Security Requirements, Risk Assessment & Penetration Tests (Lecture 10)

Prof. Mathy Vanhoef

DistriNet – KU Leuven – Belgium

KU LEUVEN DistriNet

Secure Requirements Management

Overview

- Requirements engineering is a discipline by itself.
  - Method to define/manage/prioritize/.. your requirements
  - We will only scratch the surface in this lecture.
- **Security requirements collection** should be integrated in the development lifecycle
- **SQUARE**: Security Quality Requirements Engineering
  - Defines 9 steps to manage security requirements
  - Done by team of engineers and the project's stakeholders
  - Stakeholders = the client or the client's organization

1. Agree on Definitions

Agree on a set of terminology and definitions

- Important because the team that defines requirements is diverse (internal developers, requirements engineers, clients)
- Ambiguity in the detailed associated to a term can also vary
  - E.g., “access controls” can mean a set of policies that govern who can access what, but others can interpret this term as the software elements that implement this functionality.
- Can reuse public definitions (Wikipedia, IEEE, books, etc.)

2. Identity Security Goals

Agree on prioritized security goals for the project

- Rough guideline: have around 6 security goals for the project and prioritize them. Amount depends on size of the project.
- Must be in clear support of the project's overall business goal

Can now identify priority & relevance of security requirements

- E.g., HR is concerned with privacy of personnel data, while finance want to assure financial data cannot be modified
- This step resolves such conflicts

3. Collect or Create Artifacts

Selected examples of artifacts:

- System architecture
- **Use cases**: user story + actors that interacts with the system
- **Misuse cases**: helps models & visually represent things from an attacker’s point of view
- **Abuse cases**: represents security requirements from a much stronger destruction aspect of the system

4. Perform Risk Assessment

Identifies threats facing the system

- Including likelihood of the threats & potential consequences
- Define what is the sensitive information to protect

Important so the defined security requirements make sense

- E.g., otherwise, stakeholders may use encryption of data at rest without understanding the problem that encryption solves
- Risk assessment also helps to prioritize security requirements

5/6. Select Elicitation technique & use it

Select an *elicitation technique* to collect security requirements:

- Ensure it can adapt to the number & expertise of stakeholders.
- Example: Accelerated Requirements Method (ARM)

Execute the select elicitation technique

- Elicited requirements must be verifiable/quantifiable once the project has been implemented
- Assure the security requirements say *what* the system should do, *not how* it should be done

7/8. Categorize & Prioritize Requirements

Common example categories:

- Essential or non-essential
- System level, software level, or as architectural constraints

Prioritize the security requirements:

- Based on results of step 4 (risk assessment)
- Structured prioritization methods exist, but often done using unstructured discussions between stakeholders
- Helps choose requirements to implement & in what order

9. Requirements Inspection

Find any defects in the requirements

- E.g., ambiguities, inconsistencies, or mistaken assumptions

Verify that each security requirement is verifiable, in scope, within financial means, and feasible to implement.

- Requirements in conflict with this should have never been added… this is a last chance to remove them

Other best practices

- Separate security requirements from functional requirements
  - Makes it possible to explicitly review & test security requirements
- Write requirements for industry standards & regulatory rules
  - HIPAA, GDPR,…
- Consider denial-of-service attacks
- Implementing security functionality might result in new assets that must be protected:
  - Cryptographic keys, TLS configurations, etc.

Secure the build and deployment pipeline

Motivation

Securing your build environment is often overlooked

- Yet supply chain attacks are becoming more common
- E.g., “Material Tailwind library is being impersonated for an apparent supply chain attack targeting developers”
- What if a malicious entity gets access to your code base and can manipulate it? Can you detect/contain the damage?

→ The UK NCSC defines 10 principles to follow

Summary of the 10 principles

1. Use a pipeline you trust
   - Must trust administrator and underlying infrastructure.
   - Use cryptographic signing and verification of code
2. Peer review code before deployment
   - Accept or reject according to the development processes.
   - Implement technical controls to prevent this from being bypassed.
3. Control how deployments are triggered
   - Ensure only the trusted code base can be used to deploy to production. Have clear processes in place who can manage this.

Summary of the 10 principles

4. Run automatic testing as part of your deployments
   - Include automated testing in the deployment pipeline
   - Fast tests on every commit. Slow tests on notable releases.

5. Carefully manage secrets and credentials
   - Only authorized employees should have access to the keys to managed to deployment pipeline

6. Prevent control bypasses
   - Ensure nobody can bypass certain steps (e.g., bypass testing)

Summary of the 10 principles

7. Avoid "self policing“
   - Pipeline enforces if code is accepted or rejected before deployment
   - Should be impossible for developer to modify these rules.
8. Be cautious of untrusted branches & pull requests
9. Be cautious of 3rd party libraries and updates
   - Ensure only intended dependencies are included and that they come from legitimate sources. Ensure latest versions are used.
10. Consider hard breaks and approval
    - Consider manual approval step before publishing public releases.

Risk assessment

Recap: qualatitive risk assessment

- Use a ranking system for **likelihood and impact**
- Rely on qualitative measures and then draw a **risk assessment matrix**
- Their combination can be assigned a risk level. Example:

C (cost or impact)	*P* (probability)				
	V.LOW	LOW	MODERATE	HIGH	V.HIGH
V.LOW (negligible)	1	1	1	1	1
LOW (limited)	1	2	2	2	2
MODERATE (serious)	1	2	3	3	3
HIGH (severe or catastrophic)	2	2	3	4	4
V.HIGH (multiply catastrophic)	2	3	4	5	5

Recap: quantative risk assessment

- Measuring the identified risk: **risk = impact * likelihood**
- **Impact**: what would happen if confidentiality, integrity, or availability of assets get compromised?
- **Likelihood**: consider the attacker's required:
  - Access to the system (local vs remote)
  - Skills, expertise, and knowledge
  - Motivation (fame, financial gain, IP, anger)
  - Budget

Alternative: DREAD by Microsoft

Rate, compare, and prioritize the severity of threats by assigning a given issue a rating between 0 and 10 for:

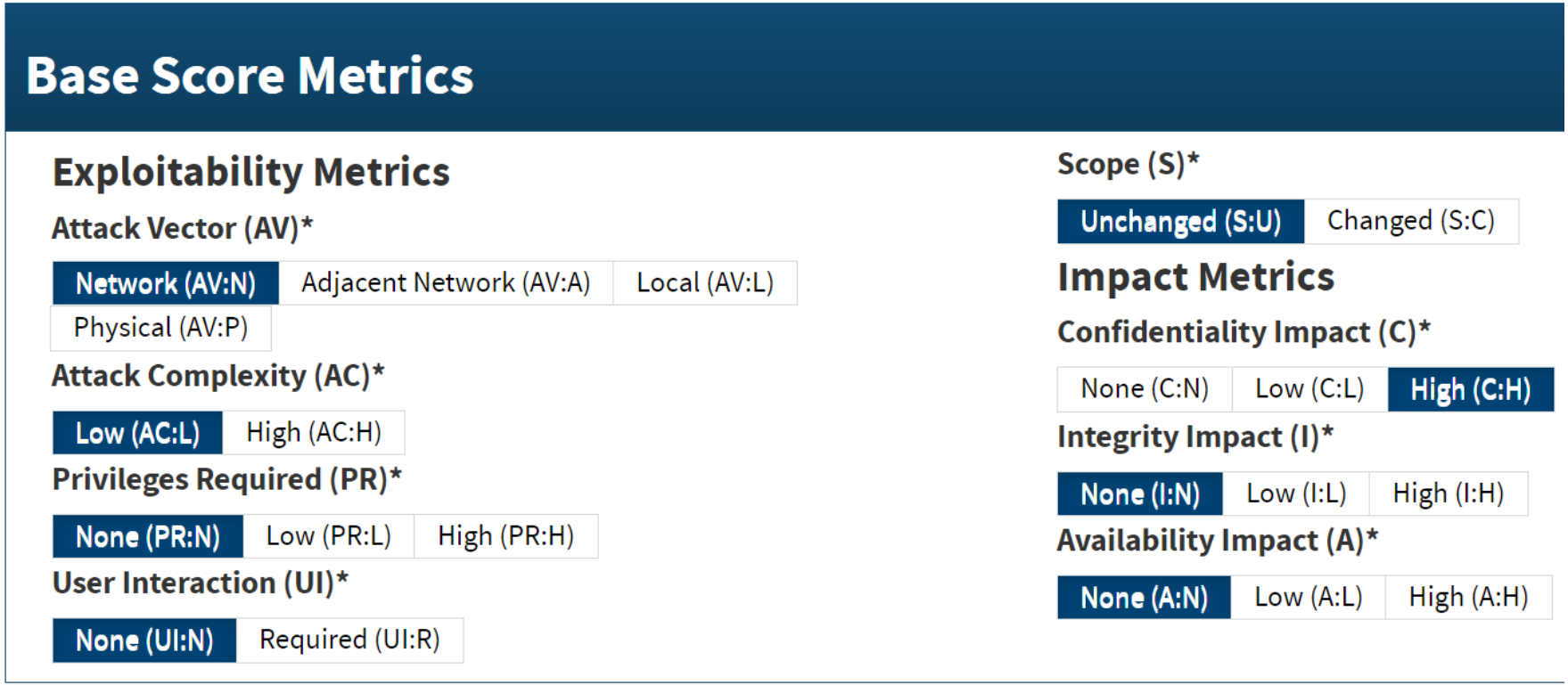
- **Damage** that could result from an attack (e.g., data loss)
- **Reproducibility** of the attack
- **Exploitability**: effort / expertise required to perform an attack
- **Affected** users: number of affected users
- **Discoverability**: likelihood that a threat will be exploited

→ Total score = average of all scores

CVSS: Common Vulnerability Scoring System

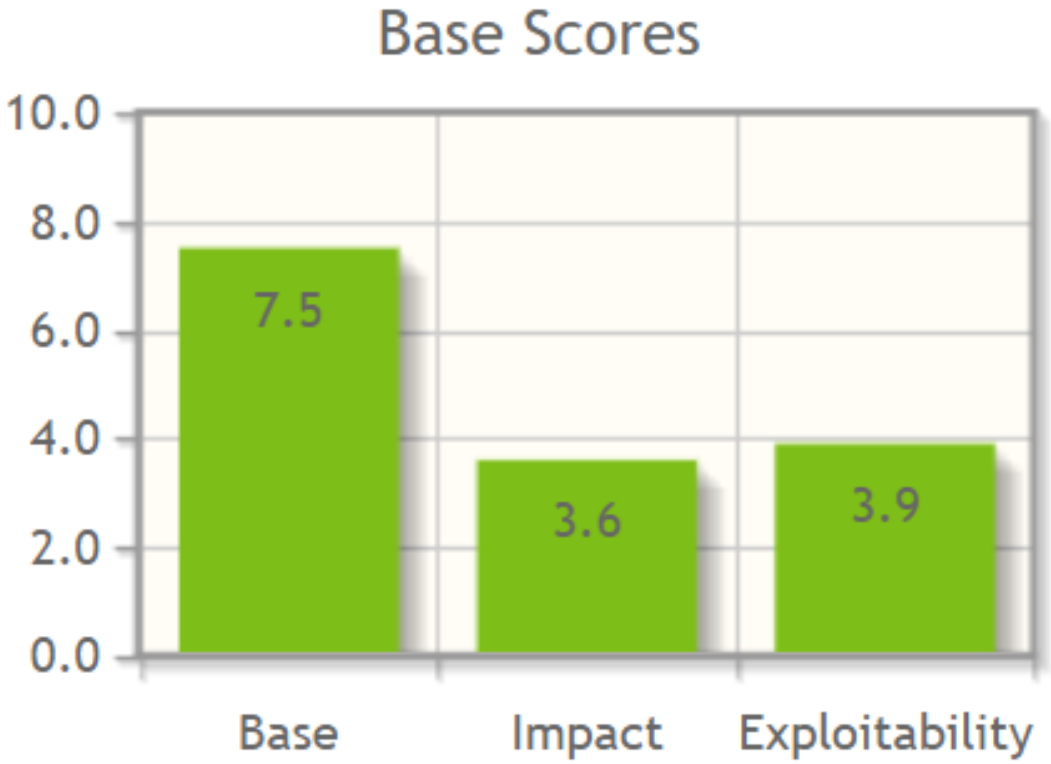
- Considers **already discovered vulnerabilities**
- Scores severity of vulnerabilities to prioritize them
- Combination of three metrics:
  - **Base metrics**: properties intrinsic to the vulnerability that will not change over its lifetime (base properties, impact, exploitability).
  - **Temporal metrics**: maturity of known exploits and defenses
  - **Environmental metrics**: adjust score to the specific environment and circumstances of the vulnerable system

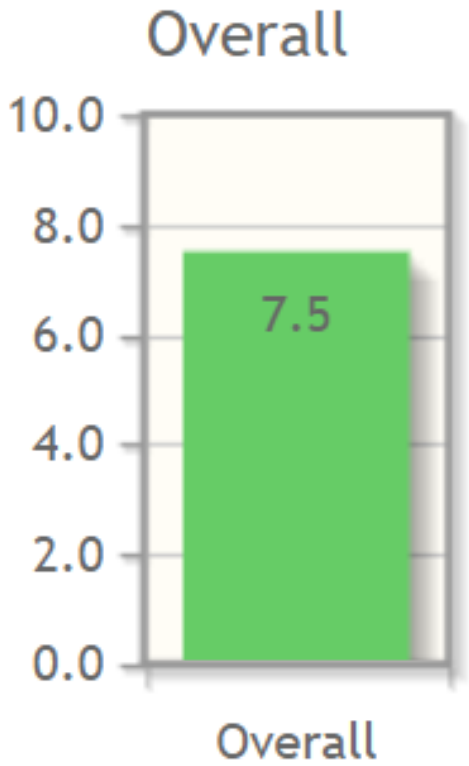
CVSS Example: Heartbleed



CVSS v3.1 Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

CVSS Example: Heartbleed





CVSS v3.1 Vector: [AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N]

Vim Use After Free (CVE-2021-3796)

Base Score Metrics

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 v3.1 Vector: AV:L/AC:H/PR:H/UI:R/S:U/C:H/I:H/A:H/E:P/RL:O/RC:C

Vim Use After Free (CVE-2021-3796)

Temporal Score Metrics

Exploit Code Maturity (E)

Not Defined (E:X)	Unproven that exploit exists (E:U)	**Proof of concept code (E:P)**
Functional exploit exists (E:F)	High (E:H)	

Remediation Level (RL)

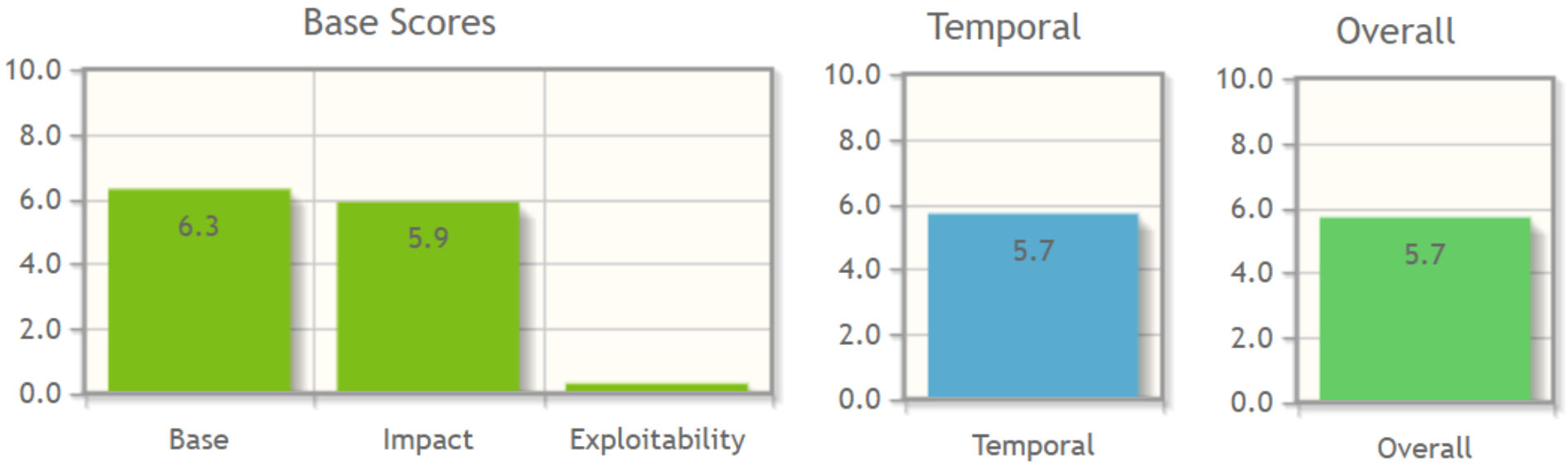
Not Defined (RL:X)	**Official fix (RL:O)**	Temporary fix (RL:T)	Workaround (RL:W)
Unavailable (RL:U)			

Report Confidence (RC)

Not Defined (RC:X)	Unknown (RC:U)	Reasonable (RC:R)	**Confirmed (RC:C)**

CVSS v3.1 Vector: AV:L/AC:H/PR:H/UI:R/S:U/C:H/I:H/A:H/E:P/RL:O/RC:C

Vim Use After Free (CVE-2021-3796)



CVSS v3.1 Vector: AV:L/AC:H/PR:H/UI:R/S:U/C:H/I:H/A:H/E:P/RL:O/RC:C

CVSS score: formula (details not important)

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

If (Impact sub score <= 0) *0 else,*
Scope Unchanged[4] $Roundup(Minimum[(Impact + Exploitability), 10])$
Scope Changed $Roundup(Minimum[1.08 \times (Impact + Exploitability), 10])$

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

Scope Unchanged $6.42 \times ISC_{\text{Base}}$
Scope Changed $7.52 \times [ISC_{Base} - 0.029] - 3.25 \times [ISC_{Base} - 0.02]^{15}$

Where,

$$ISC_{Base} = 1 - [(1 - Impact_{Conf}) \times (1 - Impact_{Integ}) \times (1 - Impact_{Avail})]$$

And the Exploitability sub score is,

$$8.22 \times AttackVector \times AttackComplexity \times PrivilegeRequired \times UserInteraction$$

CVSS score: base values

Metric	Metric Value	Numerical Value
Attack Vector / Modified Attack Vector	Network	0.85
	Adjacent Network	0.62
	Local	0.55
	Physical	0.2
Attack Complexity / Modified Attack Complexity	Low	0.77
	High	0.44
Privilege Required / Modified Privilege Required	None	0.85
	Low	0.62 (0.68 if Scope / Modified Scope is Changed)

...

Source: https://www.first.org/cvss/specification-document section “7.4. Metric Values”.

CVSS score: formula

Weights were determined as follows:

1. Security experts got together (mostly industry)
2. Analyzed a bunch of vulnerabilities in their products
3. Agreed on all the labels for each vulnerability
4. Agreed on an overall ranking of many previous vulnerabilities
5. Adjusted the weights to match from there

CWSS: Common Weakness Scoring System

- Similar to CVSS, but considers **classes of weaknesses**
- Allows rating **vulnerabilities that are not yet known**
- CWSS can score a weakness before the investigation of the vulnerability concludes
- Used much less often than CVSS…

Downsides of CVSS

- It's not explained how the formula was created
  - Only the methodology behind the weights is known
- The robustness and empirical relevance of the formula has been questioned by researchers
- Sometimes unclear how to interpret the resulting number
  - E.g., Is a CVSS of 5.9 really more severe than one of 5.7?

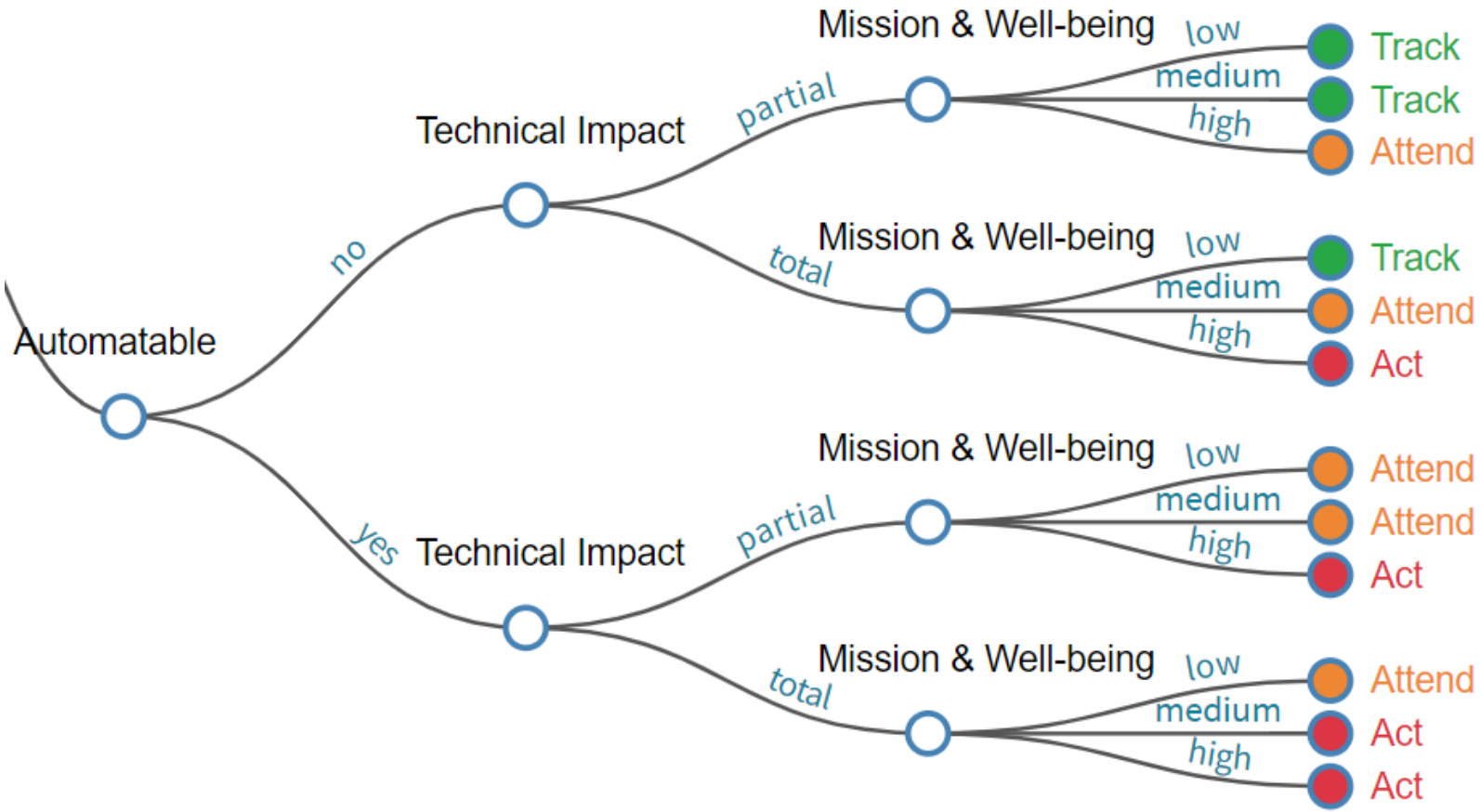
→ Alternative: SSVC system

Source: Jonathan Spring e.a. *Towards improving CVSS*. Software Engineering Institute, CarnegieMellon University, dec 2018.

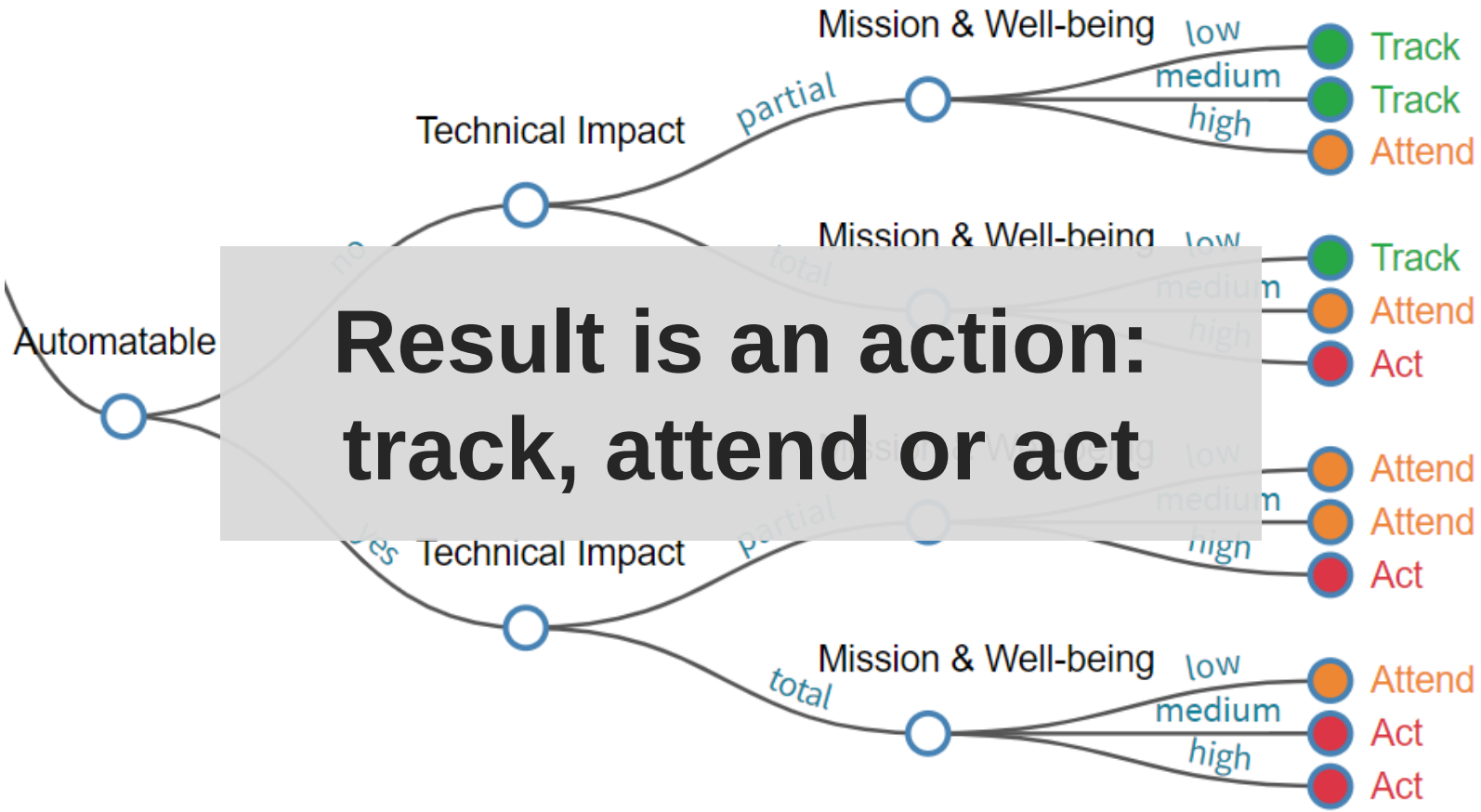
SSVC: Stakeholder-Specific Vulnerability Categorization

- Created in 2019 to address shortcomings in CCVS
- Uses decision trees instead of formulas
- Online decision tree explorer: https://www.cisa.gov/ssvc-calculator

Example: part of SSVC decision tree



Example: part of SSVC decision tree



Vulnerability scoring decisions

Track

The vulnerability does not require attention outside of Vulnerability Management (VM) at this time. Continue to track the situation and reassess the severity of vulnerability if necessary.

Track *

Track these closely, especially if mitigation is unavailable or difficult. Recommended that analyst discuss with other ana-lysts and get a second opinion.

Attend

The vulnerability requires to be attended to by stakeholders outside VM. The action is a request to others for assistance / information / details, as well as a potential publication about the issue.

Act

The vulnerability requires immediate action by the relevant leadership. The action is a high-priority meeting among the relevant supervisors to decide how to respond.

Penetration Testing

How to find vulnerabilities in the first place?

Fuzzing! But it has limitations:

- Hard to explore complex code / detect logical vulnerabilities
- Harder when not having access to the source code (may use 3rd party components without having source code access)

Alternative: manual penetration test by (external) experts

- Usually done once the product is finished
- Mimics how an actual adversary might attack the system

Types of pentest

- Blackbox
  - You have no internal knowledge of the target
  - If there’s no public registration, you might be given a user account
- Whitebox
  - You have full knowledge of the target
  - Source code, multiple accounts with different rights, design docs, etc.
- Greybox
  - In-between blackbox and whitebox.
  - E.g.: access to accounts and some documentation but no source code

Typical stages of a pentest

1. Contract
   - Scoping: what is allowed to be tested (e.g., which servers)
   - Non-disclosure agreement (NDA) and Permission to attack
2. Discovery
   - Enumeration of life hosts and services
   - Exploration of functionality in the program
3. Attack: actual testing of the system
4. Reporting: “the boring but important part”

1. The contract

- Specify what you will be doing
  - E.g., check OWASP top 10, infrastructure test using automated tools,…
- Also specify what you won't be doing
  - E.g., won't check webapps during infrastructure test
- Specify the exact version of the target
  - Prevent that the organization will do updates during the test
- Warn client that a pentest won't discover all vulnerabilities
  - Due to time constraints, black box nature of the test, etc.

1. The contract: scope screep

- Don’t test anything outside the original scope!
- Even when the client asks “can you also do a quick test of …”
- Pentest may get delayed. May lose focus and miss things.
- You may also get in legal trouble!
  - Is this new activity covered by the contract?!

→ Create a new (second) contract instead

1. The contract: permission to attack

- Also called a “get out of jail free card”
- Should contain:
  - The parties involved
  - The clearly identified target (i.e., IP range, domain, web app, …)
  - Your source IP for internet-based tests
  - The duration of the test (extend in case you need more time)
- Due do diligence
  - Does the target belong to the other party (e.g., WHOIS lookup for IP)?
  - Check that the person who signed the contract has the right authority

1. The contract: non-disclosure agreement

- Usually asked by clients
- Basically, keep any discovered company secrets private

2. Discovery: where to start?

- If an IP range was given
  - Check for live hosts. Don’t purely rely on a ping sweep!
  - Do a full range TCP port scan on life hosts
- For a single web application
  - Start your attack proxy (e.g., Burp Suite)
  - Surf on the web application & use it like a normal user
  - Watch for hints of technologies being used

2. Discovery: port scanning

- Most common tool: nmap or zenmap
- Advice: turn on service discovery and version detection during full range port scan
- Gives you a list of potential targets per host
- Scan the low ports first (below 10 000)
  - Usually contains most standard software/services

2. Discovery: example nmap command

```
nmap –p 1-65535 –sV –sS –T4 -O –oA testbed –v 10.0.10.1-255
```

- -p ports to be scanned
- -sV Service & version discovery
- -sS SYN (Stealth) Scan (never completes TCP handshake)
- -T4 “Scanning speed”, 4 is relatively aggressive
- -O OS detection
- -oA Do output of all available formats
- -v Verbose output to terminal, even though -oA is used

2. Discovery: example nmap output

```
manav@ubuntulinux:~$ nmap 103.76.228.244 157.240.198.35 172.217.27.174
Starting Nmap 7.80 ( https://nmap.org ) at 2020-05-19 16:57 UTC
Nmap scan report for bridgei2p.com (103.76.228.244)
Host is up (0.062s latency).
Not shown: 991 filtered ports
PORT    STATE SERVICE
22/tcp  open  ssh
25/tcp  open  smtp
80/tcp  open  http
110/tcp open  pop3
443/tcp open  https
465/tcp open  smtps
587/tcp open  submission
993/tcp open  imaps
995/tcp open  pop3s
```

2. Discovery: website discovery

- Always have an attack proxy running
  - Burp Suite, OWASP ZAP, …
  - Keeps a log of request/responses for later review
- Some commercial ones point out flaws by analyzing traffic
- Manually analyzing requests & responses may give hints
  - Misconfigurations like missing cookie flags
  - Technologies (i.e. ASP.NET)

2. Discovery: BRUP Suite example

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Exten
7	https://update.googleapis.com	POST	/service/update2/json?cup2key=10:1...	✓		200	14648	JSON	
8	http://redirector.gvt1.com	GET	/edgedl/release2/chrome_component/...			302	1053	HTML	
12	https://portswigger-labs.net	GET	/index_files/jquery-2.js			200	85908	script	js
14	https://portswigger-labs.net	GET	/index_files/portswigger-logo.svg			200	8309	XML	svg
15	https://portswigger-labs.net	GET	/index_files/ps-mobile-logo.svg			200	963	XML	svg
17	https://portswigger-labs.net	GET	/Content/Fonts/DroidSans/s-BiyweUP...			200	21722		woff2
18	https://update.googleapis.com	POST	/service/update2/json	✓		200	1026	JSON	
20	http://redirector.gvt1.com	GET	/edgedl/release2/chrome_component/...			302	1023	HTML	
22	https://update.googleapis.com	POST	/service/update2/json	✓		200	1026	JSON	
23	http://redirector.gvt1.com	GET	/edgedl/release2/chrome_component/...			302	1067	HTML	
25	https://update.googleapis.com	POST	/service/update2/json	✓		200	1026	JSON	
26	http://redirector.gvt1.com	GET	/edgedl/release2/chrome_component/...			302	1027	HTML	
28	https://update.googleapis.com	POST	/service/update2/json	✓		200	1026	JSON	

Request

Pretty | Raw | \n | Actions

```
1 GET /index_files/ps-mobile-logo.svg HTTP/1.1
2 Host: portswigger-labs.net
3 Connection: close
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/87.0.4280.88 Safari/537.36
```

Response

Pretty | Raw | Render | \n | Actions

```
1 HTTP/1.1 200 OK
2 Date: Wed, 03 Feb 2021 09:55:06 GMT
3 Server: Apache/2.4.41 (Ubuntu)
4 Upgrade: h2
5 Connection: Upgrade, close
6 Last-Modified: Fri, 29 May 2020 10:53:20 GMT
```

INSPECTOR

3. Attack: automated scanners

- Good for getting a big picture of a large IP range
- Most well known: Nessus and OpenVAS
- Regularly updated with most recent exploits
- Can be run on a schedule to monitor development
- Not suitable for web apps

3. Attack: OWASP

- Open Web Application Security Project
- Provides Top 10 lists of most common vulnerabilities
  - Web Apps
  - IoT
- And provides helpful cheatsheets/guides
  - https://cheatsheetseries.owasp.org/
- Creators of ZAP attack proxy

3. Attack: OWASP Web Top 10

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using components with known vulnerabilities
10. Insufficient logging and monitoring

4. Report must contain

- Exact version/configuration that was tested
- Detailed explanation of the tests performed
- Clearly states the results. They should be repeatable.
- Identify any limitations of the teste
  - “Absence of proof isn’t proof of absence”
  - You can be held liable for what is written in the report!
- Optionally rate findings & give advice on fixes

Redteaming

- The “new” way to do pentests
- Done over an extended time (months). Usually black-box.
- Goal: provide more realistic tests by a combination of
  - Infrastructure test
  - Webapp test
  - Phishing
  - Social engineering (also on-site)
- Gets a realistic view of a company’s security

Get out of jail free card

- Very important for engagements *on-site*
- A piece of paper, signed (ideally) by the highest person in the company
- States that you were hired
- *(Show a fake one first as an additional test if caught)*