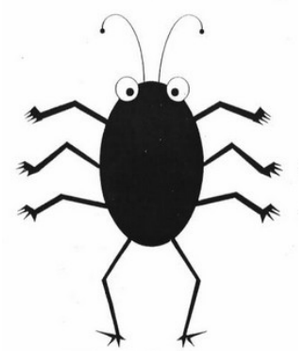
- **Specification***: given* (GET request parameter) X reply with "Hello X !”.
- **Implementation**: reliable, but not secure … why?

Reliability != Security (3)

```
<html>
 <head>
  <title>PHP Hello program</title>
 </head>
 <body>
 <?php echo ‘Hello‘ .  htmlspecialchars($_GET[“X"])’; ?>
 </body>
</html>
```

- The security issue is that arbitrary HTML could be passed in X, in particular it can inject a script that is executed in the client browser, the typical strategy for a cross-site scripting attack.
- Defense mechanism — **input sanitization**: **htmlspecialchars** sanitizes the input by “escaping” HTML characters, preventing code execution.

Historical note on the term "bug"



"'Bug'—as such little faults and difficulties are called—show themselves, and months of anxious watching, study, and labor are requisite before commercial success—or failure—is certainly reached." [Thomas Edison, 1878]

"Did You Know? Edison Coined the Term "Bug", A. Magoun and P. Israel, The Institute, IEEE. 2013.
http://theinstitute.ieee.org/technology-focus/technology-history/did-you-know-edison-coined-the-term-bug

Historical note on the term "bug"

9/9

0800 Arctan started
1000 " stopped - arctan ✓ { 1.2700 9.037 847 025
9.037 846 795 correct
13" wc (032) MP - MC ~~1.9826447000~~ ~~2.13047641515~~) 4.615925059(-2)
(033) PRO 2 2.13047641515
correct 2.13067641515
Relays 6-2 in 033 failed special speed test
In relay " " 11.00 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Multy Adder Test.
1545 Relay #70 Panel F
(moth) in relay.

First actual case of bug being found.
1700 closed down.

"First actual case of bug being found" [!!]

Note in Harvard Mark II logbook by Grace Hopper, 1947 [actual moth (bug) part of the logbook], U.S. Naval Historical Center Online Library Photograph

Security features != Secure features

700 - Seven Pernicious Kingdoms

- 7PK - Security Features - *(254)*
- 7PK - Time and State - *(361)*
- 7PK - Errors - *(388)*
- 7PK - Input Validation and Representation - *(1005)*
- 7PK - API Abuse - *(227)*
- 7PK - Code Quality - *(398)*
- 7PK - Encapsulation - *(485)*
- 7PK - Environment - *(2)*

- Security features
  - Features that are related directly to security goals like the use of cryptography, password handling, access control, etc.
  - They should of course be conceived carefully, but development cannot focus on security features alone.
- “Security Features” are just one of the “kingdoms” in 7PF. Why ?
- Secure features
  - Any feature, even if not directly related to a security requirement or mechanism, may pose security at risk.

The “Trinity of Trouble” (ToT)

- We cannot anticipate all possible use/deployments of a software system.
- Modern software is subject to the **Trinity of Trouble (ToT)**, a term introduced by Gary McGraw:
  - **Connectivity**: software systems are connected
  - **Complexity**: their organization can be intricate and complex
  - **Extensibility:** they evolve and can be extended in unpredictable manner
  - All these aspects are naturally inter-related.
- Let us look at a simple “tutorial” example.

ToT - An example (1)

```
<html> <head
<script>
function showUser(str) {
if (str=="") {
  document.getElementById("txtHint").innerHTML="";
  return;
}
if (window.XMLHttpRequest) {// code for IE7+, Firefo
   xmlhttp=new XMLHttpRequest();
} else {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.onreadystatechange=function()  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200) {
     document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
  }
}
xmlhttp.open("GET","getuser.php?q="+str,true);
xmlhttp.send();
}
</script>
</head>
<body>
<form>
<select name="users" onchange="showUser(this.value)">
<option value="">Select a person:</option>
<option value="1">Peter Griffin</option>
<option value="2">Lois Griffin</option>
<option value="3">Glenn Quagmire</option>
<option value="4">Joseph Swanson</option>
</select>
</form>
<br>
<div id="txtHint"><b>Person info will be listed here.</b></div>

</body>
</html>
```

begin HTML

A Javascript method invoked by the HTML form below

Use of the AJAX Javascript API
HTTP communication
and XHTML/XML formats are implicit

invocation of server side PHP script; use of PHP revealed

static HTML form

dynamic HTML section

end HTML

A “simple” example
“PHP - AJAX and MySQL”
http://www.w3schools.com/php/php_ajax_database.asp

ToT — An example (2)

```php
<?php
$q=$_GET["q"];
$con = mysql_connect('localhost', 'peter', 'abc123');
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
mysql_select_db("ajax_demo", $con);

$sql="SELECT * FROM user WHERE id = '".$q."'";
$result = mysql_query($sql);

echo "<table border='1'>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>Age</th>
<th>Hometown</th>
<th>Job</th>
</tr>";
while($row = mysql_fetch_array($result))
  {
  echo "<tr>";
  echo "<td>" . $row['FirstName'] . "</td>";
  echo "<td>" . $row['LastName'] . "</td>";
  ...
  }
echo "</table>";
mysql_close($con);
?>
```

DB connection
Hard-coded credentials!

DB query
SQL injection possible!

generation of dynamic HTML

Use of possibly unsanitized database data

A “simple” example
“PHP - AJAX and MySQL”
http://www.w3schools.com/php/php_ajax_database.asp

ToT — An example (3)

```php
<?php
$q=$_GET["q"];
$con = mysql_connect('localhost', 'peter', 'abc123');
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
mysql_select_db("ajax_demo", $con);

$sql="SELECT * FROM user WHERE id = '".$q."'";
$result = mysql_query($sql);
```

DB connection
Hard-coded credentials!

DB query

- In summary — Only a simple example (intended as a tutorial!) … but one that illustrates how much of modern software is developed:
  - everything mixed / low modularity, different languages/formats (PHP, Javascript, SQL, HTML, XML)
  - wide attack surface helped by intricate/fragile client-side + server-side + database dependencies
  - little security concerns

```php
?>
```

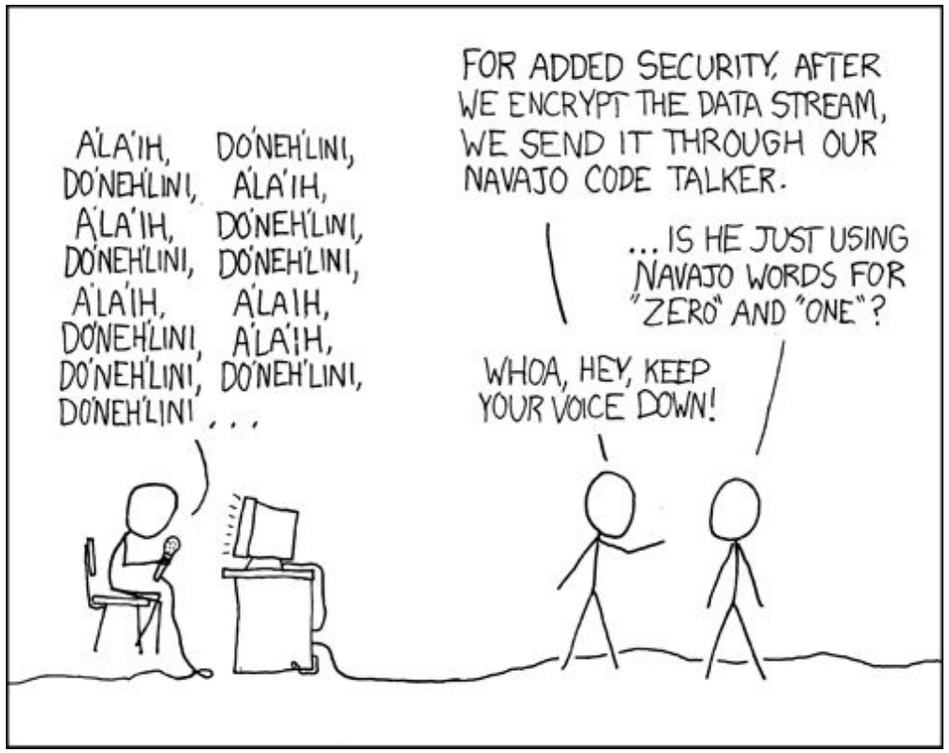
ToT — the BlueBorne case

- BlueBorne is an attack vector directed at Bluetooth devices, just reported by Armis (Sep. 2017)
  - can target most devices (billions of them!)
  - lead to control of devices, access corporate data and networks, network penetration, and malware spread …
  - 8 vulnerabilities can be combined or used in isolation to perform an attack
- “So, what seems to be the problem?” section in the Armis white paper
  - “*Bluetooth is **complicated**. Too complicated. Too many specific applications are defined in the stack layer, with e**ndless replication of facilities and features.** These over-complications are a direct result of the immense work, and **over-engineering** that was put into creating the Bluetooth specification. […]*”
  - *“Bluetooth’s **complexity** kept researchers from auditing its implementations at the same level of scrutiny that other highly exposed protocols, and outwards-facing interfaces have been treated with.”*
  - *“The **complications** in the specifications translate into **multiple pitfall junctions** in the various implementations of the Bluetooth standard.”*

The attacker’s advantage and the defender’s dilemma

- We just cannot anticipate all possible threats!
- Howard & Leblanc summarize the problem in 4 principles:
  - *“Principle #1: The defender must defend all points; the attacker can choose the weakest point.”*
  - *“Principle #2: The defender can defend only against known attacks; the attacker can probe for unknown vulnerabilities.”*
  - *“Principle #3: The defender must be constantly vigilant; the attacker can strike at will.”*
  - *“Principle #4: The defender must play by the rules; the attacker can play dirty.”*
- For instance consider this report on automated malware generation:
  - “the automation of malware production means that attackers can generate and propagate malicious software at lightning speed, outpacing the efforts of human security teams to identify and block new variants of threats” from "Dark Trace Global Threat Report 2017, Selected Case Studies"

Security by obscurity



Code talkers, xckd

- So: “The code is closed-source and the binary releases are obfuscated. Our cryptography mechanisms are also secret.”
- **Security by obscurity.** relies on secrecy as a general method for security.
- It works as a deterrent / increased work factor for an adversary:
  - Secrets are hard to keep for a long time (e.g. consider leaks, reverse engineering techniques).
  - Also in general, recall the “attacker’s advantages” principles

Security by design

- Instead we should seek that a **system should be secure by design.**
- Two famous guidelines from the realm of cryptography, that should be taken to secure software in general:
  - **Kerckhoffs's principle**: *“a cryptosystem should be secure even if everything about the system, except the key, is public knowledge”*
  - **Shannon’s maxim**: *“one ought to design systems under the assumption that the enemy will immediately gain full familiarity with them”*
- Design principles?

Security by design — some example principles by Viega and McGraw

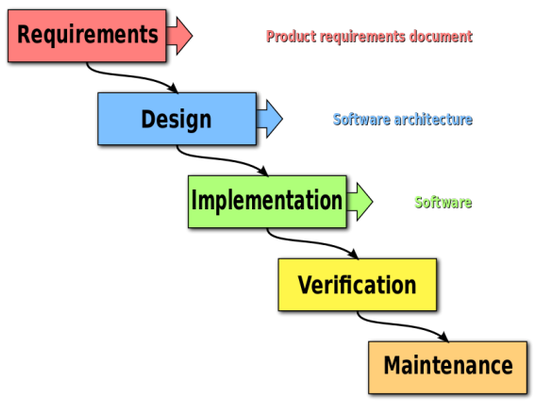
- **1. Secure the weakest link**
  - Security defense should be seen as a chain, attackers will look for the weakest link in that chain.
- **2. Defense in depth**
  - Manage risk by defense at all layers / components, such that if one of them fails, the other has a fair chance.
- **3. Fail securely**
  - Handle failures correctly & securely; failure handling is often overlooked in reliability and security terms.
- **4. Least privilege**
  - Execute software with the minimum required privileges to mitigate possible impact of an attack (e.g. do not run servers with super-user privileges)
- **5. Compartmentalize**
  - Try to limit the damage by compartmentalizing (e.g. using VPNs, containers, firewalls …)
- **6. Keep It Simple (KISS) !!!**
- **…**

[Viega and McGraw, Building Secure Software, chapter 5]

The software development life cycle (SDLC) and security

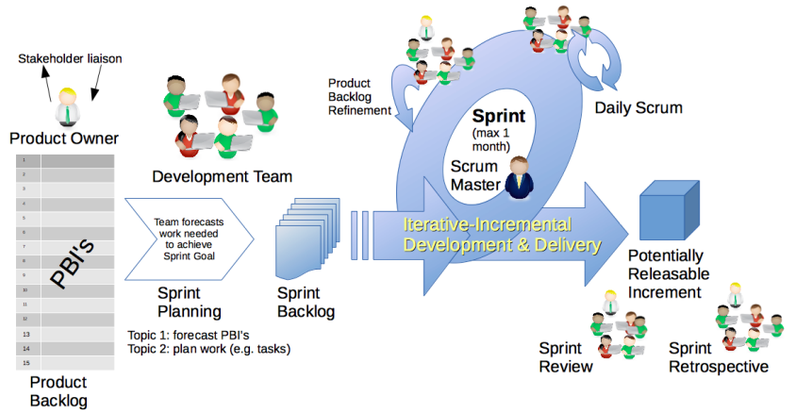
Images from Wikipedia

Waterfall



XP

SCRUM



- The SLDC Involves different:
  - "philosophies" — more or less "agile" …
  - stages (typically in a feedback loop) & related artifacts: requirements, design, code, tests, deployments, …
  - people: "architects", project managers, programmers, testers, QA people, …
- So: "For security, we conduct extensive pen-testing at the end of the software development cycle."
  - The "penetrate and patch model" that Michael Hicks refers to (slide 2)…
  - Pen-testing is useful but only one of the possible security-driven activities in the SDLC.

Security touch-points

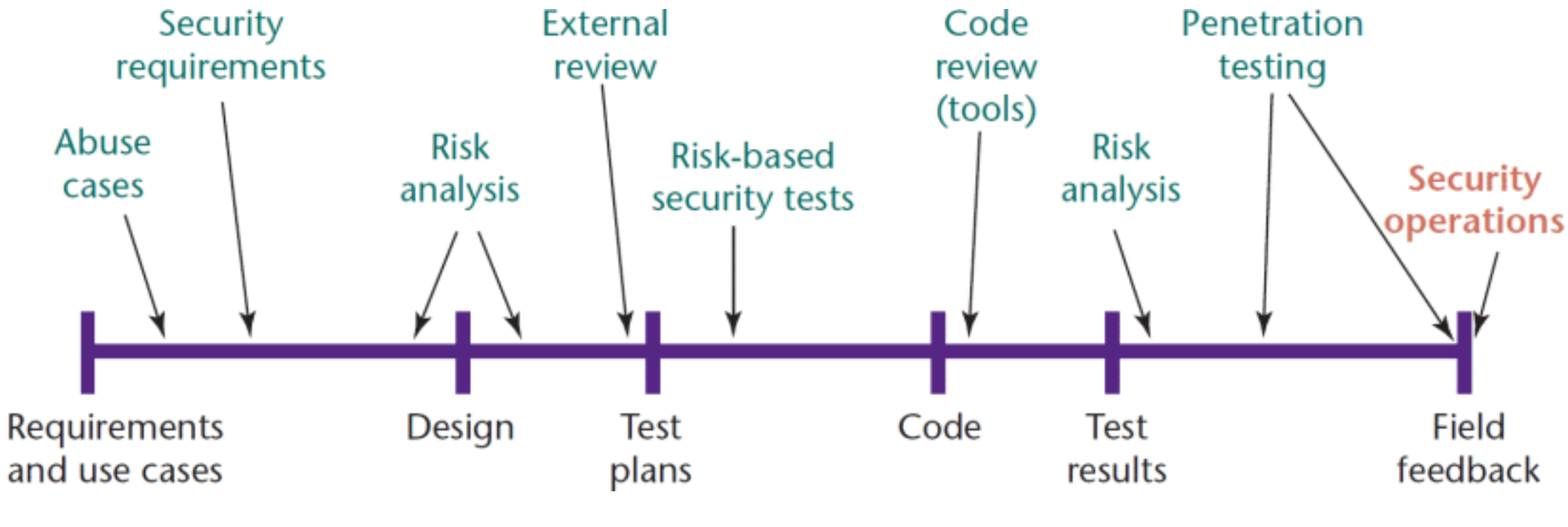


Image source: “Software security”, G. McGraw, IEEE SECURITY & PRIVACY

- **Security is a *cross-cutting* concern and an *emergent* property; it must be accounted for during the entire SDLC.**
- With that in mind, Gary McGraw proposed the influential touch-point model.
- The idea is that touch-points define security-oriented tasks for different stages of the SLDC.
- Detailed touch-points have been identified and organized by initiatives such as BSIMM and OpenSAMM.
- *We will cover the touchpoint model in the next class and go through some of the touch-points in detail throughout the semester.*

Window of vulnerability

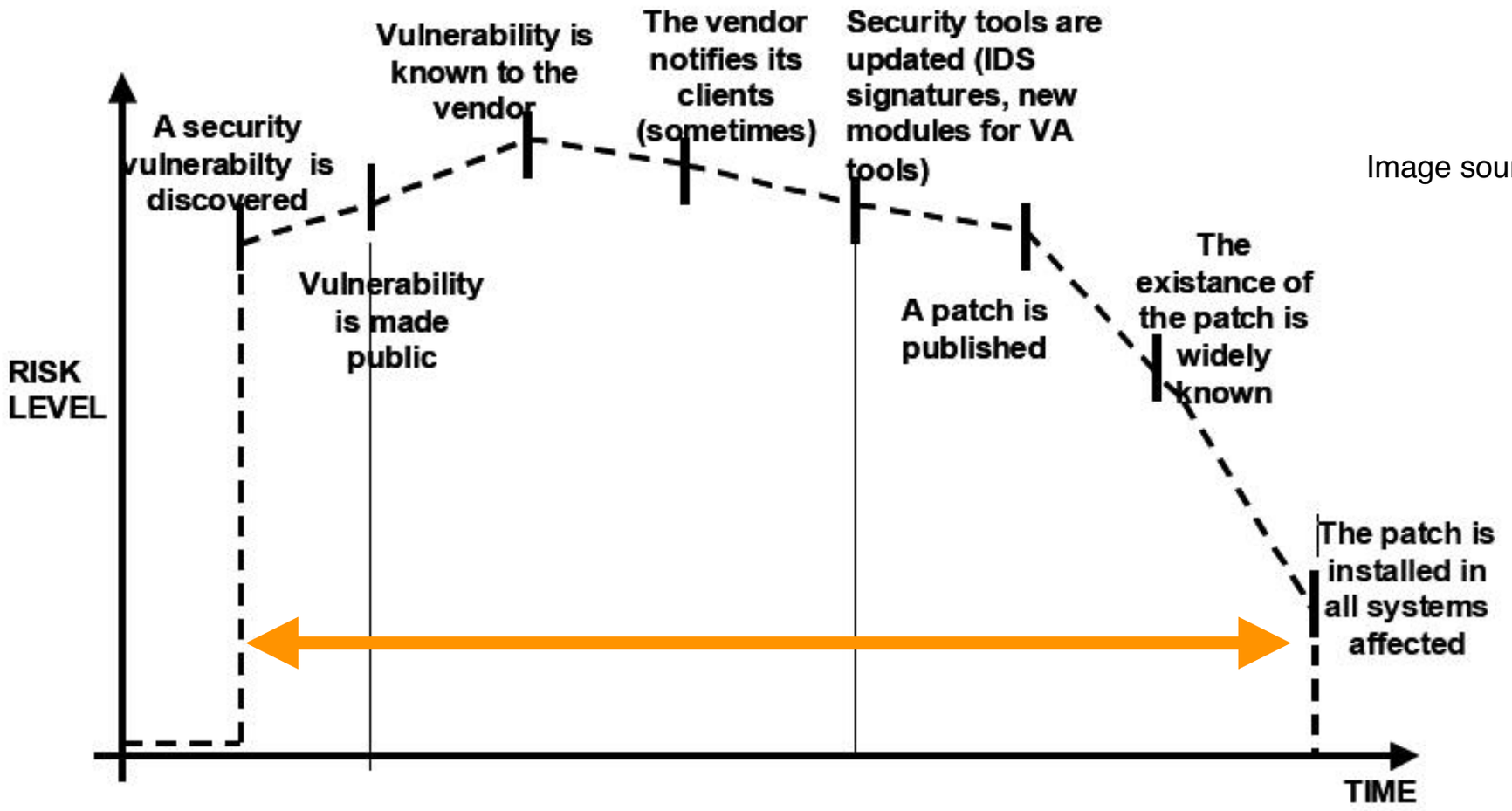


Image source: OWASP Testing Guide introduction

- So: “We just released a patch that fixed all the security issues, and it is protected against every known exploit.”
- **Window of vulnerability**: time period of exposure to a vulnerability & associated exploits
  - It starts when the vulnerability is discovered, by attackers or a security analyst
  - It continues over a period of time during which the vulnerability is publicized (by itself potentially triggering attacks) and a patch is developed to fix the problem.
  - It ends only when all affected systems are effectively patched.

The Equifax data breach

- "*Equifax […] lost control of customer data that included Social Security numbers, home addresses, credit card numbers, drivers license numbers and birth dates. The company estimates that the data of 143 million people were exposed, which equals roughly half the US population.*"
  - From: "Your guide to surviving the Equifax data breach", Sharon Profis, CNET
- "***The flaw in the Apache Struts framework was fixed on March 6. Three days later, the bug was already under mass attack by hackers […] Equifax has said the breach on its site occurred in mid-May, more than two months after the flaw came to light and a patch was available.***"
  - From: "Failure to patch two-month-old bug led to massive Equifax breach", Dan Goodwin, ArsTechnica.com, 14/09/2007
- CVE-2017-5638: "*The Jakarta Multipart parser in Apache Struts 2 2.3.x […] allows remote attackers to execute arbitrary commands via a #cmd= string in a crafted Content-Type HTTP header, as exploited in the wild in March 2017. "*
  - From: MITRE vulnerability database

Zero-day vulnerabilities

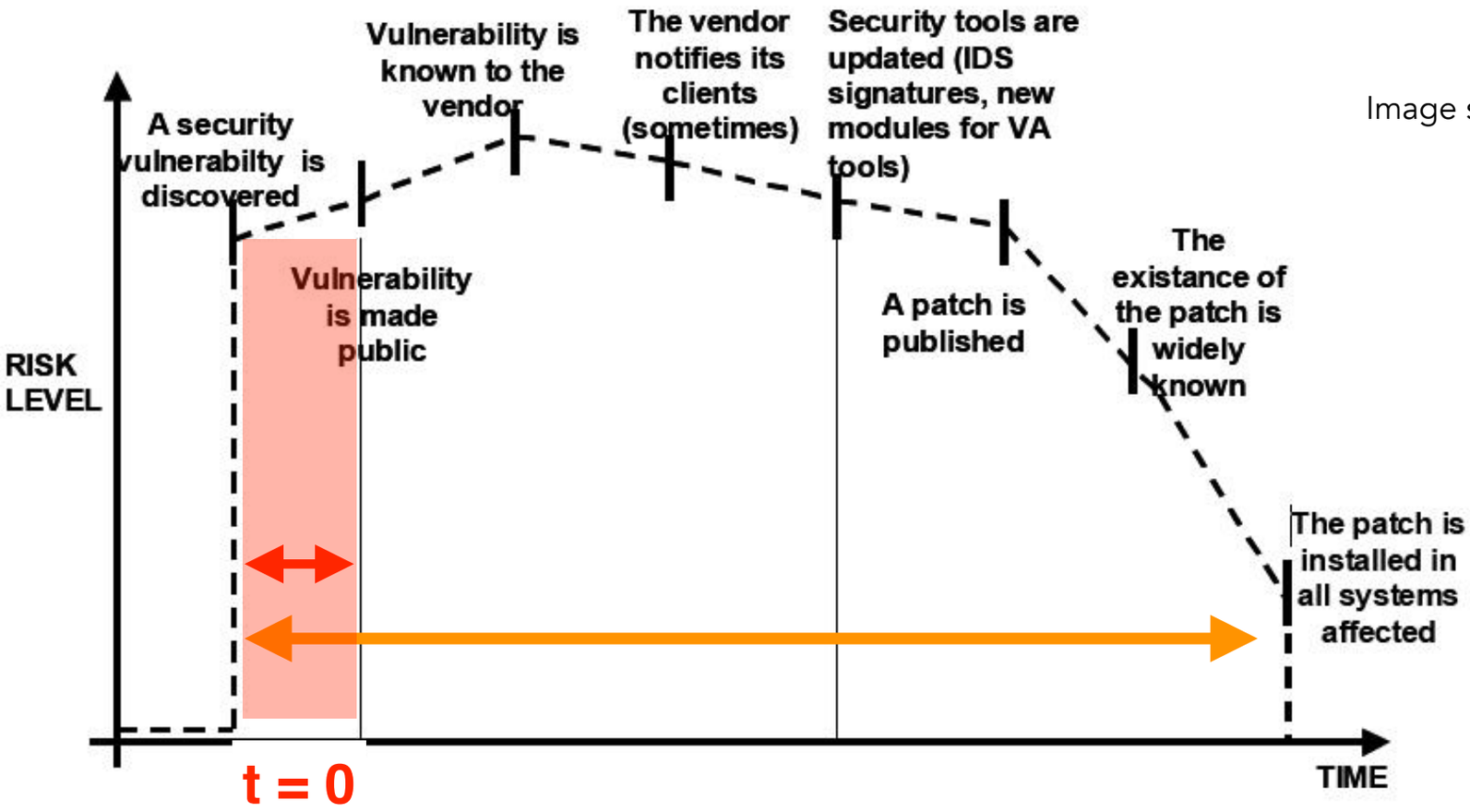


Image source: OWASP Testing Guide introduction

- ***t*-day vulnerability (exploit)** : a vulnerability hat has been known for *t* days (Equifax case: t = 60)
- Note that **t *may be 0 for a possibly long time!*** **Zero-day vulnerability:** the vulnerability has not been disclosed by whoever found it … in effect it "does not exist" and no time has passed for countermeasures.
- The hunt for zero-day vulnerabilities promotes both good and bad initiatives
  - Zero Day Initiative
  - "New Dark-Web Market Is Selling Zero-Day Exploits to Hackers", Wired article