Application Security

Objectives

After completing this unit, you should be able to:

- What is Application Security?
- Approach towards Web Application Security
- OWASP — Top 10 Web Application Vulnerabilities
- SSDLC (Secure Software Development Life Cycle)

Application Security Introduction

FTVETI

ICT @ FTVETI

What Do You Mean By Application Security

- Application security comprises of measures taken to improve the security of an application often by **finding**, **fixing** and **preventing security vulnerabilities**
- Different techniques are used to surface such **security vulnerabilities** at different stages of an **applications lifecycle** such as:-
  - Design
  - Development
  - Deployment
  - Upgrade
  - Maintenance

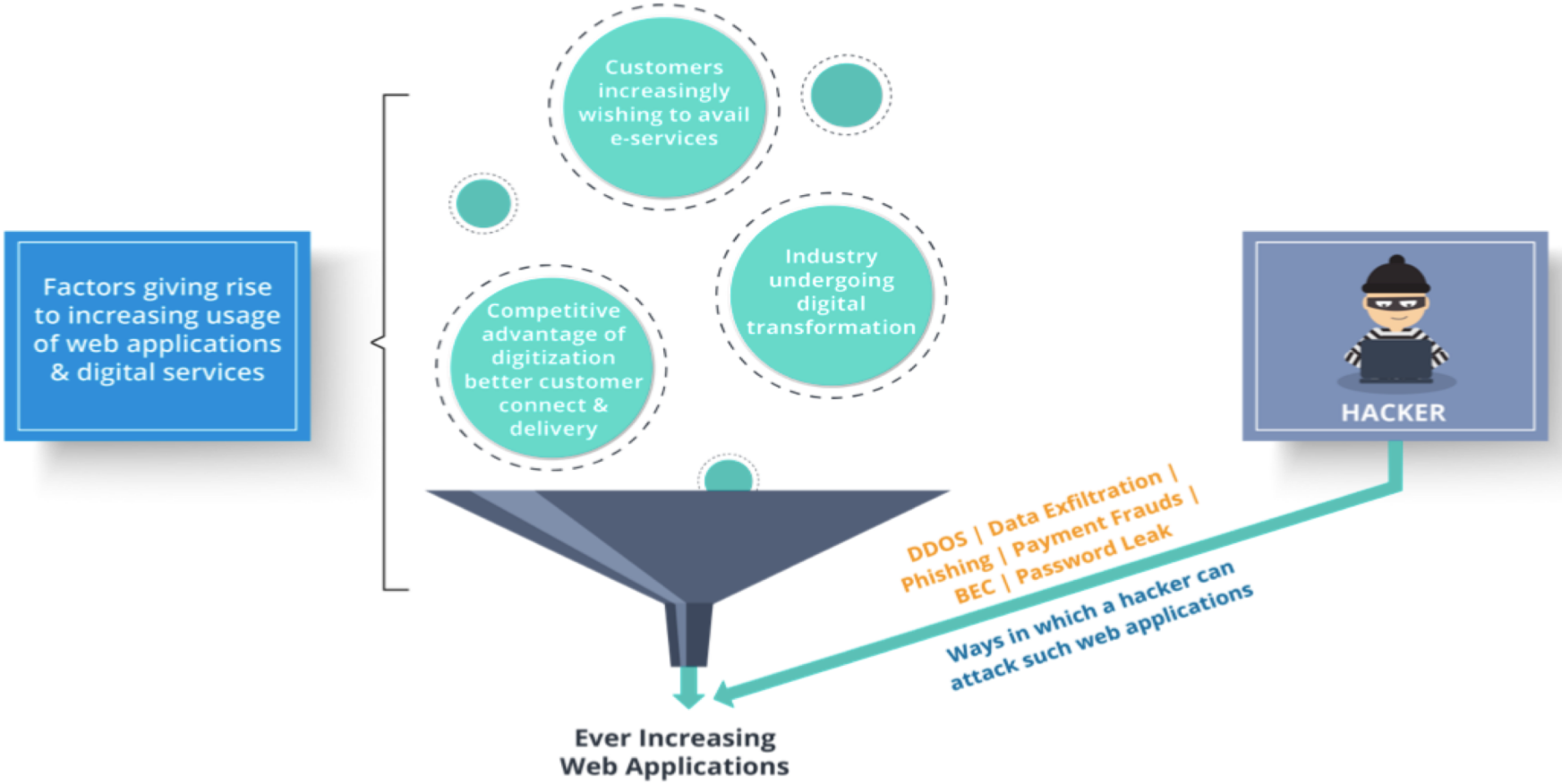
We have seen the importance of application security. Let's move on to its perspective and the approach towards web application Security

Perspective

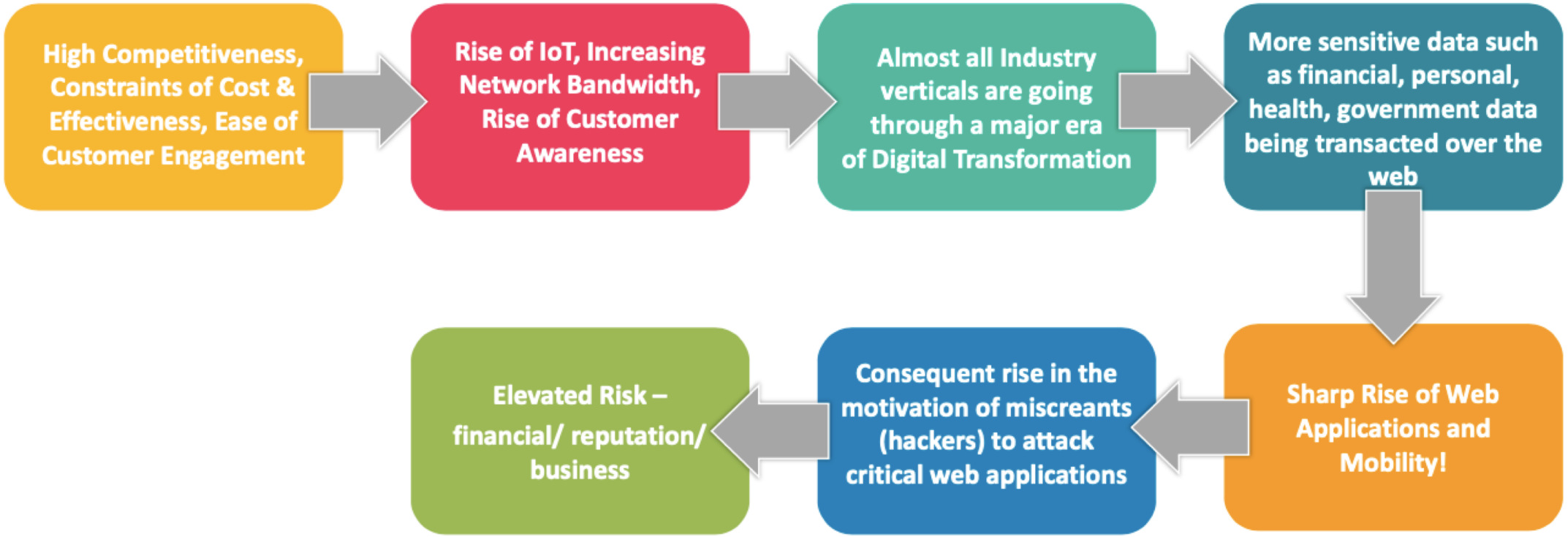
Age Of Digitally Interconnected Services

The image talks about today's era, of digital transformation and increasing use of web applications worldwide

Digital Era


Factors giving rise to increasing usage of web applications & digital services
Customers increasingly wishing to avail e-services
Industry undergoing digital transformation
Competitive advantage of digitization better customer connect & delivery
HACKER
DDOS | Data Exfiltration | Phishing | Payment Frauds | BEC | Password Leak
Ways in which a hacker can attack such web applications
Ever Increasing Web Applications

Key Take – Away

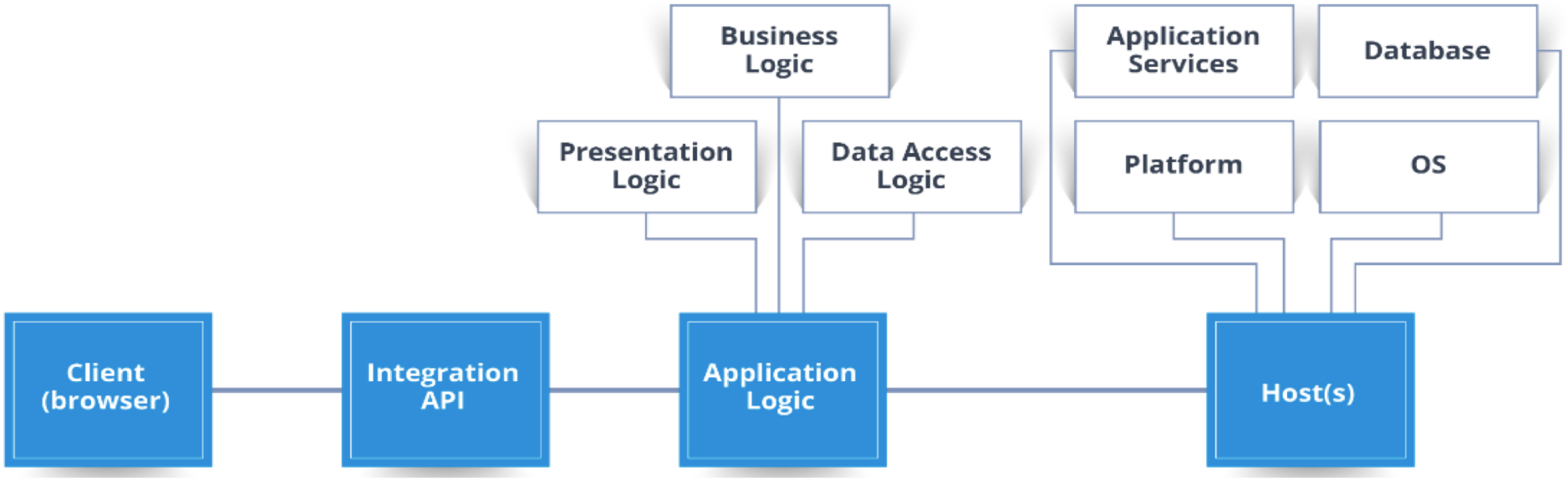


Approach Towards Web Application Security

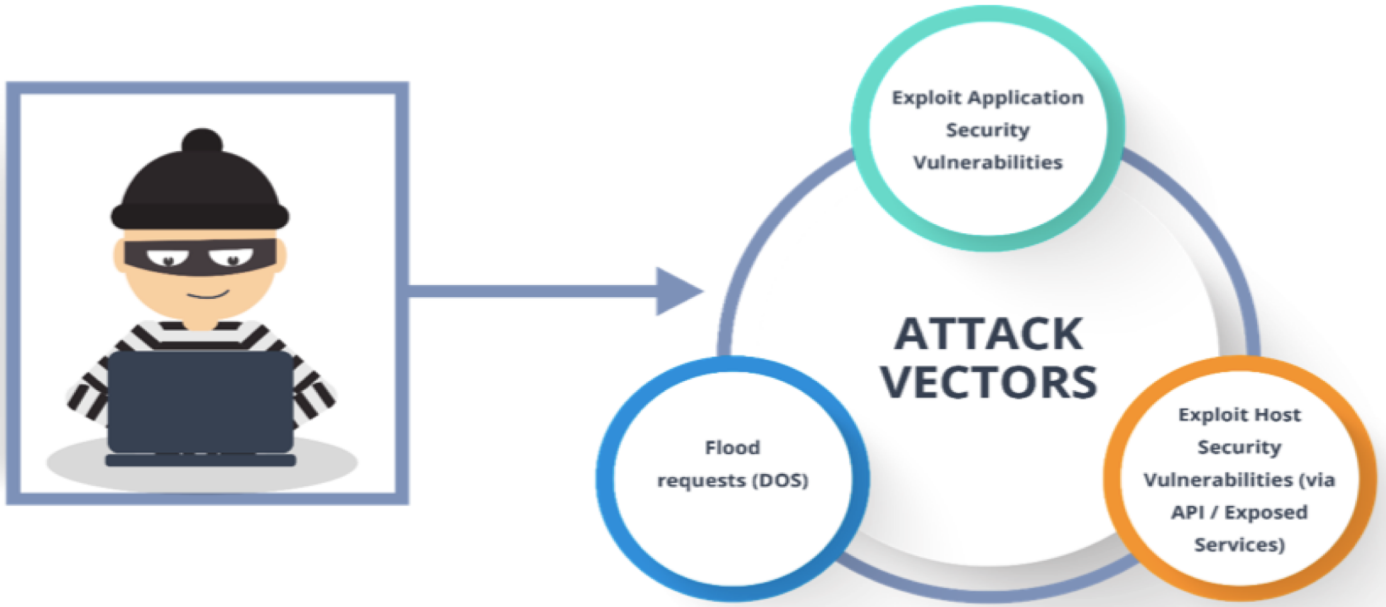
ICT @ FTVETI

Web Application – Components

A web application is not just one application but a collection of several units as shown:



Attackers Viewpoint

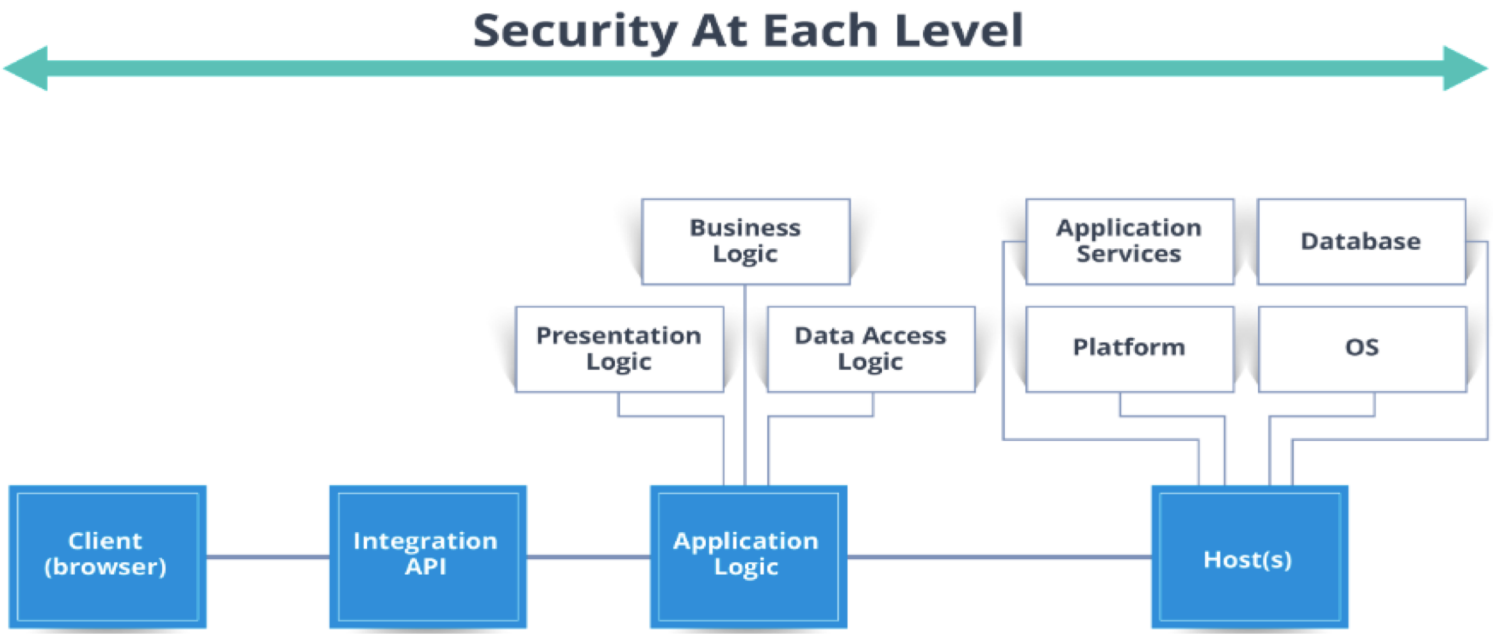


Typical ways employed by hackers to attack web applications:

- **Exploit Application Security Vulnerabilities:** Exploitation of known weaknesses in the web applications
- **Exploit Host Security Vulnerabilities (via API & Exposed Services):** Exploitation of weaknesses in the hosting environment and communication channels of the application
- **Flood Requests (DOS):** Unmanaged weakness of an application such that the application gives up if it is used beyond a limit

Web Application Security – High Level Approach

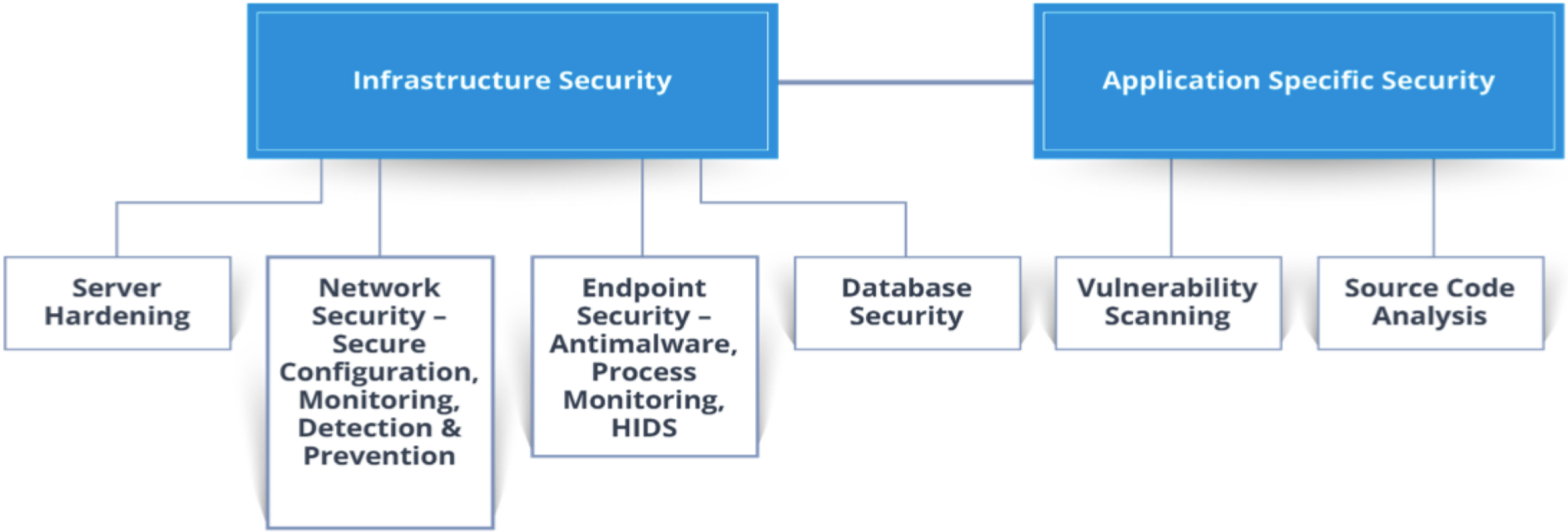
As various parts associated with a web application may be vulnerable and can contribute towards failure of an application, security should span across various layers and parts to ensure holistic security



Web Application Security

Two major pillars of Web Application Security :

- **Infrastructure Security:** Involves Security of various layers within infrastructure part
- **Application Security:** Refers to areas like continual vulnerability scanning (while creation & after deployment) of applications and source code analysis




In order to enable organizations to conceive, develop, acquire, operate, and maintain applications that can be trusted, we need to have the understanding of OWASP

OWASP – Introduction

FTVETI

ICT @ FTVETI

OWASP – Overview

01 OWASP is an open community dedicated to enabling organizations to conceive, develop, acquire, operate, and maintain applications that can be trusted

02 All of the OWASP provided resources are declared free to the community

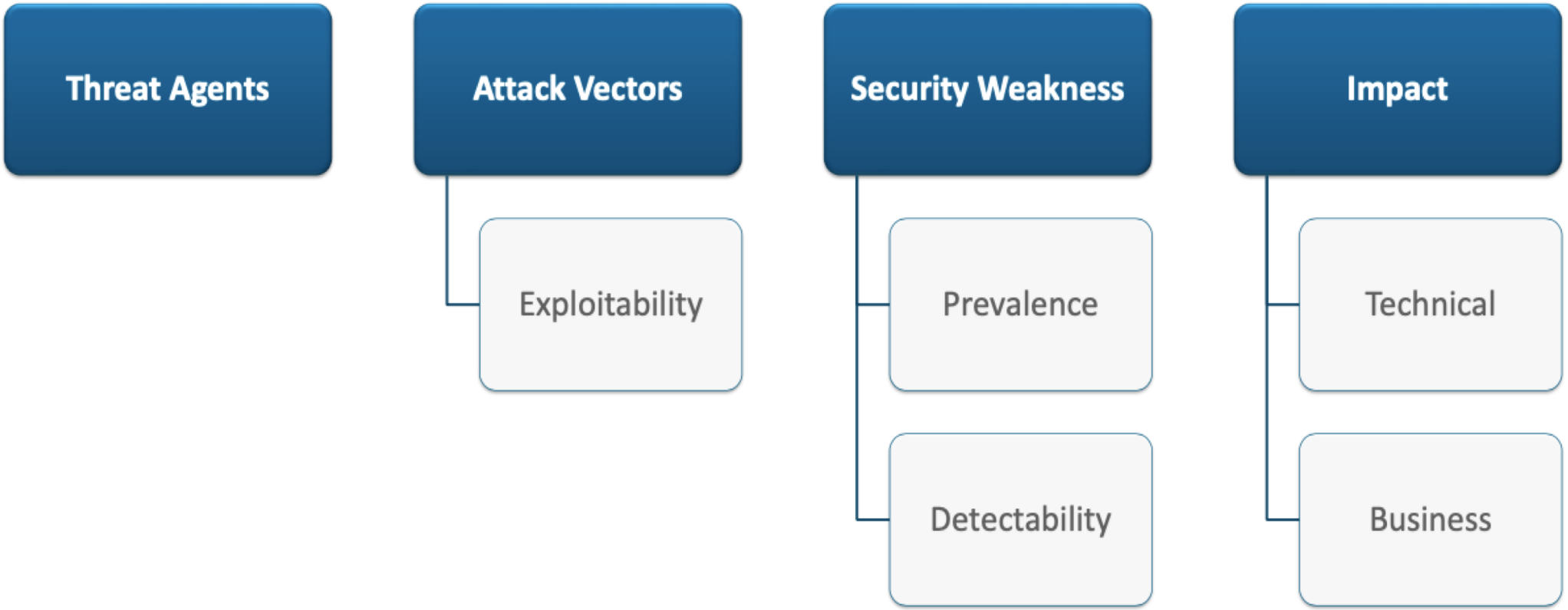
03 OWASP provides a direction to help understanding of security issues. More importantly it lends credibility to penetration testing reports allowing security practitioners to promote security best practices

04 OWASP Top 10 focuses on identifying the most serious web application security risks for a broad array of organizations

05 OWASP Software Assurance Maturity Model (SAMM) is an open framework to help organizations formulate and implement a strategy for software security that is tailored to the specific risks facing the organization

Source : https://owasp.org/

Vulnerability Scoring Methodology



Top Web Application Vulnerabilities

OWASP Top 10 Vulnerabilities – 2017

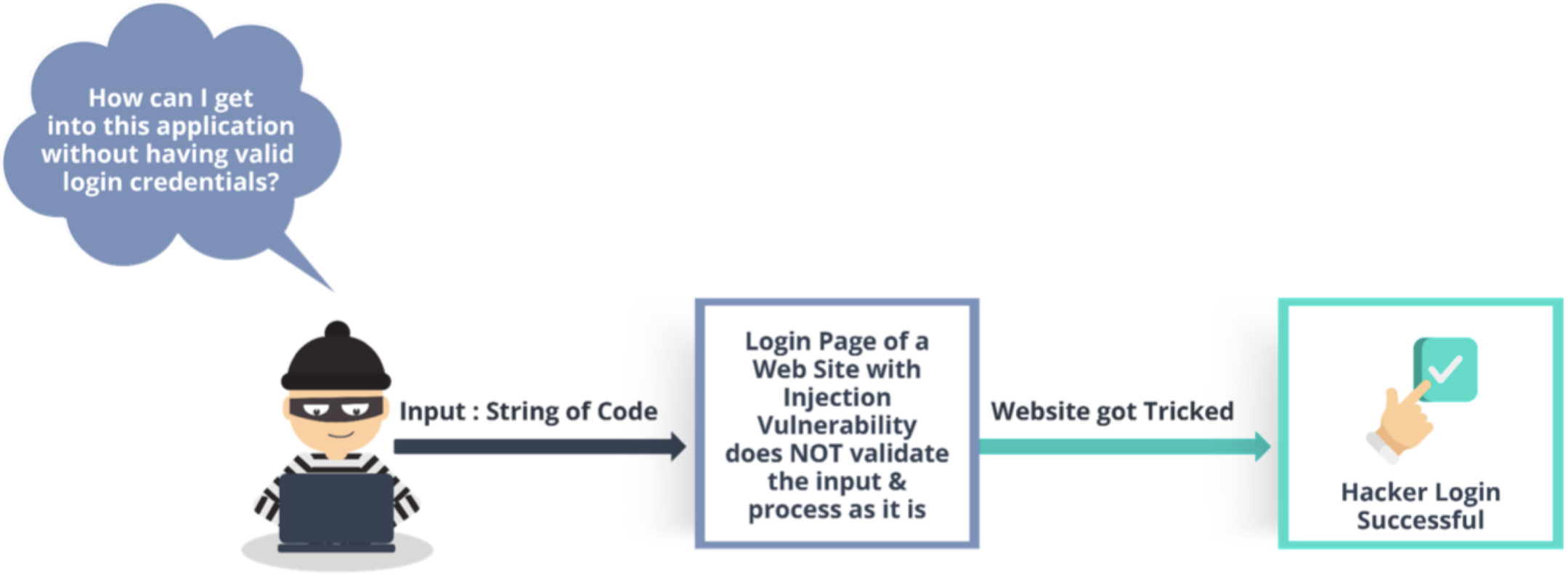
A1:2017 . Injection	• Untrusted data is sent to an interpreter as a part of command or query
A2:2017. Broken Authentication	• Incorrect implementation of application functions related to authentication & session management
A3:2017. Sensitive Data Exposure	• Data at rest or transit not protected enough
A4:2017. XML External Entities (XXE)	• Poorly configured XML processors evaluating external entity references within XML documents
A5:2017. Broken Access Control	• Improperly enforced restrictions on non-authenticated / guest users

OWASP Top 10 Vulnerabilities – 2017

A6:2017. Security Misconfiguration	• Insecure default configurations of various application functions or ad-hoc settings
A7:2017. Cross-Site Scripting (XSS)	• Inclusion of untrusted data in a new web page without validation causing attackers to execute scripts in the victim's browser thus causing hijack user sessions, deface websites, redirection to malicious websites
A8:2017. Insecure De-Serialization	• unmanaged or incorrect deserialization leading to unauthorized RCE (remote code execution)
A9:2017. Using Components with known Vulnerabilities	• Usage of libraries, frameworks and other software modules with exploitable vulnerabilities has a potential of compromise of the application as a whole
A10:2017. Insufficient Logging & Monitoring	• Not logging enough or not monitoring the logs for timely incident response causes a bigger attacks to succeed over time.

Injection

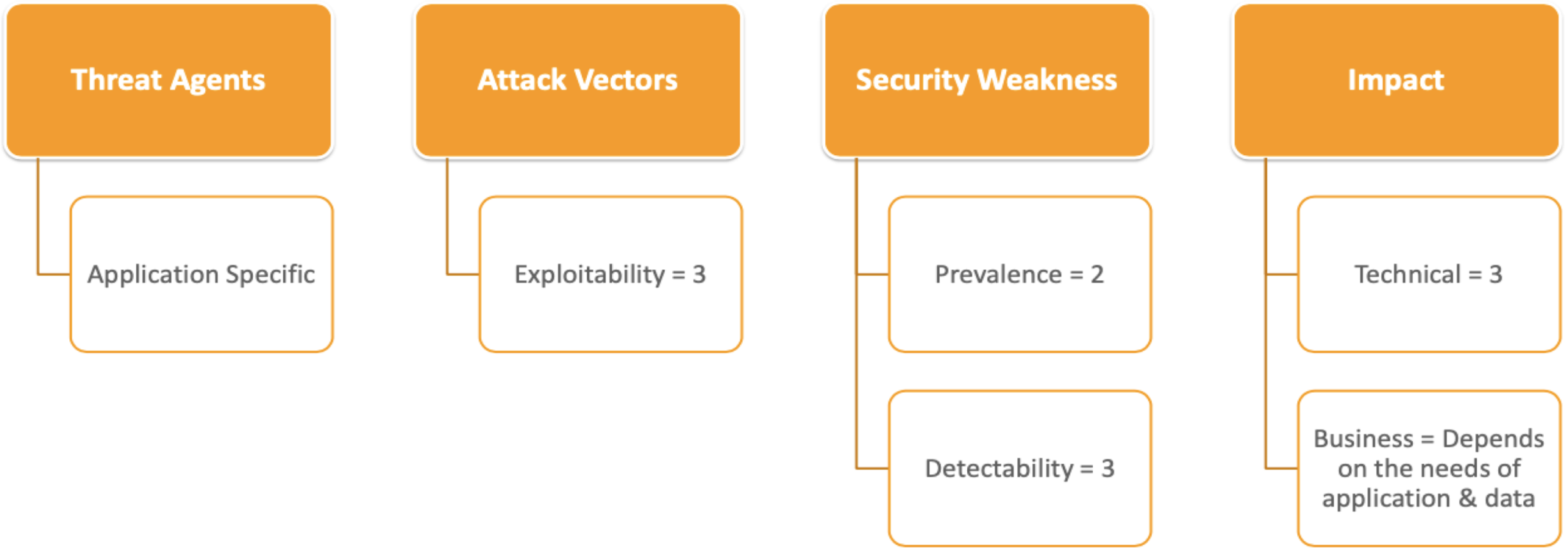
Injection


How can I get into this application without having valid login credentials?
Input : String of Code
Login Page of a Web Site with Injection Vulnerability does NOT validate the input & process as it is
Website got Tricked
Hacker Login Successful

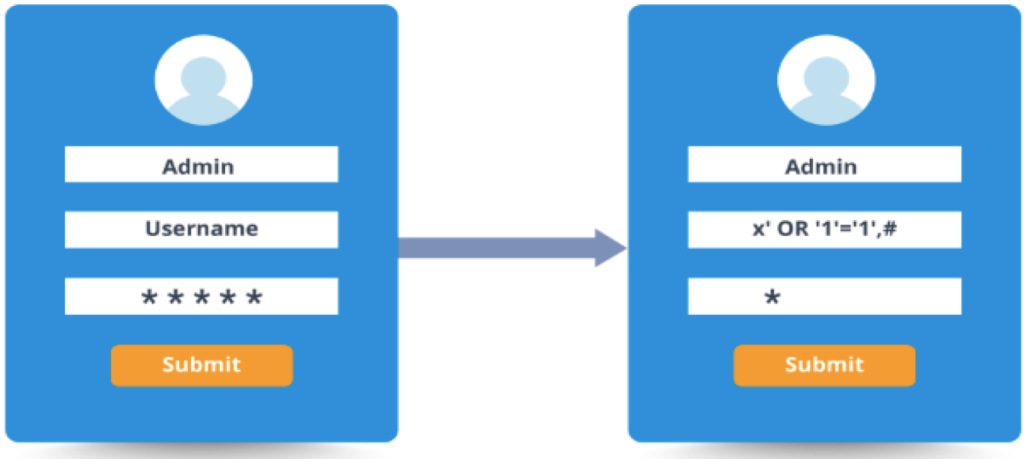
What Is Injection?

- Injection is a bug that occur when an attacker gets an opportunity to send some hostile data to an application interpreter
- Injection flaws are easy to discover when examining Injection flaws are easy to discover when examining code
- Scanners and fuzzers can help attackers find code
- Injection can result in data loss , corruption, or disclosure to unauthorized parties or denial of access or complete host takeover

Injection – Scorecard



Injection – Example (SQLi)



Background SQL Query

Select * from admin where USERNAME = 'x' or '1'='1'; # And PASSWORD = 'z'

Result

Successful Login

Injection – Prevention

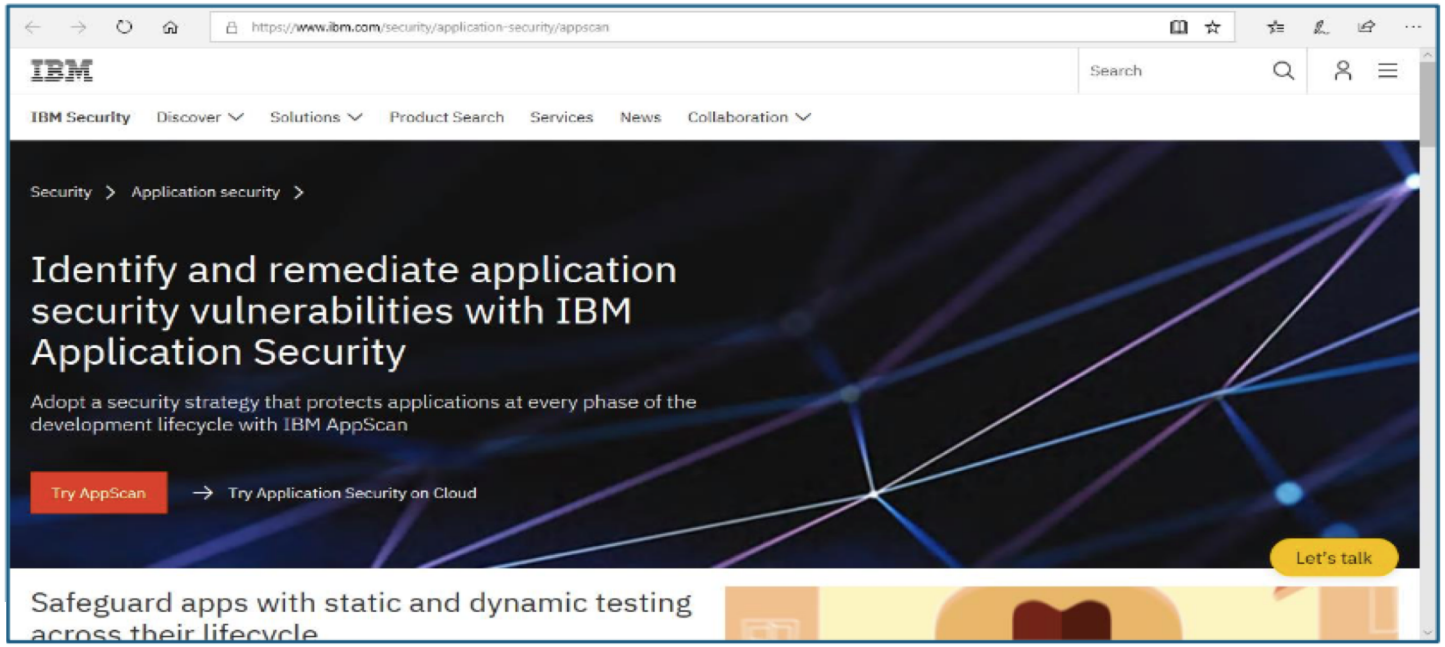
- Keeping data separate from commands & queries
- Usage of Safe API which provides parameterized interface
- Server side validation
- Usage of LIMIT & other SQL controls within Queries to prevent mass disclosure of records in case of a successful SQL injection

Injection – References

SQL Injection	https://www.owasp.org/index.php/SQL_Injection
SQLi	https://www.websec.ca/kb/sql_injection
Parameterized Queries	https://www.websec.ca/kb/sql_injection
OWASP Proactive controls	https://www.owasp.org/index.php/OWASP_Proactive_Controls#2:_Parameterize_Queries
Deliberately insecure web application maintained by OWASP	https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project

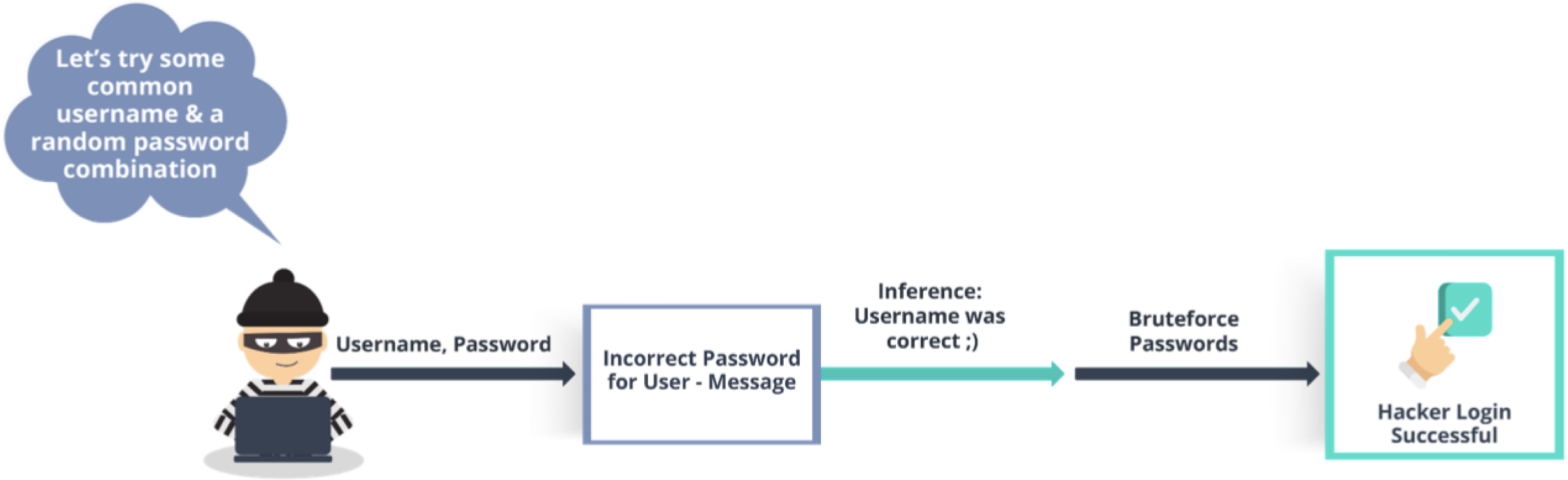
Demo 1: SQL Injection

- Scan a web application for various vulnerabilities and security issues
- Download and install **IBM AppScan** tool from the link: https://www.ibm.com/security/application-security/appscan
- Test SQL injection and generate report

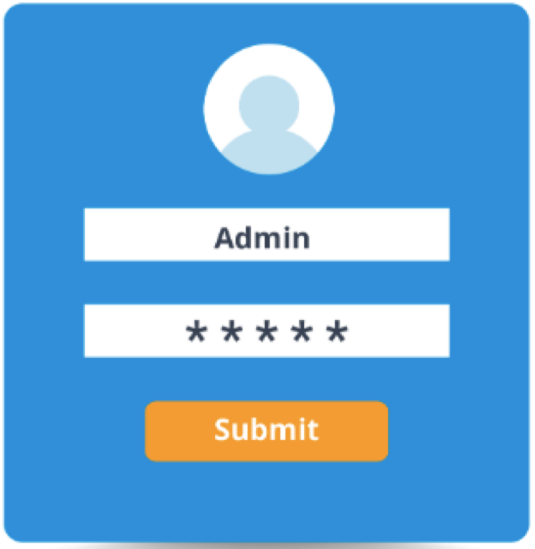


Broken Authentication

Broken Authentication



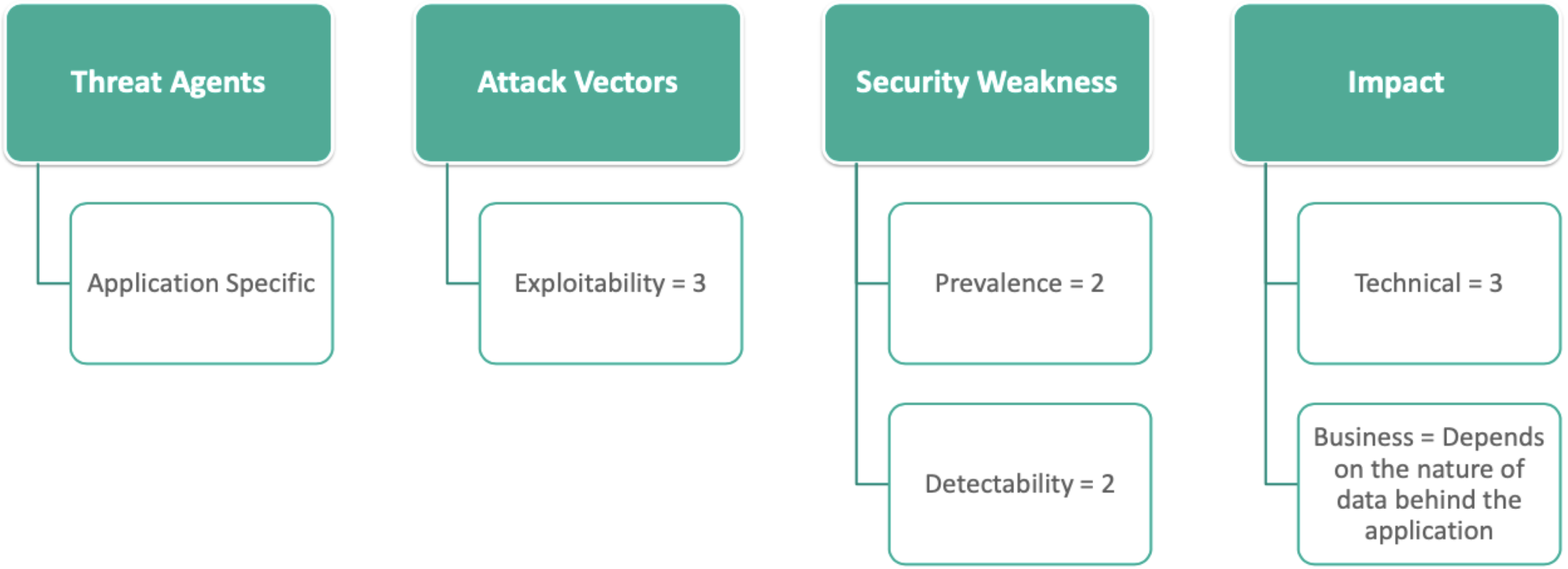
Broken Authentication Characteristics



Application vulnerable to Broken Authentication typically:

- Permits automated attacks such as Credential stuffing (automated injection of breached username/password pairs in order to fraudulently gain access to user accounts) or Brute Forcing
- Permits weak default user/password combinations (admin/admin, admin/password, admin/password123 and so on)
- Uses weakly hashed password storage
- Misses on MFA (multi factor authentication) mechanism
- Exposes Session IDs in the URL
- Improper or No Invalidation of Session IDs of inactive users

Broken Authentication – Scorecard



Broken Authentication – Prevention

- Use MFA
- Do not deploy default credentials, especially for admin users
- Enforce Weak Password Checks and appropriate Password complexity requirements [Ref. NIST SP 800-63 B]
- Log all failed login attempts
- Limit Repeated Logons
- Use a server-side secure built in session manager that generates a new random session ID and invalidates on logout

Broken Authentication – References

Credential Stuffing	https://www.owasp.org/index.php/Credential_Stuffing_Prevention_Cheat_Sheet
Forgot Password Cheat Sheet	https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet
Authentication Cheat Sheet	https://www.owasp.org/index.php/Authentication_Cheat_Sheet
NIST SP 800-63 [5.1.1 Memorized Secrets]	https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret

Sensitive Data Exposure

Sensitive Data Exposure

What Is Sensitive Data Exposure?

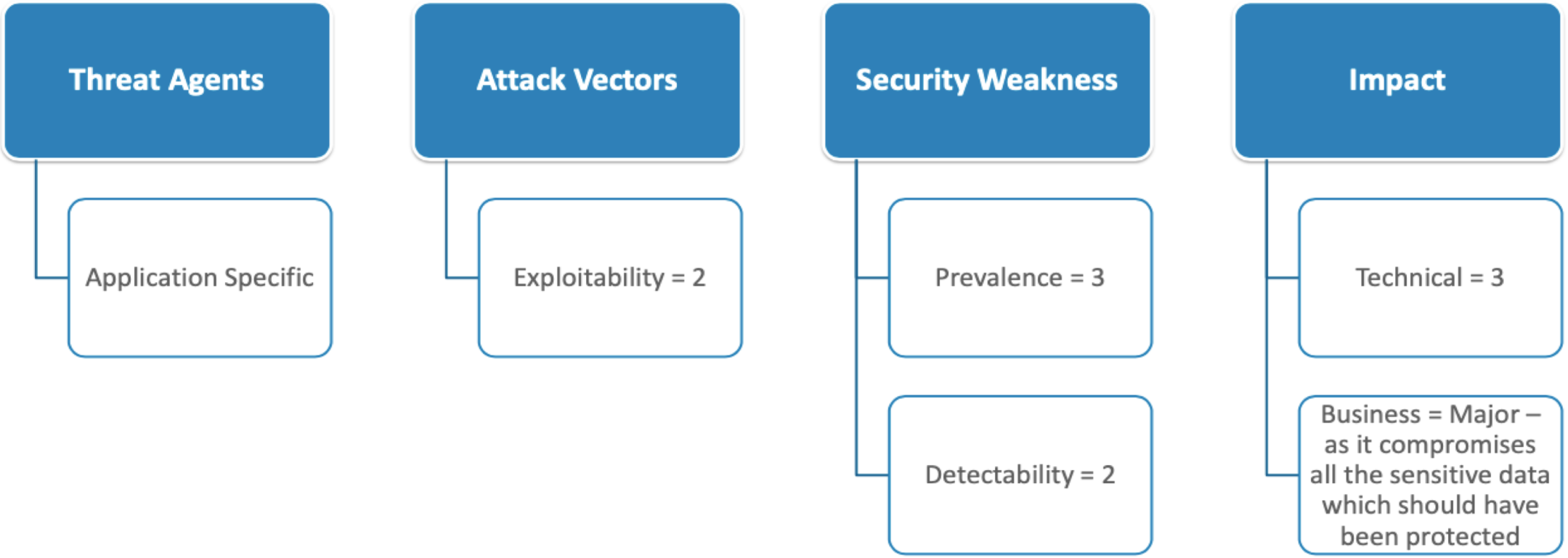
Aim of the Hacker is to obtain sensitive data held / processed by the application

One of the most impactful attacks in terms of legal/compliance and business reputation perspective

Typically the vulnerable application :

- Transmits or backups the data in clear text over protocols such as HTTP, SMTP, FTP and so on
- Uses old cryptographic algorithms
- Uses default crypto keys
- Does not enforce encryption (missing browser security directives)
- User agent (browser) does not verify if the received server certificate is valid

Sensitive Data Exposure – Scorecard

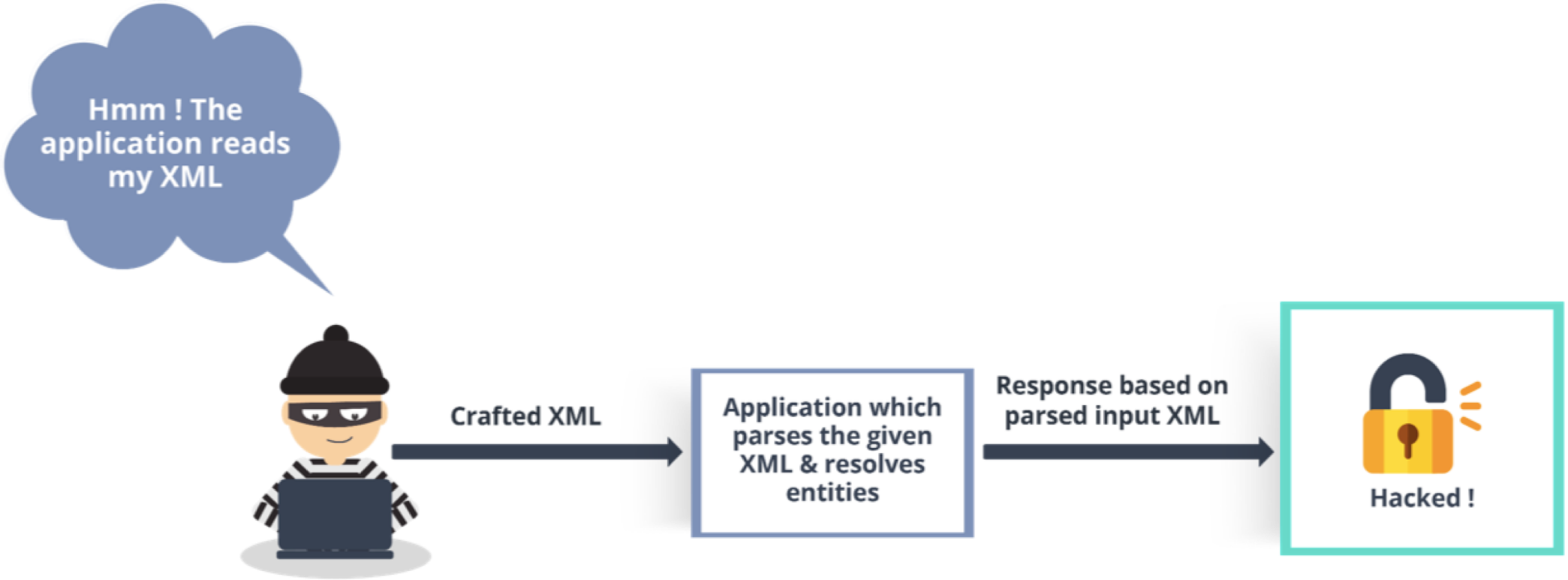


Sensitive Data Exposure – References

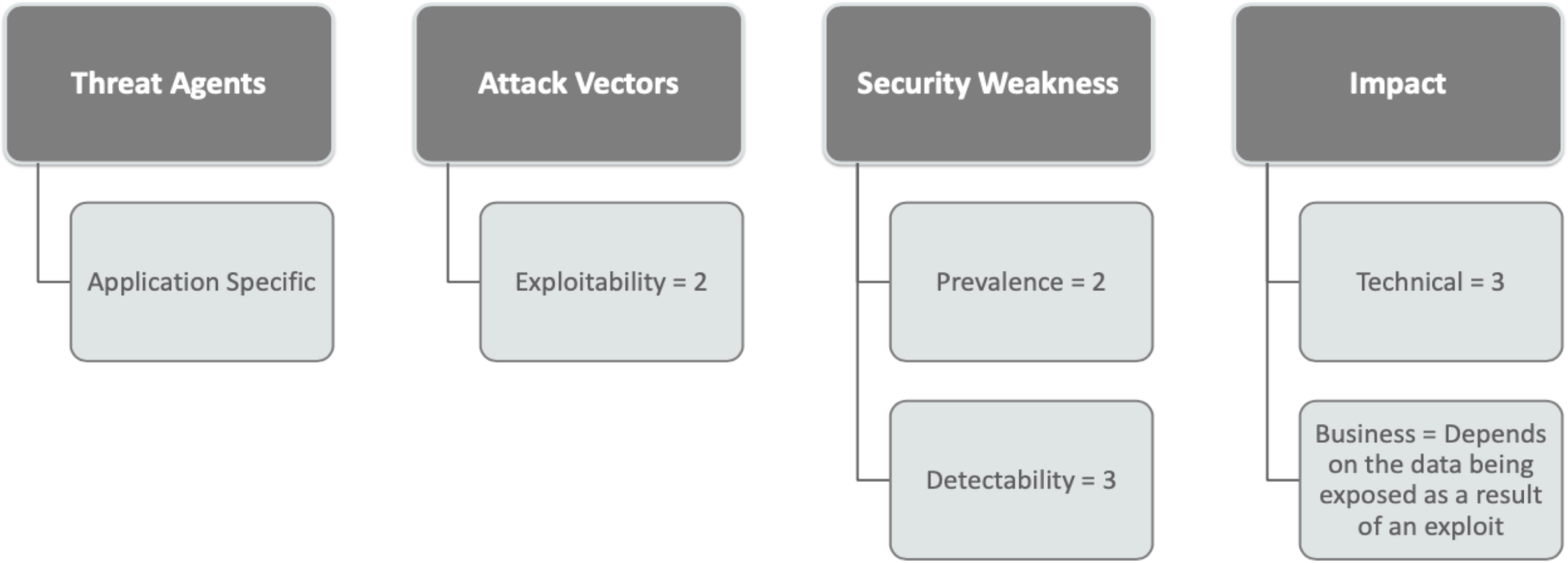
OWASP Proactive Controls	https://www.owasp.org/index.php/OWASP_Proactive_Controls#7:_Protect_Data
Transport Layer Protection Cheat Sheet	https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet
Testing for weak Cryptography	https://www.owasp.org/index.php/Testing_for_weak_Cryptography
Password Storage Cheat Sheet	https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet
OWASP Secure Headers Project	https://www.owasp.org/index.php/OWASP_Secure_Headers_Project

XML External Entities (XXE)

XML External Entities (XXE)



XXE – Scorecard

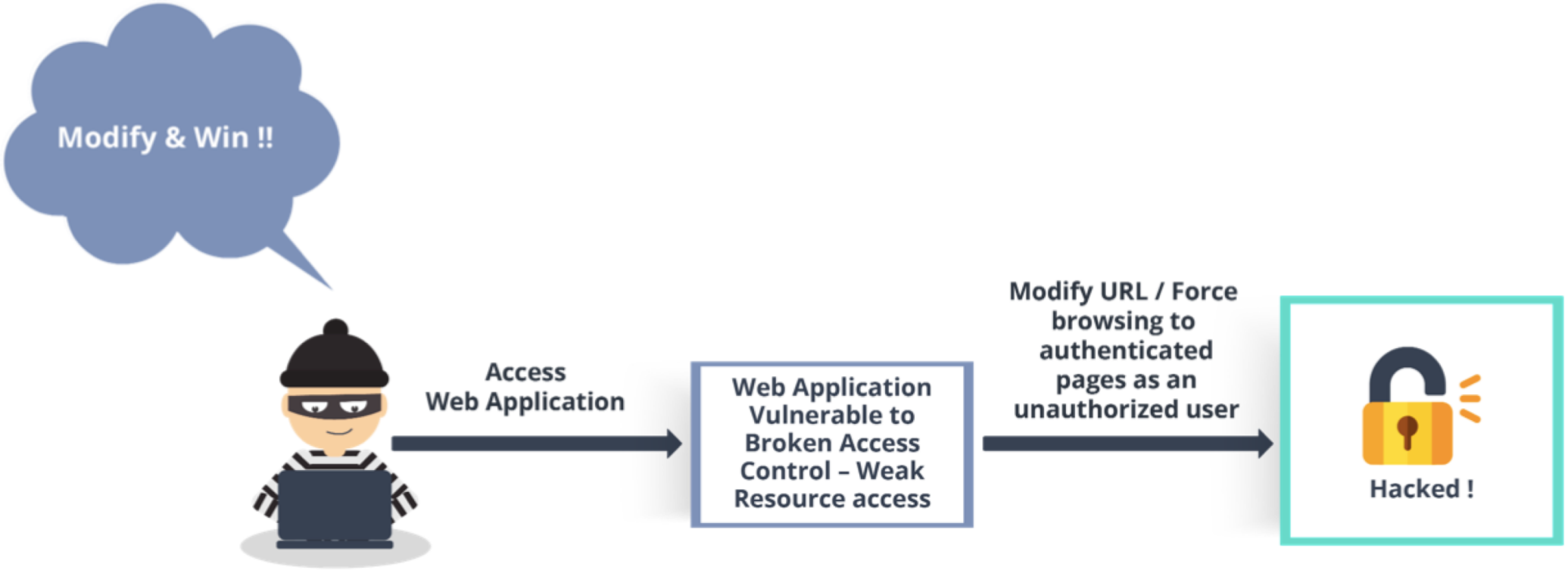


XXE – References

Testing for XML Injection	https://www.owasp.org/index.php/Testing_for_XML_Injection_(OTG-INPVAL-008)
XML Security Cheat Sheet	https://www.owasp.org/index.php/XML_Security_Cheat_Sheet
XXE Prevention Cheat Sheet	https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet
Detecting and exploiting XXE in SAML Interfaces	https://web-in-security.blogspot.tw/2014/11/detecting-and-exploiting-xxe-in-saml.html

Broken Access Control

Broken Access Control



Broken Access Control – Characteristics

Access control refers to enforcement of a boundary of actions for a user. Users typically should not be able to act outside of the granted permissions

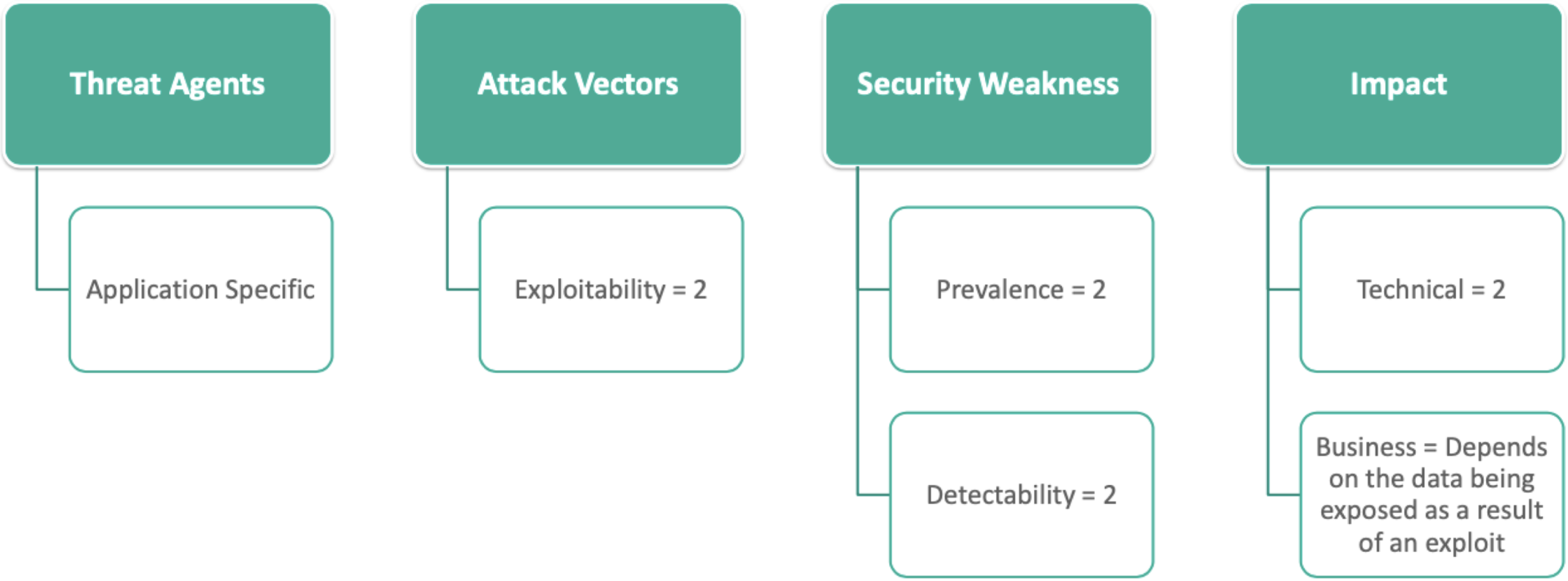
Failures typically lead to unauthorized information disclosure, modification or destruction of all data, or the hacker performing a business function outside the limits of the user level he otherwise has.

This involves Bypassing access control checks by modification of the URL, internal application state and so on

EOP - Elevation of Privilege

Forceful browsing to authenticated pages as an unauthenticated user

Broken Access Control – Scorecard

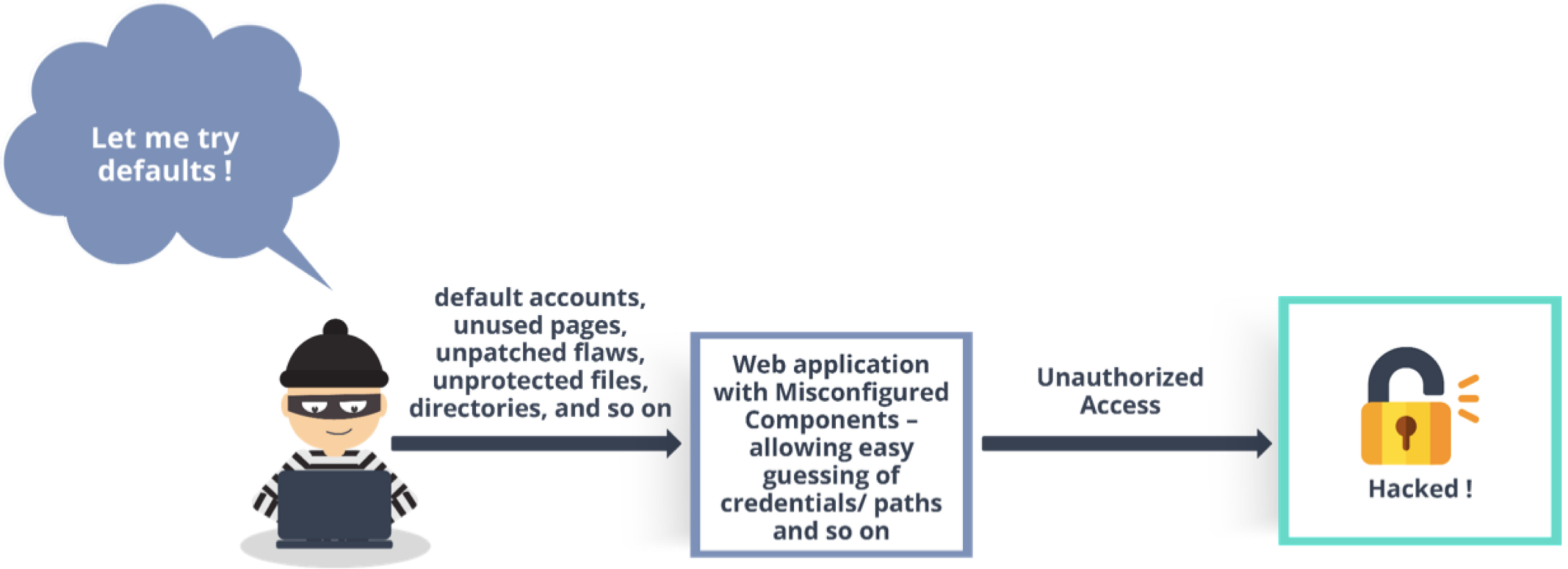


Broken Access Control – References

OWASP ASVS	https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home
Testing for Authorization	https://www.owasp.org/index.php/Testing_for_Authorization
Access Control Cheat Sheet	https://www.owasp.org/index.php/Access_Control_Cheat_Sheet

Security Misconfiguration

Security Misconfiguration



Security Misconfiguration – Characteristics

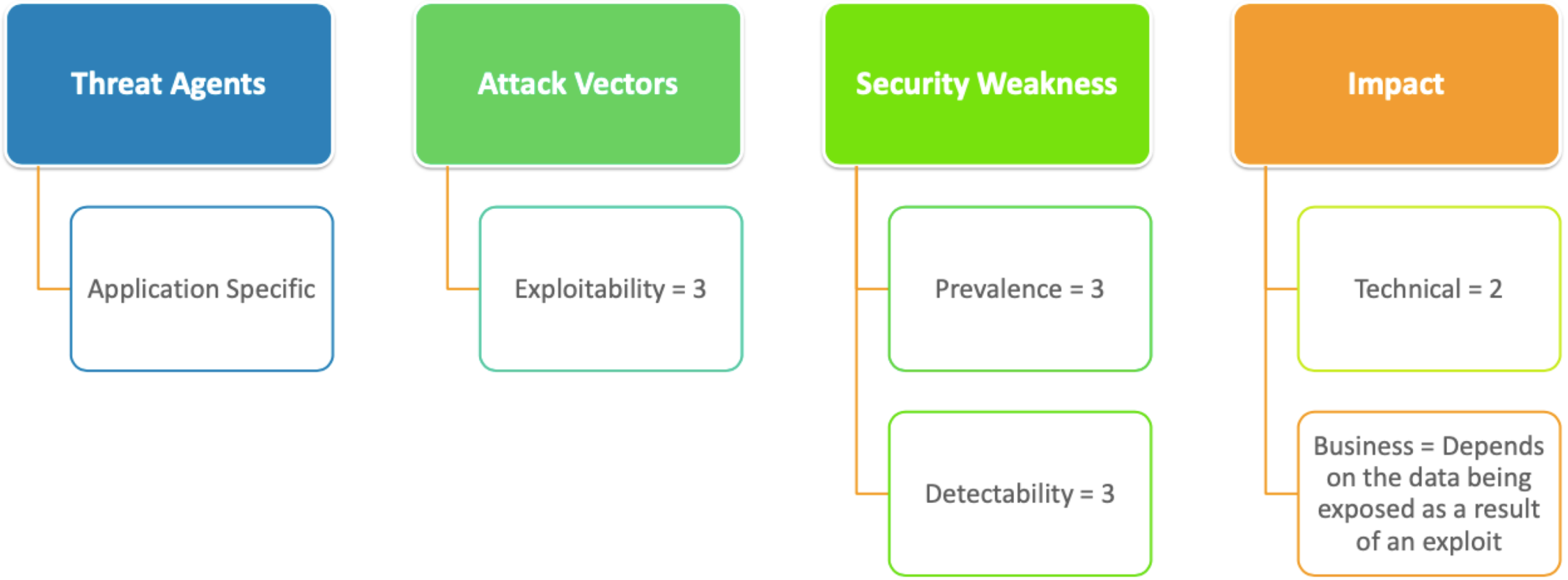
01 Missing appropriate security hardening across any part of the application stack, or even misconfigured permissions on cloud services

02 Unnecessary features enabled on installed on servers on application stack

03 Default Accounts & passwords still enabled on unchanged

04 Poor & revealing error handling

Security Misconfiguration – Scorecard



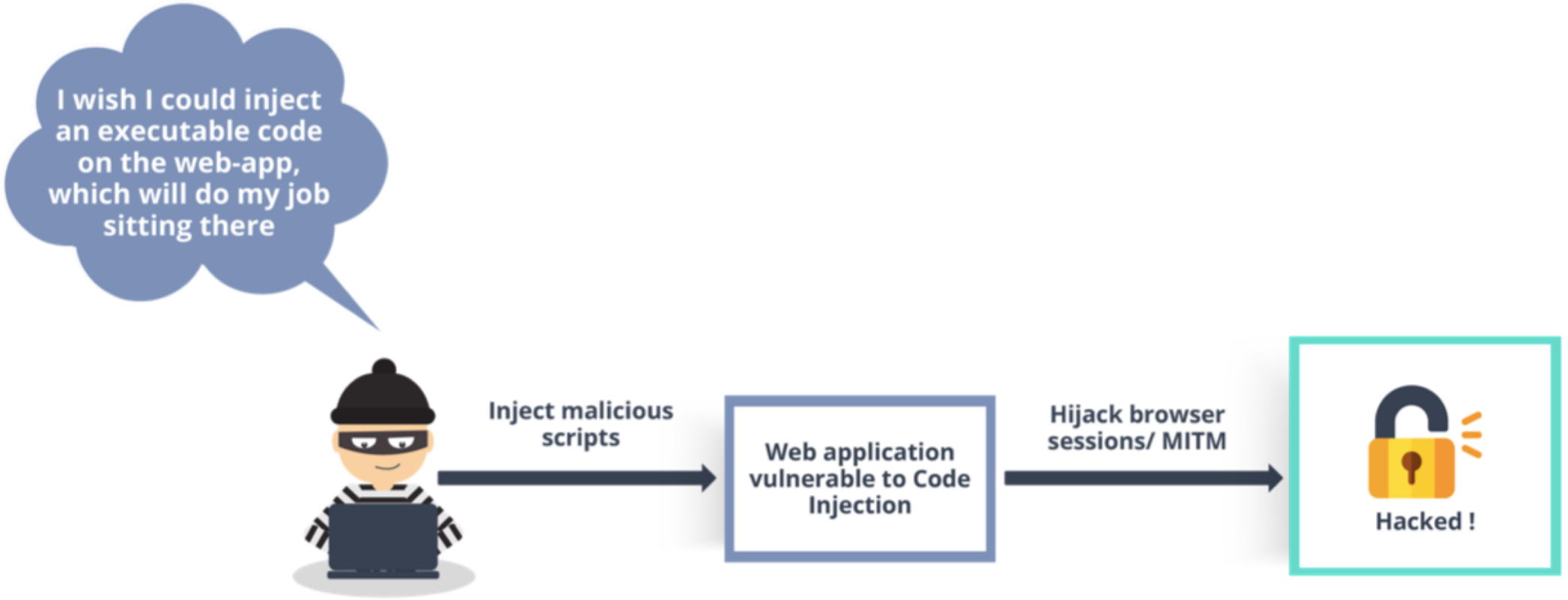
Security Misconfiguration – References

CIS Security Configuration Benchmarks	https://www.cisecurity.org/cis-benchmarks/
OWASP Secure Headers Project	https://www.owasp.org/index.php/OWASP_Secure_Headers_Project
Testing for Configuration Management	https://www.owasp.org/index.php/Testing_for_configuration_management

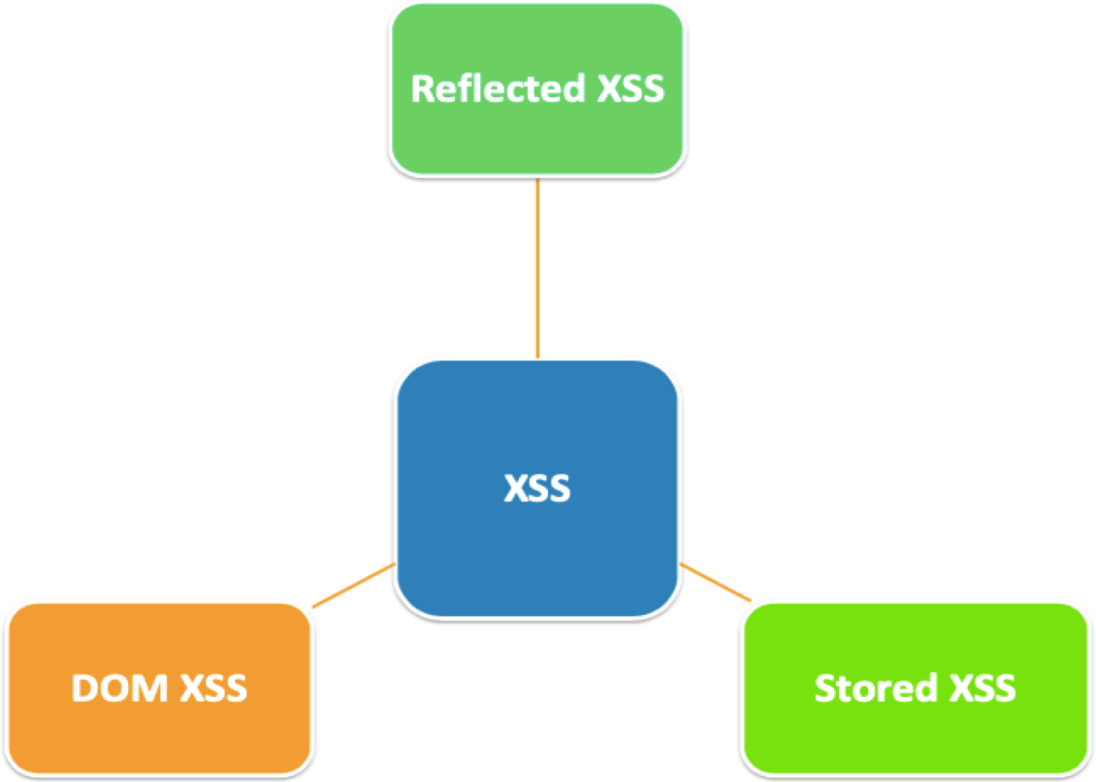
Cross – Site Scripting (XSS)

FTVETI

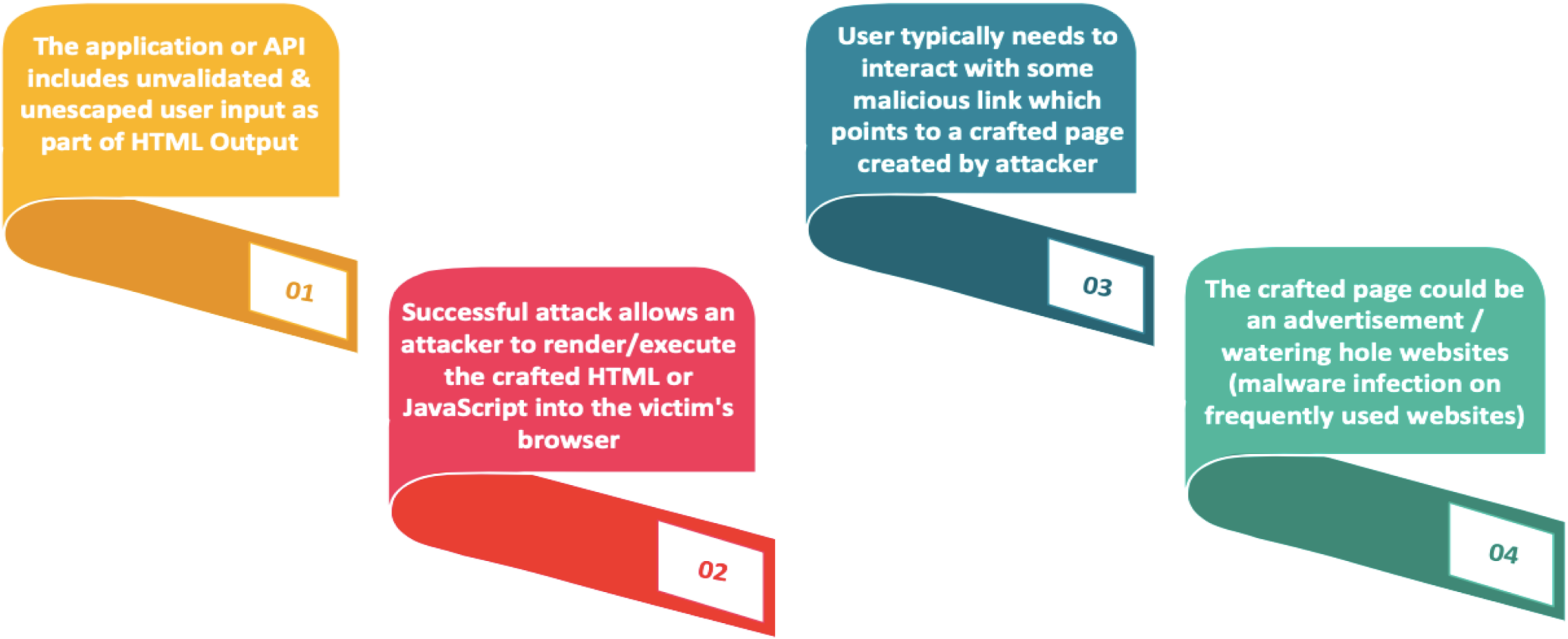
Cross – Site Scripting (XSS)



XSS – Types



Reflected XSS



Stored XSS

The application or API stores un-sanitized user input (crafted code) which gets viewed/loaded/run by another user (or admin) thus victimizing it as the malicious code then runs in the context of the viewer

This is a critical or high risk

DOM XSS

01 DOM refers to Document Object Model

02 The Document Object Model (DOM) is a cross-platform and language-independent application programming interface that treats an HTML, XHTML, or XML document as a tree structure wherein each node is an object representing a part of the document. The objects can be manipulated programmatically and any visible changes occurring as a result may then be reflected in the display of the document

03 When a web page is loaded, the browser creates a Document Object Model of the page

04 The HTML DOM model is constructed as a tree of Objects

05 DOM Based XSS simply means a Cross-site scripting vulnerability that appears in the DOM instead of part of the HTML

XSS Example

Imagine the following page http://www.sample.com/test.html contains the below code:

<script>document.write ("Current URL : " + document.baseURI); </script>

If you send a modified HTTP request :

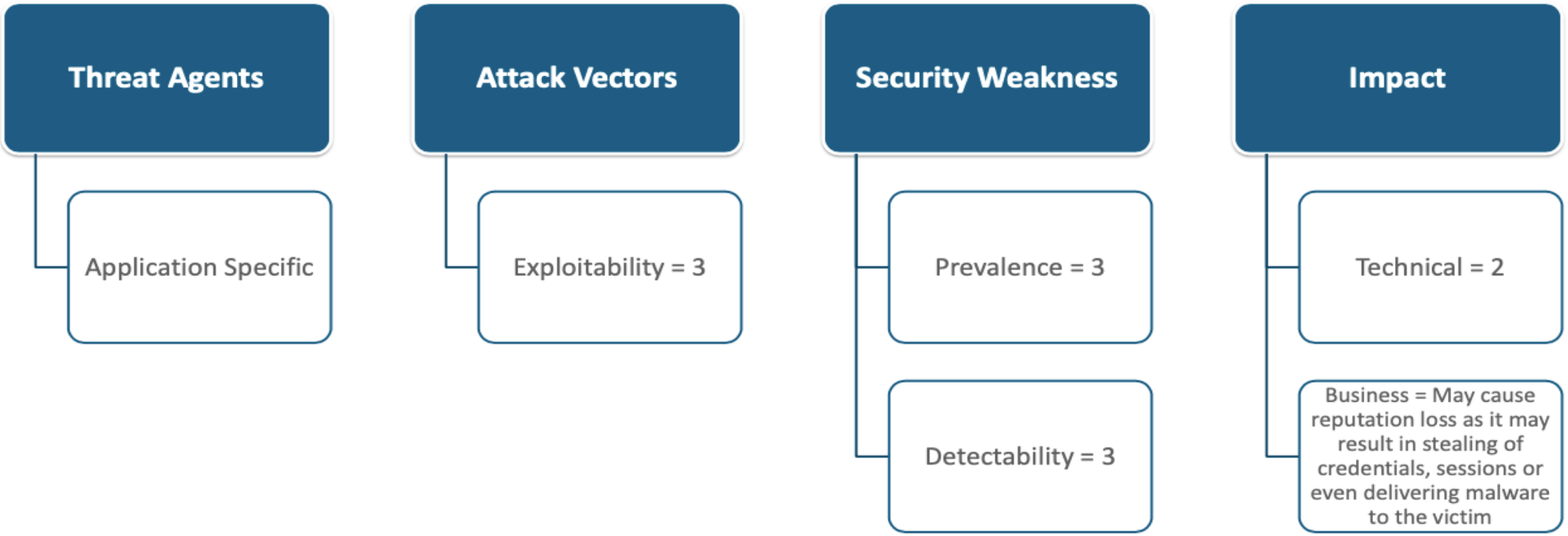
http://www.sample.com/test.html#<script>alert('Oops!')</script>

Injected JavaScript code will get executed, because the page is writing whatever you typed in the URL to the page with document.write function

If you look at the source of the page, you won't see <script>alert('Oops')</script> because it's all happening in the DOM and done by the executed JavaScript code

After the malicious code is executed by page, you can simply exploit this DOM based cross-site scripting vulnerability to steal the cookies of the user or change the page's behavior as you like

XSS – Scorecard

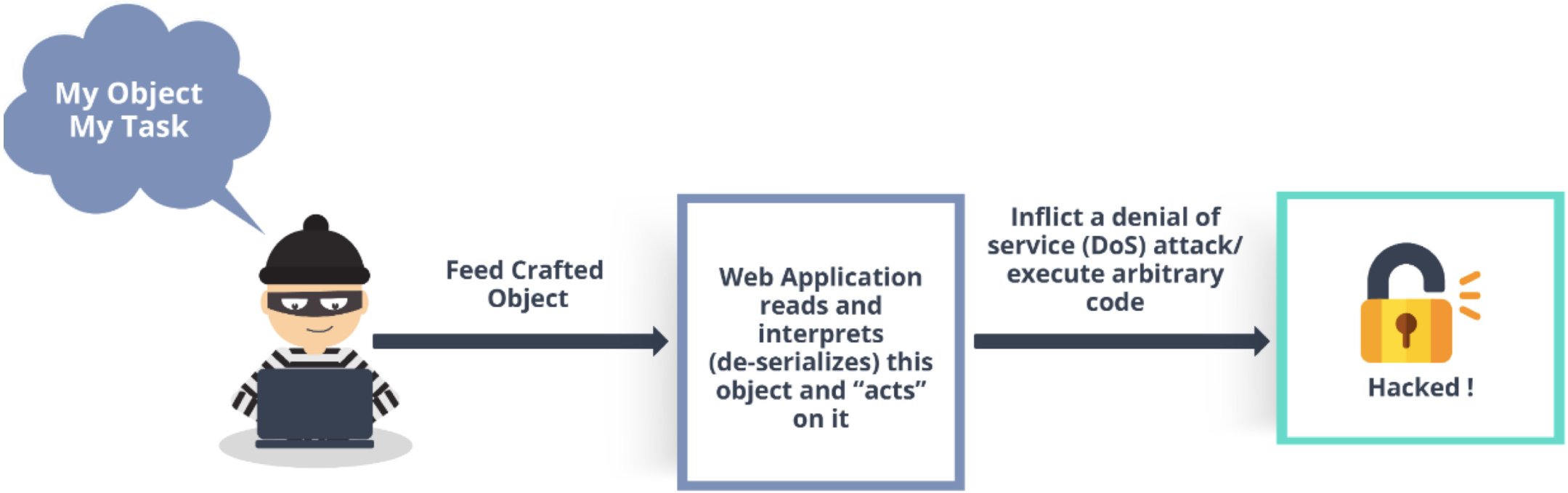


XSS – References

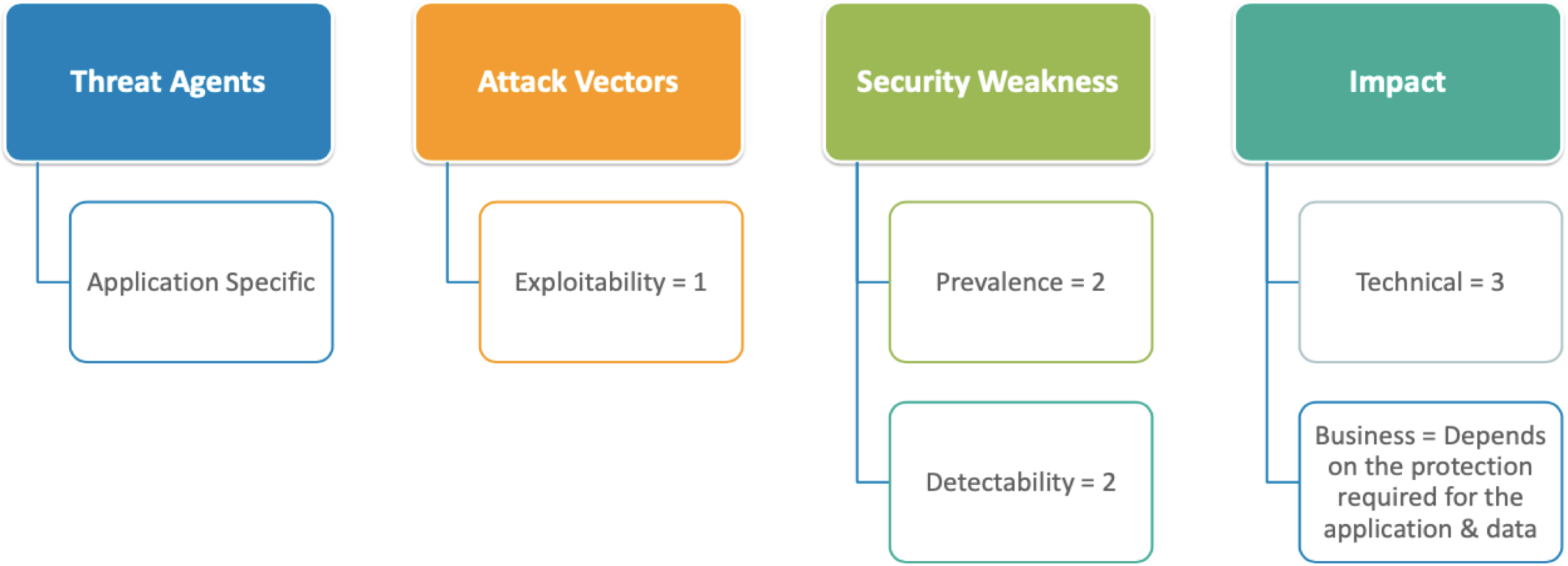
OWASP Proactive Controls	https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=OWASP_Proactive_Controls_2016
Testing for Reflected XSS	https://www.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_(OTG-INPVAL-001)
Testing for Stored XSS	https://www.owasp.org/index.php/Testing_for_Stored_Cross_site_scripting_(OTG-INPVAL-002)
Testing for DOM-based XSS	https://www.owasp.org/index.php/Testing_for_DOM-based_Cross_site_scripting_(OTG-CLIENT-001)
XSS Prevention Cheat Sheet	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
Client-side template injection	https://portswigger.net/kb/issues/00200308_client-side-template-injection

Insecure De-Serialization

Insecure De-Serialization



Insecure De-Serialization – Scorecard



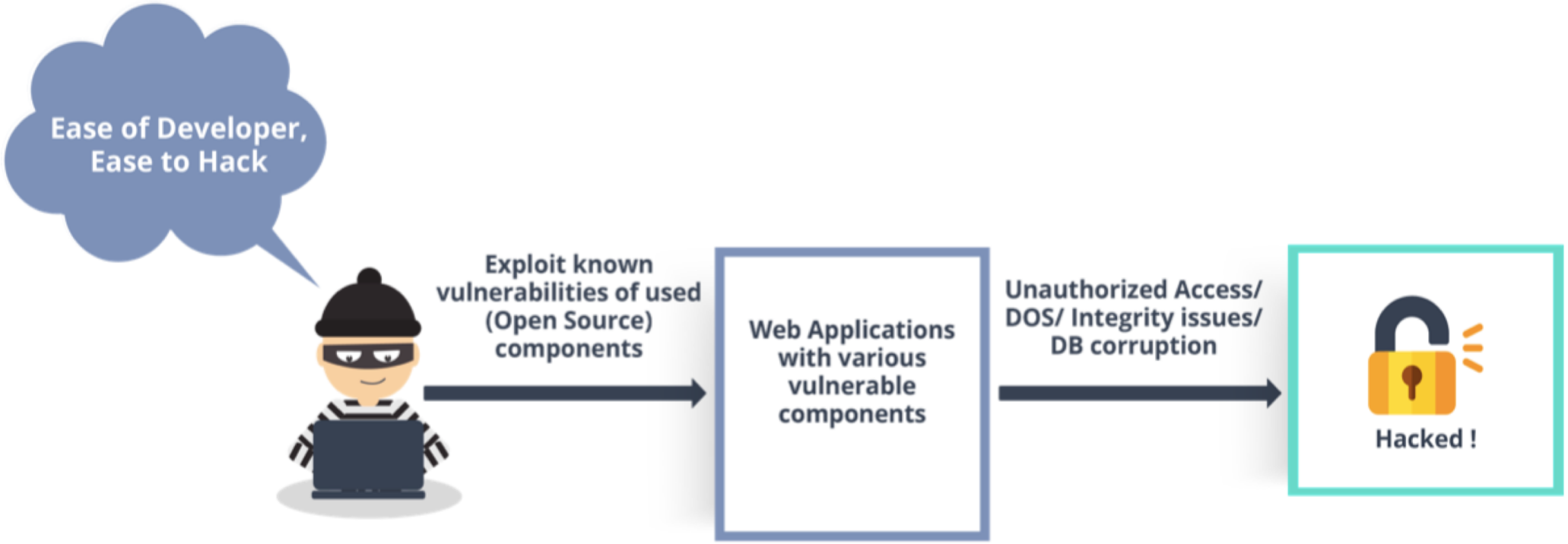
Insecure De-Serialization – References

OWASP Application Security Verification Standard Project	https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home
OWASP Proactive Controls	https://www.owasp.org/index.php/OWASP_Proactive_Controls#4:_Validate_All_Inputs
Deserialization Cheat Sheet	https://www.owasp.org/index.php/Deserialization_Cheat_Sheet

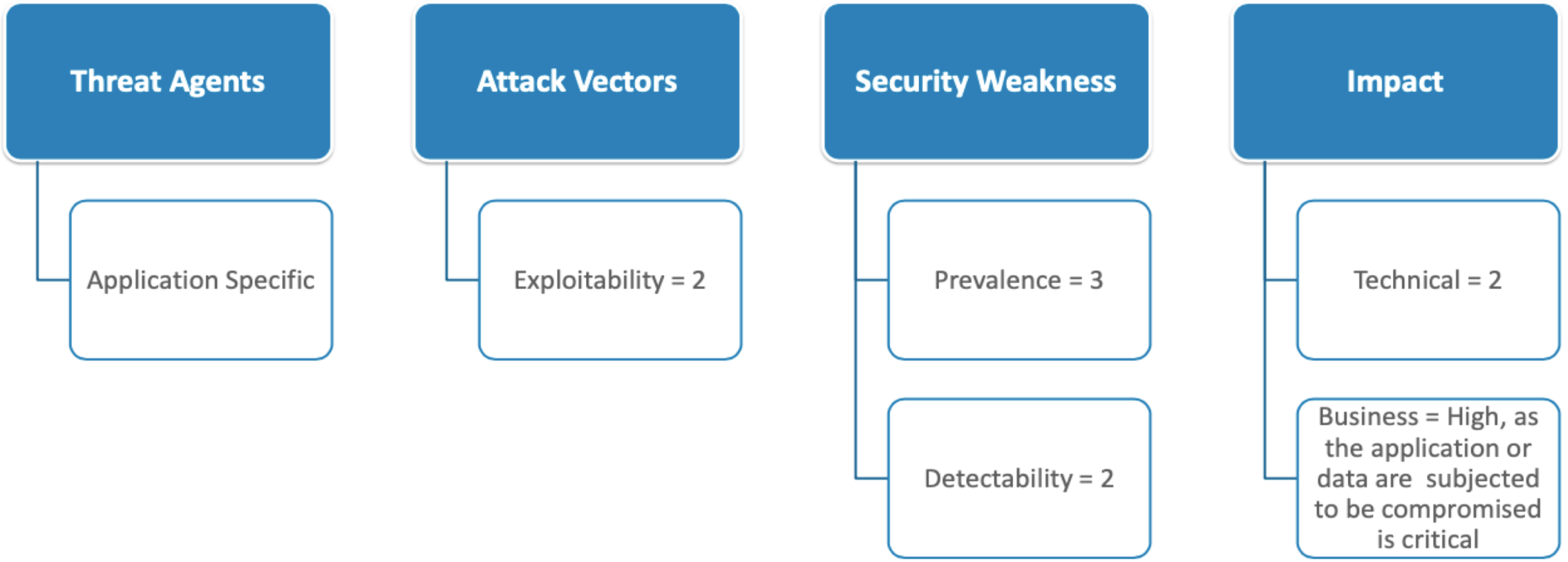
FTVETI

Using Components With Known Vulnerabilities

Using Components With Known Vulnerabilities



Vulnerable Components – Scorecard



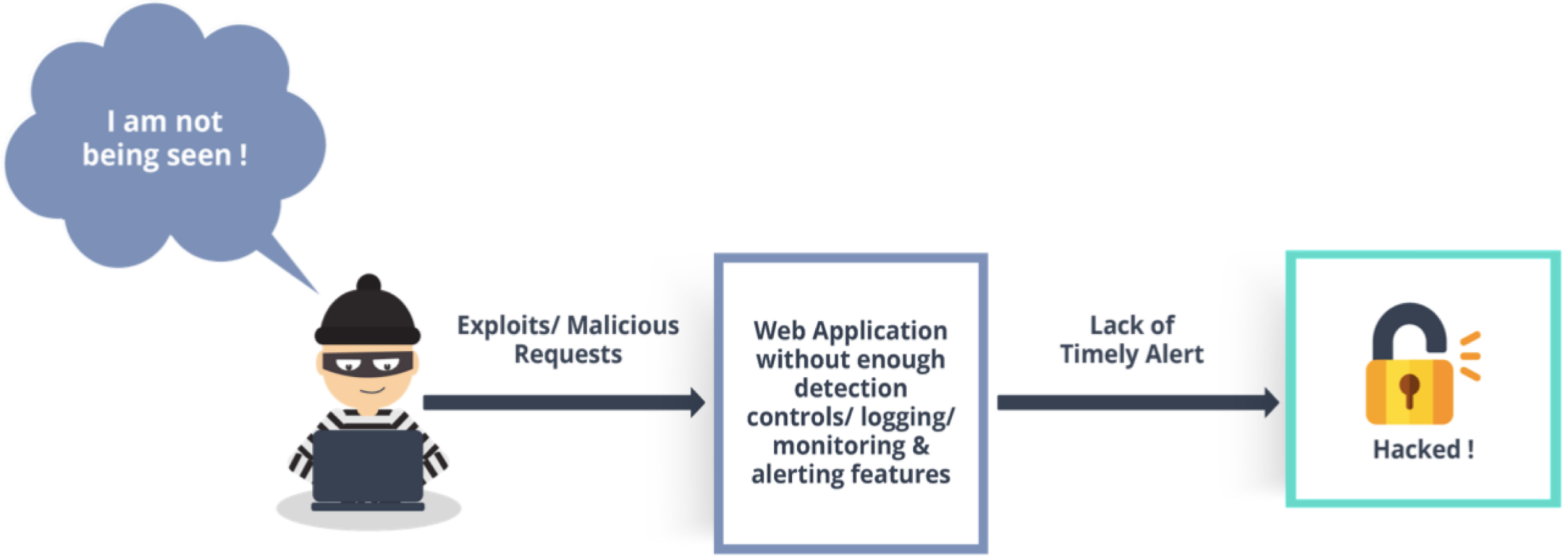
Using Vulnerable Components – References

Mapping the Application Architecture	https://www.owasp.org/index.php/Map_Application_Architecture_(OTG-INFO-010)
CVE Details - Vulnerability Data source	https://www.cvedetails.com/version-search.php
NVD (National Vulnerability Database)	https://nvd.nist.gov/
Virtual Patching Best Practices	https://www.owasp.org/index.php/Virtual_Patching_Best_Practices
Heartbleed Vulnerability	https://en.wikipedia.org/wiki/Heartbleed
POODLE Vulnerability	https://en.wikipedia.org/wiki/POODLE

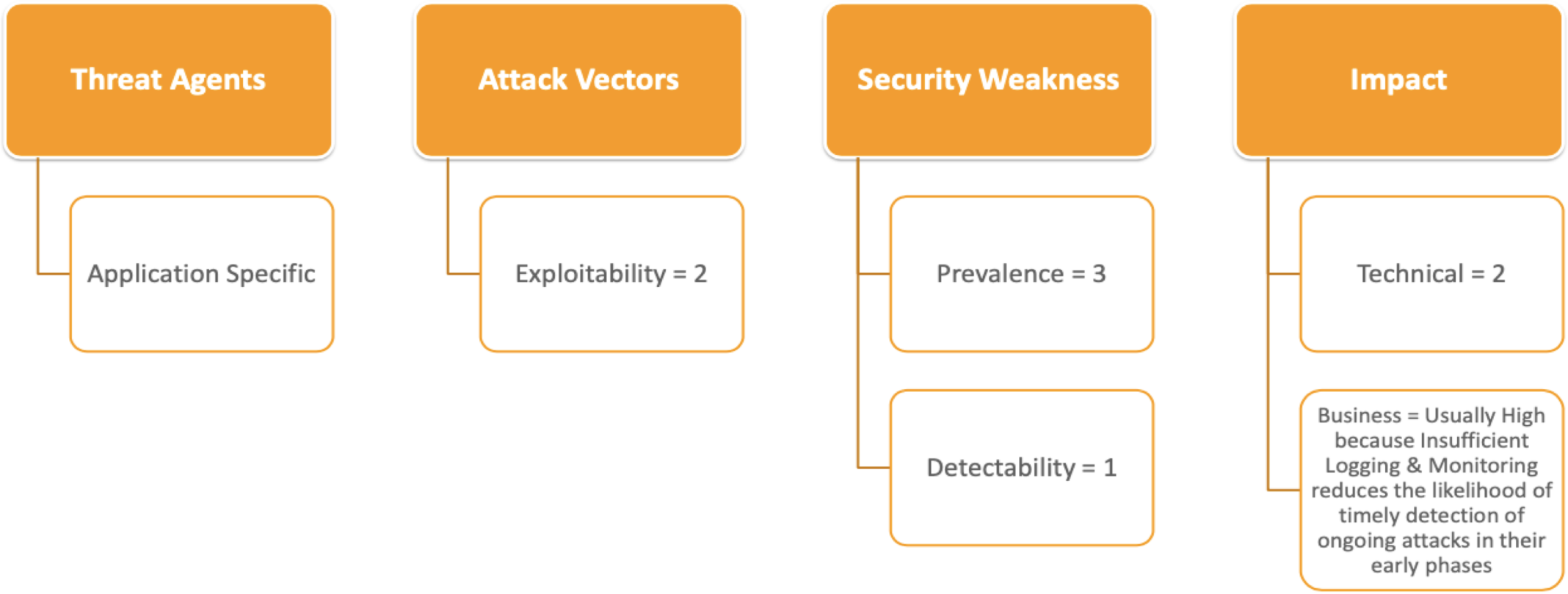
Insufficient Logging & Monitoring

FTVETI

Insufficient Logging & Monitoring



Insufficient Logging & Monitoring – Scorecard



Insufficient Logging & Monitoring – References

OWASP Application Security Verification Standard Project	https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home
Logging Cheat Sheet	https://www.owasp.org/index.php/Logging_Cheat_Sheet

Secure Software Development Life Cycle

Software Assurance Maturity Model (SAMM)

SAMM – Characteristics

01 Flexible framework

02 Fit to be utilized by organizations with any size

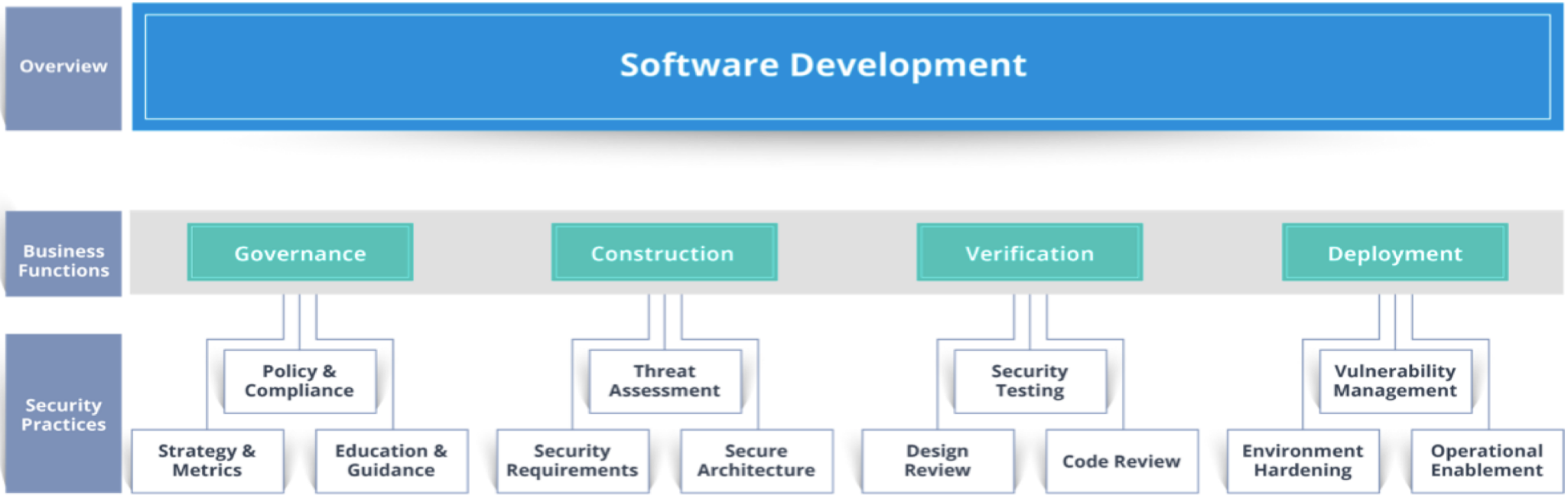
03 Fits any development methodology or style

04 Potential to be applied organization – wide/ single line–of–business/ an individual project

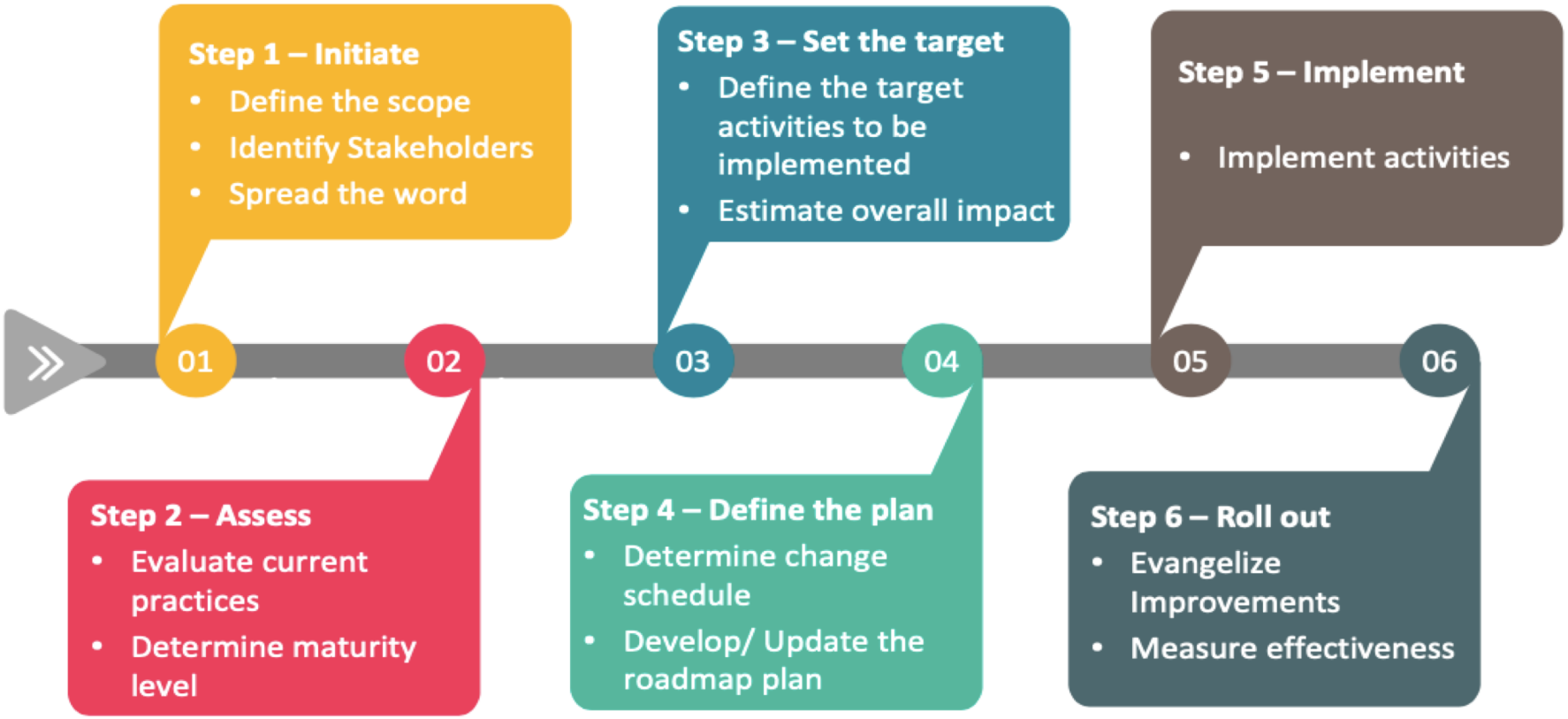
05 Vendor neutral & freely available

SAMM Overview

- The basis of this model stands for the core business functions of software development with interwoven security practices
- The building blocks of the model are 3 maturity levels defined for each of the 12 security practices
- These define a wide variety of activities in which an organization could engage to increase software assurance & minimize security risks

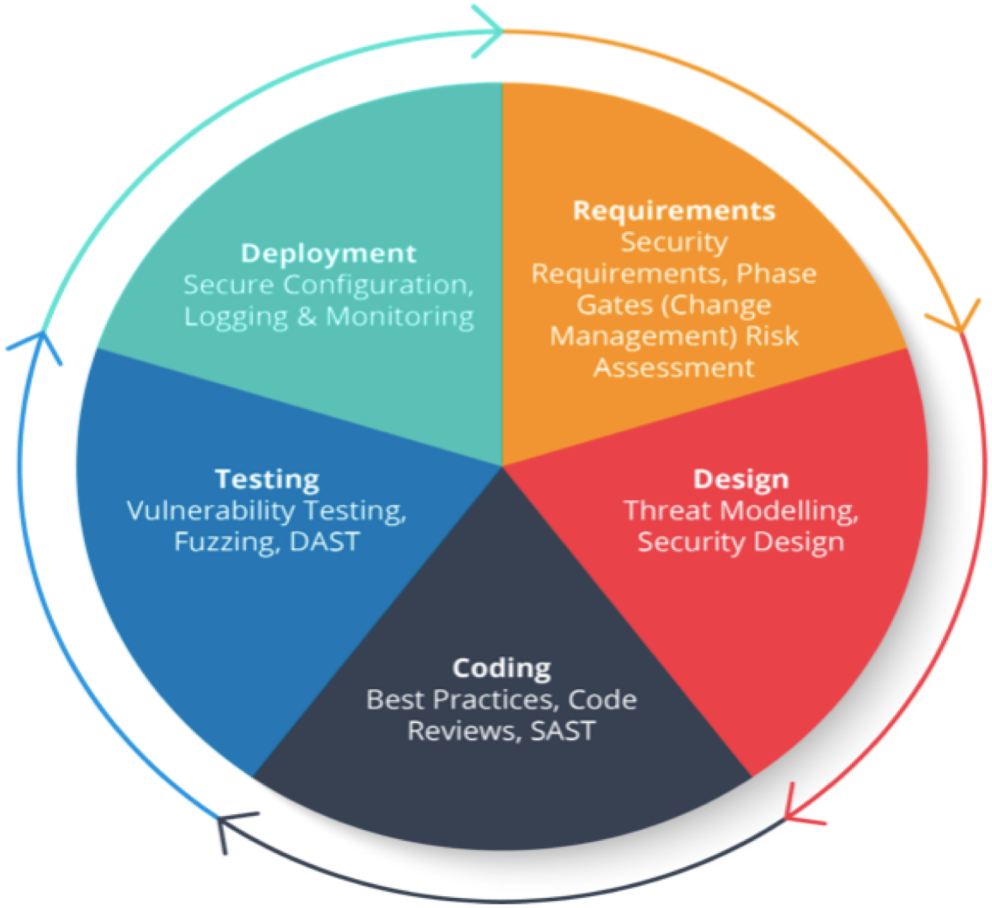


Application Of SAMM



Traditional SDLC – Phases

Secure SDLC – Phases



Demo 2: Buffer Overflow

- Increase the security by preventing buffer overflows and other malicious exploits

Install and configure **BufferShield** from the below link: https://www.sys-manage.com/BufferShield/tabid/61/Default.aspx

Quiz #1

- A Website that allows users to enter text, such as a comment or a name, and then stores it and later displays it to other users, is potentially vulnerable to which type of attack?
  a. MFA (Multi Factor Authentication)
  b. CSRF (Cross-Site Request Forgery)
  c. XSS (Cross-Site Scripting)
  d. Brute-Forcing attacks

Answer #1

- A Website that allows users to enter text, such as a comment or a name, and then stores it and later displays it to other users, is potentially vulnerable to which type of attack?
  a. MFA (Multi Factor Authentication)
  b. CSRF (Cross-Site Request Forgery)
  c. **XSS (Cross-Site Scripting)**
  d. Brute-Forcing attacks

Answer c:
Explanation: In such an attack, a malicious user enters code written in a client-side scripting language such as JavaScript or Flash instead of entering a valid name or comment

Quiz #2

- A website implements 2-Factor authentication for user login. However, it may still be intrinsically vulnerable to which form of attack?
  a. SQLi (SQL Injection)
  b. Brute force attack
  c. SMURF attack
  d. Man-in-the-middle attack (MITM)

Answer #2

- A website implements 2-Factor authentication for user login. However, it may still be intrinsically vulnerable to which form of attack?
  a. SQLi (SQL Injection)
  b. Brute force attack
  c. SMURF attack
  d. Man-in-the-middle attack (MITM)

Answer d:
Explanation: MITM is very natural here. Usually this would start with a fake, phishing page looking exactly like the target website & when the user uses it - he may or may not be directed to the 2FA

Quiz #3

- URL filtering may restrict access to Internet sites based on which of the following criteria?
  a. Virus/Malware signature
  b. Web address
  c. Configuration Baseline
  d. Website page content

Answer #3

- URL filtering may restrict access to Internet sites based on which of the following criteria?
  a. Virus/Malware signature
  b. Web address
  c. Configuration Baseline
  d. Website page content

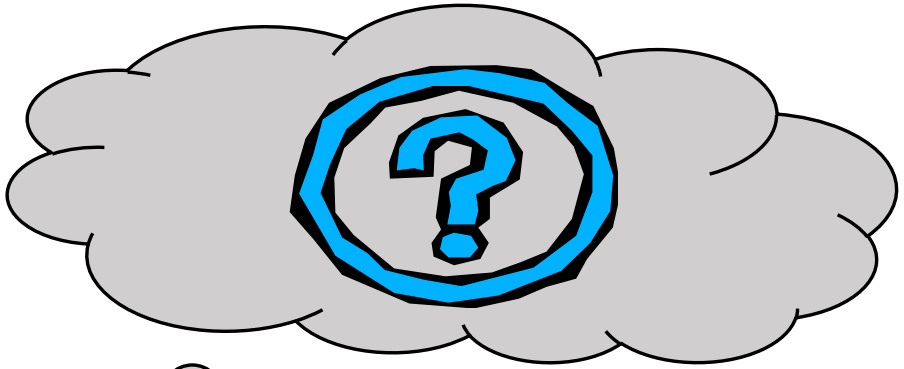
Answer b:
Explanation: URL Filtering typically works on a concept of RBL (Rejection Black List). The RBL contains a list of known bad URLs which are not allowed to be visited by an endpoint or browser

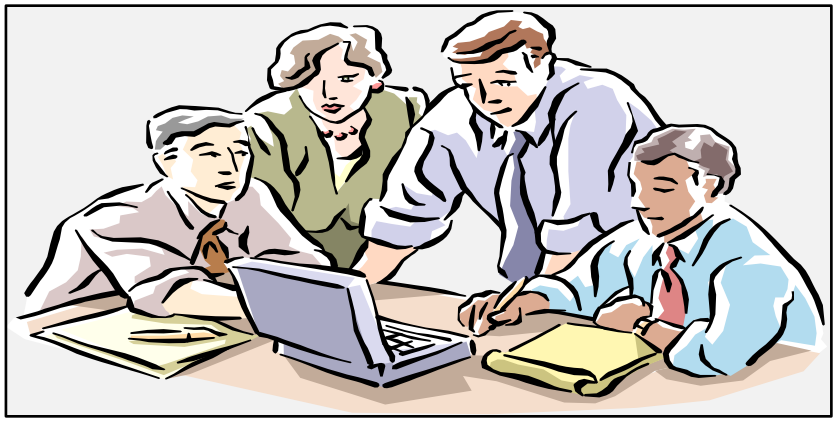
Summary

In this unit, you should have learnt:

- What is Application Security?
- Approach towards Web Application Security
- OWASP — Top 10 Web Application Vulnerabilities
- SSDLC (Secure Software Development Life Cycle)

QUESTIONS PLEASE ☺





FTVETI

ICT @ FTVETI