OWASP Top 10 Schulung

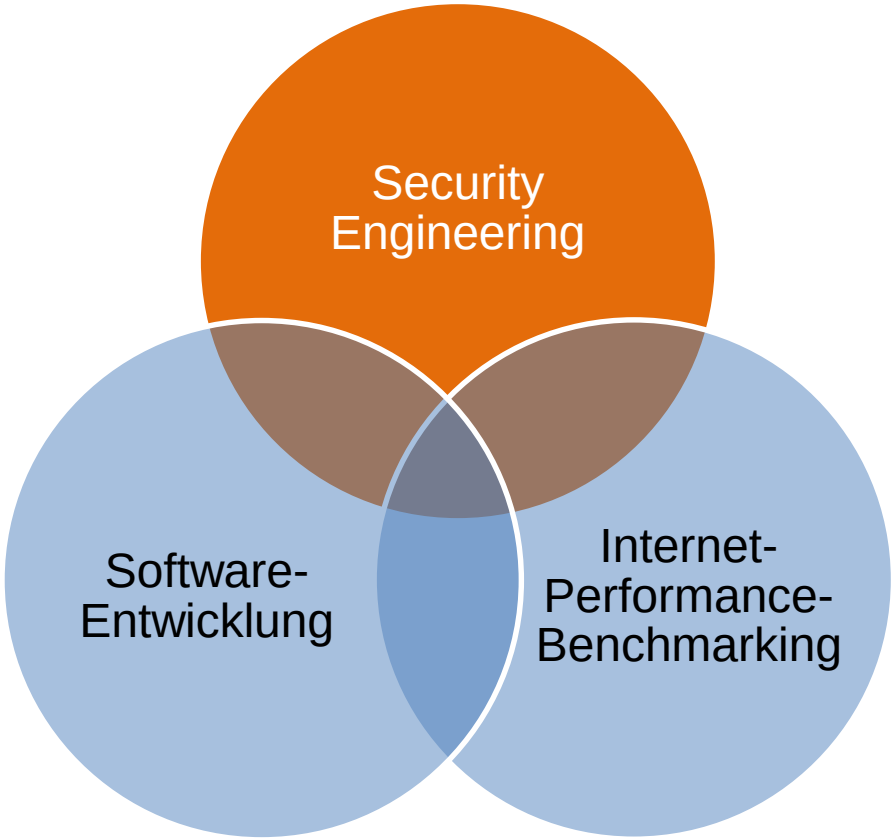
Agenda

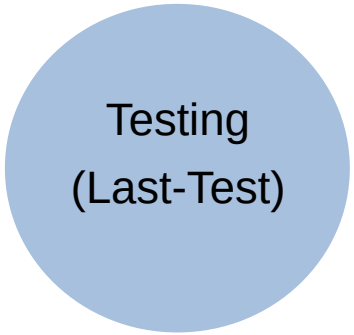
- Übersicht
- A1:2017 – Injection
- A2:2017 – Broken Authentication
- A3:2017 – Sensitive Data Exposure
- A4:2017 – XML External Entities (XXE)
- A5:2017 – Broken Access Control
- A6:2017 – Security Misconfiguration
- A7:2017 – Cross-Site Scripting (XSS)
- A8:2017 – Insecure Deserialization (Community)
- A9:2017 – Using Components with Known Vulnerabilities
- A10:2017 – Insufficient Logging & Monitoring (Community)
- What’s Next for Developers?
- Übungen an WebGoat

Cnlab Gruppe

cnlab Organisation
- 15 Ingenieure
- Sitz Rapperswil / SG
- Im Besitz des Managements

cnlab Partner
- HSR (Hochschule Rapperswil)
- ETH Zürich
- CSI Consulting





Aufbau

- Worum geht's?
- Wie finden?
- Was dagegen tun?

Threat Agents	Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts	Business Impacts
Appli-cation Specific	Easy: 3	Widespread: 3	Easy: 3	Severe: 3	Business Specific
	Average: 2	Common: 2	Average: 2	Moderate: 2	
	Difficult: 1	Uncommon: 1	Difficult: 1	Minor: 1	

Begriffe [1]

- OWASP Top 10 Web Application Security Risks
- OWASP: Open Web Application Security Project
- Schwachstelle hinter dem Risiko wird hier betrachtet.

Deutsch	Englisch	Berechnung
Bedrohung	threat	
Gefahr, Gefährdung	applied threat	F(threat, weakness)
Schwachstelle	weakness	
Risiko	risk	Probability*Intensity

1. https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/Glossar/glossar_node.html

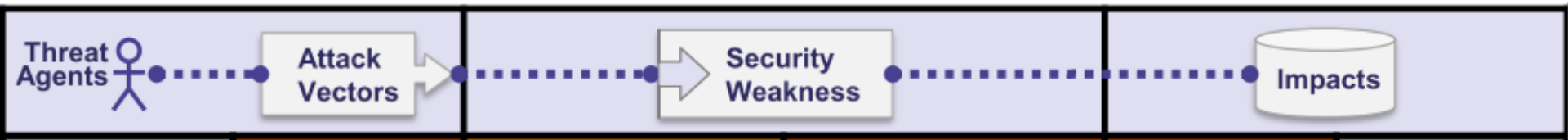
Entwicklung der OWASP Top 10

OWASP Top Ten - 2010	OWASP Top Ten - 2013	OWASP Top Ten – 2017
A1 – Injection Flaws	A1 – Injection Flaws	A1 – Injection Flaws
A2 - Cross Site Scripting (XSS)	A2 - Broken Authentication and Session Management	A2 - Broken Authentication and Session Management
A3 - Broken Authentication and Session Management	A3 - Cross Site Scripting (XSS)	A3 – Sensitive Data Exposure
A4 - Insecure Direct Object Reference	A4 - Insecure Direct Object Reference (fliesst in 2017 A5 ein)	A4 – XML External Entities (XXE)
A5 - Cross Site Request Forgery (CSRF)	A5 - Security Misconfiguration	A5 – Broken Access Control
A6 - Security Misconfiguration	A6 - Sensitive Data Exposure	A6 – Security Misconfiguration
A7 - Insecure Cryptographic Storage (fliesst in 2013 A6 ein)	A7 - Missing Function Level Access Control (fliesst in 2017 A5 ein)	A7 – Cross-Site Scripting (XSS)
A8 - Failure to Restrict URL Access	A8 - Cross Site Request Forgery (CSRF)	A8 – Insecure Deserialization
A9 - Insufficient Transport Layer Protection (fliesst in 2013 A6 ein)	A9 - Using Known Vulnerable Components	A9 - Using Known Vulnerable Components
A10 - Unvalidated Redirects and Forwards (Fliesst in 2013 A6 ein)	A10 - Unvalidated Redirects and Forwards	A10 – Insufficient Logging & Monitoring

Im nächsten Rel. gelöscht

Neu

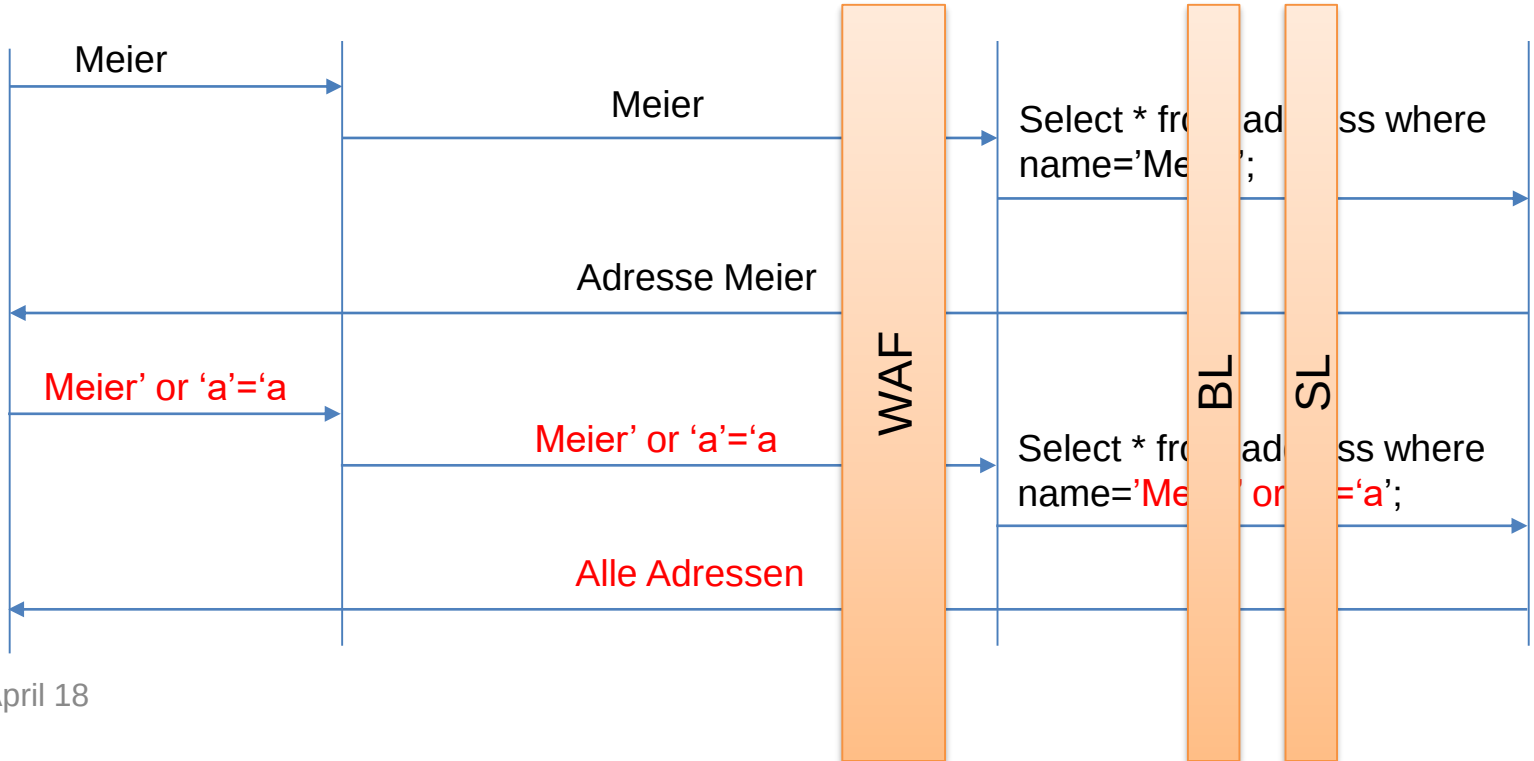
A1:2017 – Injection - Worum geht's?



App. Specific	Exploitability: 3	Prevalence: 2	Detectability: 3	Technical: 3	Business ?
Almost any source of data can be an injection vector, environment variables, parameters, external and internal web services, and all types of users. Injection flaws occur when an attacker can send hostile data to an interpreter.		Injection flaws are very prevalent, particularly in legacy code. Injection vulnerabilities are often found in SQL, LDAP, XPath, or NoSQL queries, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries. Injection flaws are easy to discover when examining code. Scanners and fuzzers can help attackers find injection flaws.		Injection can result in data loss, corruption, or disclosure to unauthorized parties, loss of accountability, or denial of access. Injection can sometimes lead to complete host takeover. The business impact depends on the needs of the application and data.	

Ein guter WAF findet die Standard-Patterns

A1:2017 – Injection - Worum geht's?



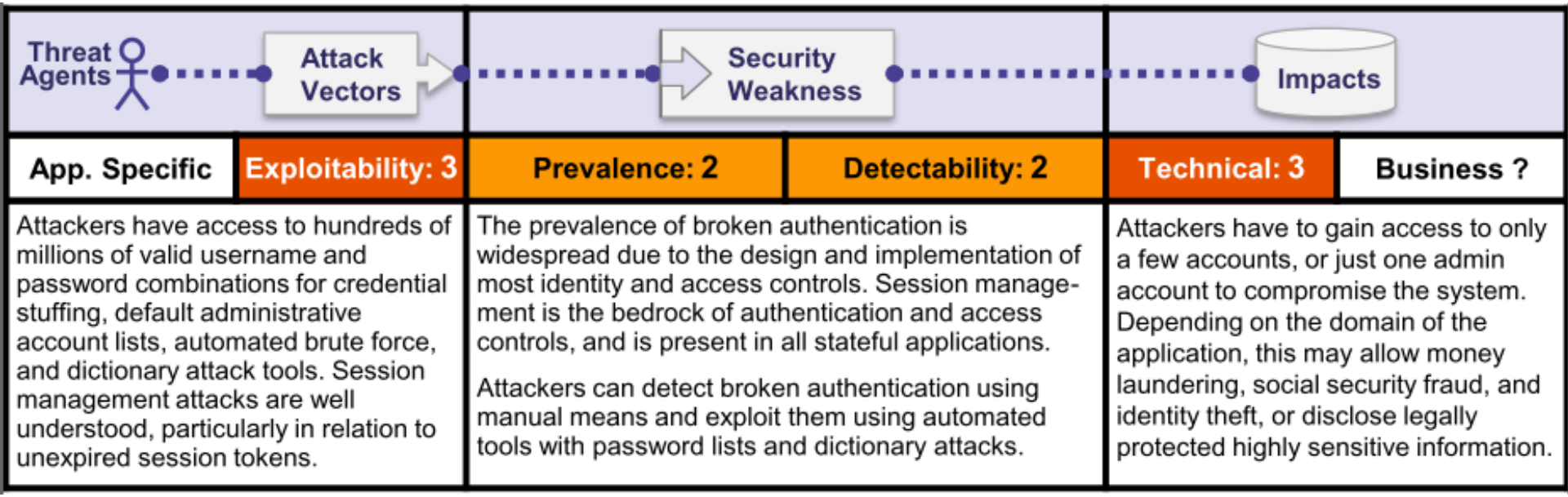
A1:2017 – Injection - Wie finden?

- Code-Review auf sicheren Gebrauch von Interpretern
  - Aufruf von Interpretern muss zwischen Eingabedaten und Befehlen unterscheiden
- Scanner können hilfreich sein.
  - SAST (Static Application Security Testing, Source-Code-Analyse)
  - DAST (Dynamic Application Security Testing)

A1:2017 – Injection - Was dagegen tun?

- Nutzung einer sicheren API
- Rechte der Anwendung limitieren
- Validierung des User-Input auf dem Server per «Whitelist»
- Metazeichen entschärfen
- Stored Procedures verwenden
- Keine detaillierten Fehlermeldungen

A2:2017 – Broken Authentication and Session Management - Worum geht's?

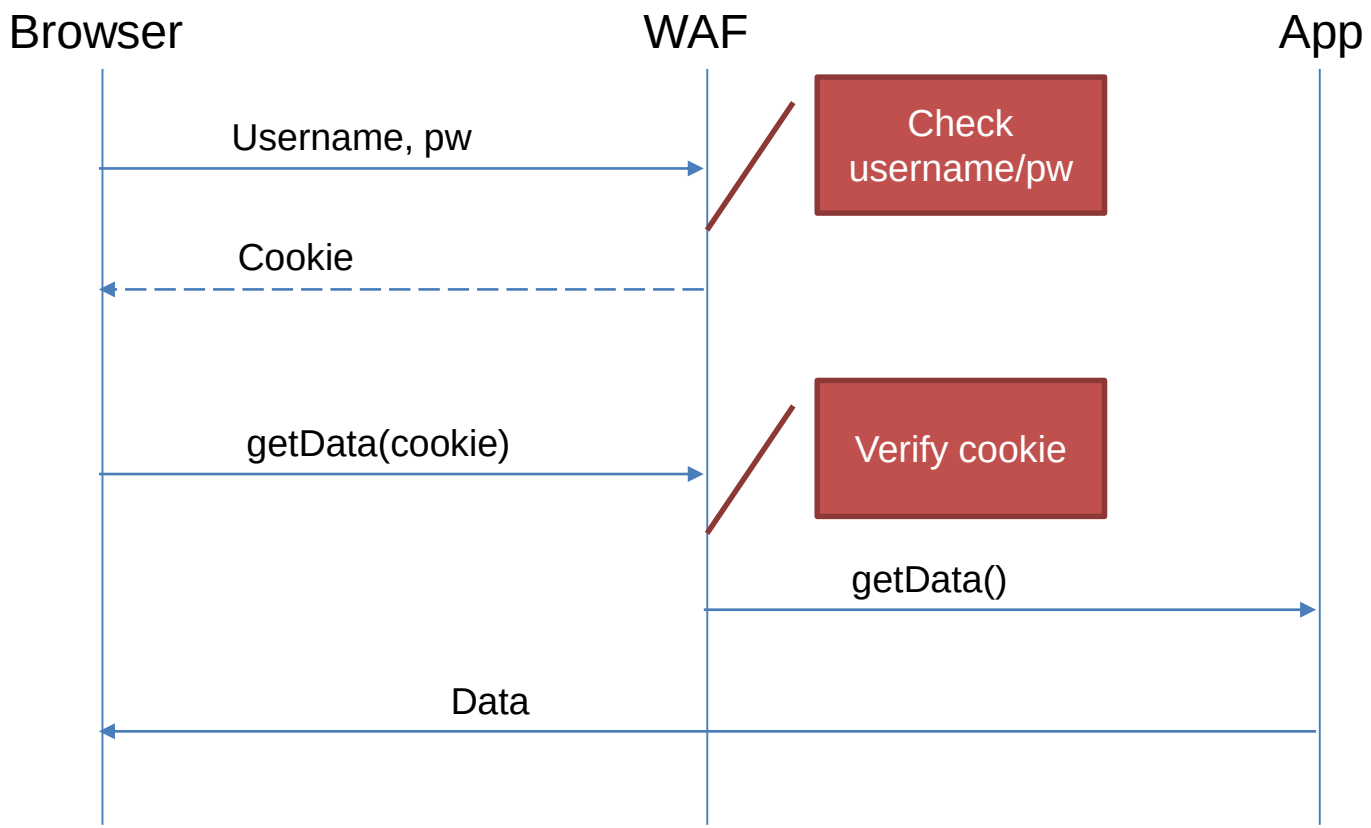


- Authentisierung und Session Management sind zentral für die Sicherheit von Web-Applikationen.

A2:2017 – Broken Authentication and Session Ma

Worum geht's?

Ein guter WAF macht ein sicheres Session-Management



A2:2017 – Broken Authentication and Session Management - Wie finden?

- Wie wird der Benutzer authentisiert?
- Wie werden die Benutzer-Passwörter gespeichert?
- Wie wird die Benutzererkennung erstellt bzw. verwaltet?
- Wie werden Passwörter, Session IDs und Benutzername übertragen?
- Handhabung von Session IDs?
  - Sichtbar in der URL?
  - Session-Fixation möglich?
  - Sessions laufen nicht ab / Token werden beim Logout nicht invalidiert

A2:2017 – Broken Authentication - Was dagegen tun?

- Standard-Mechanismen für die Authentisierung verwenden
- Multi-Faktor-Authentisierung verwenden
- Default-Zugänge entfernen / Passwort ändern
- Starke Passwort-Policy (z.B. NIST Richtlinien[1]) / Schwache Passwörter ausschliessen
- Passwörter nur verschlüsselt übertragen
- Sichere Passwort-Wiederherstellung
- Hardening von Registrierung, Passwort-Wiederherstellung, API-Zugriffen
- Detektion von und Alarmierung bei Angriffen (z.B. Brute Force)
- Gescheiterte Login-Versuche limitieren oder verzögern

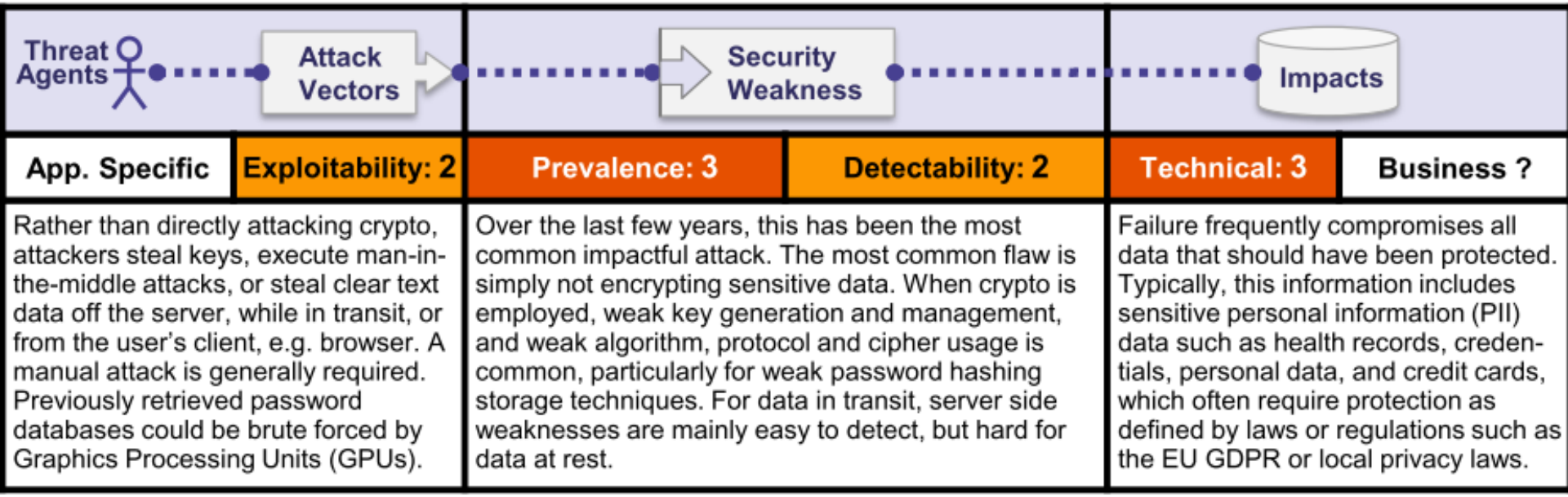
1. https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret

A2:2017 – Broken Authentication and Session Management - Was dagegen tun?

- Zufällige Session IDs
- Session ID über HTTPS
- Cookies HTTPOnly
- Logout-Funktion
- Timeout
- Serverseitiger Session-Manager: Session ID zufällig, geschützt, nach Logout und Timeout invalidiert

Ein guter WAF garantiert, dass alle Internet-Verbindungen chiffriert sind

A3:2017 – Sensitive Data Exposure - Worum geht's?



- Z.B. Passwörter und Bankkundendaten in Logfiles
- Passwörter als Klartext gespeichert.

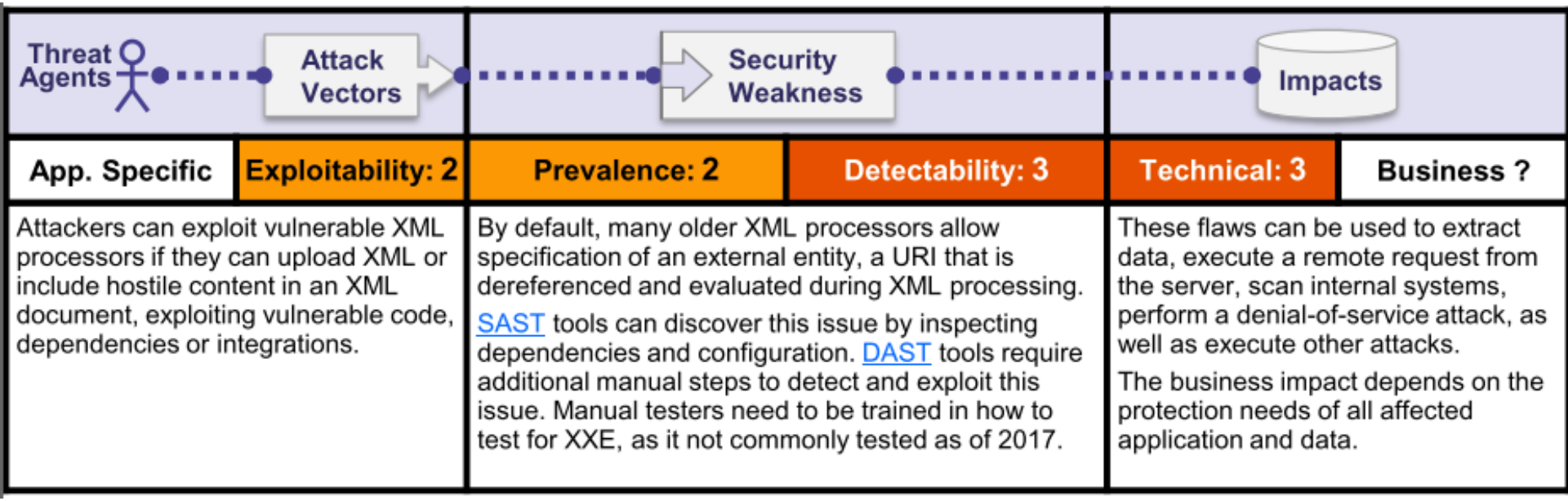
A3:2017 – Sensitive Data Exposure - Wie finden?

- Welche Daten sind besonders schützenswert?
- Wo und wie werden diese Daten gespeichert, inkl. Backup?
- Wie werden diese Daten übertragen?
- Werden schwache / veraltete Kryptoverfahren / veraltete SSL Versionen genutzt?
- Werden schwache Schlüssel erzeugt?
- Fehlt eine geeignete Schlüsselverwaltung?
- Wird Verschlüsselung erzwungen?

A3:2017 – Sensitive Data Exposure - Was dagegen tun?

- Daten klassifizieren
- Restriktive Rechtevergabe («deny by default»)
- Nur nötige Daten speichern
- Schützenswerte Daten verschlüsseln (Speicherung und Übertragung)
- Starke Algorithmen und Schlüssel verwenden
- Angemessene Schlüsselverwaltung
- HTTP Strict Transport Security (HSTS)
- Passwörter mit angemessener Hash-Funktion mit Salt speichern (z.B. PBKDF2)
- Autovervollständigung und Caching für Formulare mit vertraulichen Inhalten deaktivieren

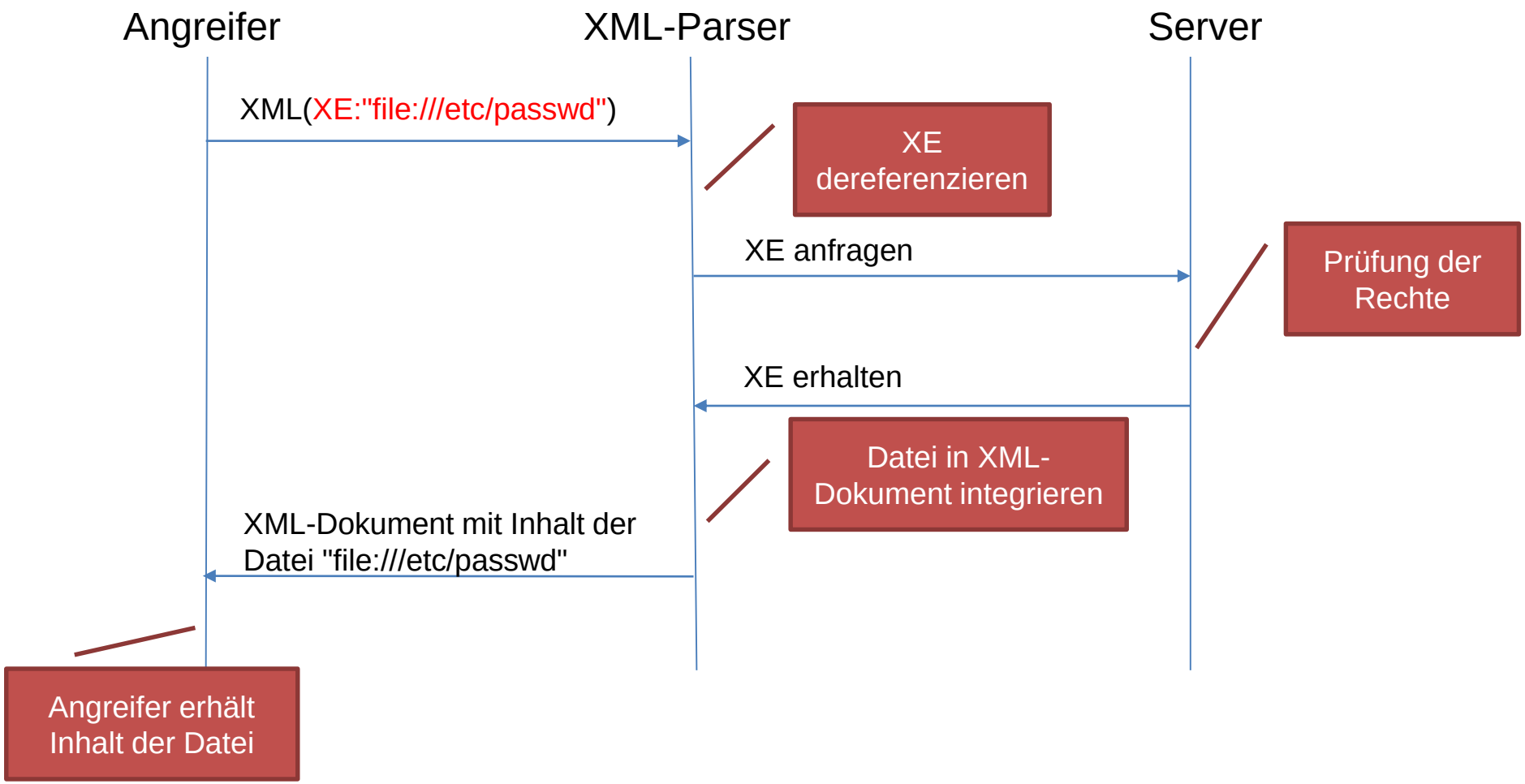
A4:2017 – XML External Entities (XXE) - Worum geht's?



- SAST (Static Application Security Testing → Source Code Review
- DAST (Dynamic Application Security Testing)

Ein WAF kann DTD verbieten und damit XXE verhindern.

A4:2017 – XML External Entities (XXE) - Worum geht's?



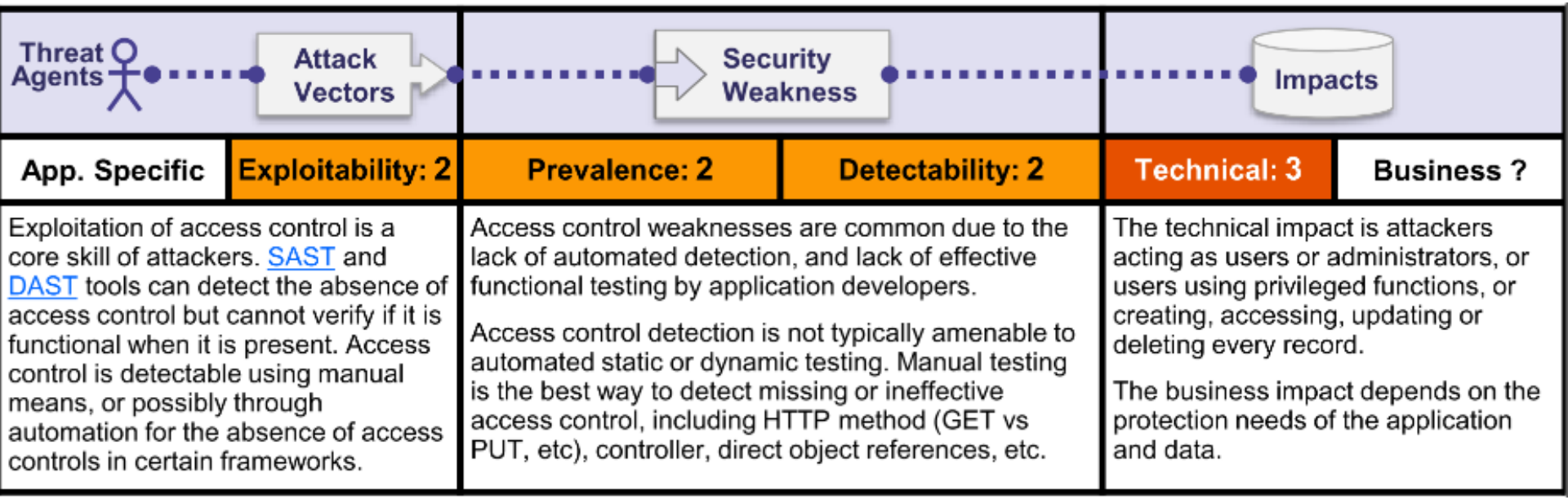
A4:2017 – XML External Entities (XXE) - Wie finden?

- Source Code Analysis Tools (SAST)
- Dynamic Application Security Testing (DAST)
- Manuelle Tests
  - Akzeptiert die Anwendung XML (direkt oder als Upload)?
  - Document Type Definitions (DTD) sind aktiviert
  - SAML kann anfällig sein
  - SOAP vor V1.2
- «Billion Laughs»

A4:2017 – XML External Entities (XXE) - Was dagegen tun?

- Schulungen
- Nur Dateiformate ohne Referenzen von nicht vertrauenswürdigen Quellen akzeptieren
- XML-Prozessoren und -Bibliotheken aktuell halten
- SOAP 1.2 oder neuer verwenden
- XML External Entity und DTD abschalten
- Serverseitiges Whitelisting
- Code-Review
  - Manuell
  - SAST (Source Code Analysis Tools)

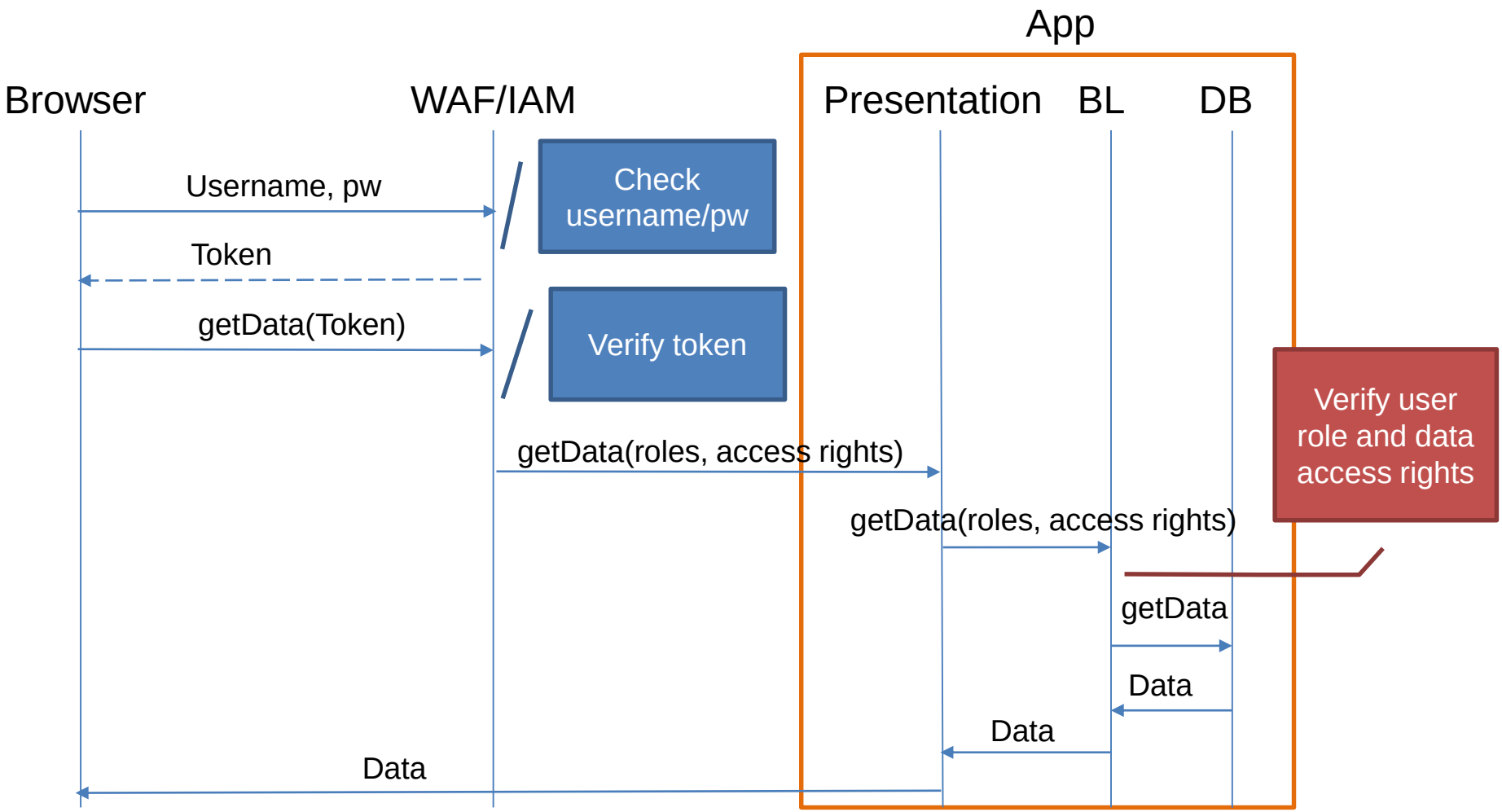
A5:2017 – Broken Access Control - Worum geht's?



- SAST (Static Application Security Testing → Source Code Review
- DAST (Dynamic Application Security Testing)

Häufig kommen die Rollen aus dem IAM. Die Anwendungen müssen sie umsetzen.

A5:2017 – Broken Access Control - Worum geht's?



A5:2017 – Broken Access Control - Wie finden?

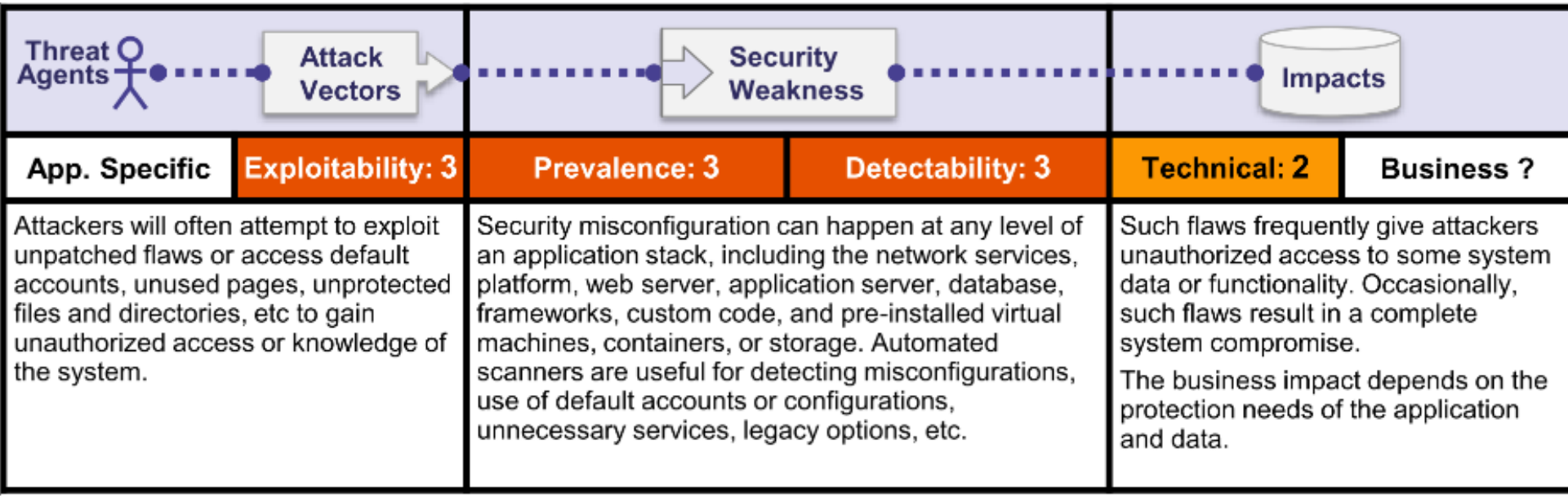
- DAST (Dynamic Application Security Testing) / SAST (Source Code Analysis Tools) erkennen fehlende Zugangskontrolle

  ABER: Funktionsfähigkeit wird nicht bewertet!
- Manuelle Tests (Fachbezug notwendig)

A5:2017 – Broken Access Control - Was dagegen tun?

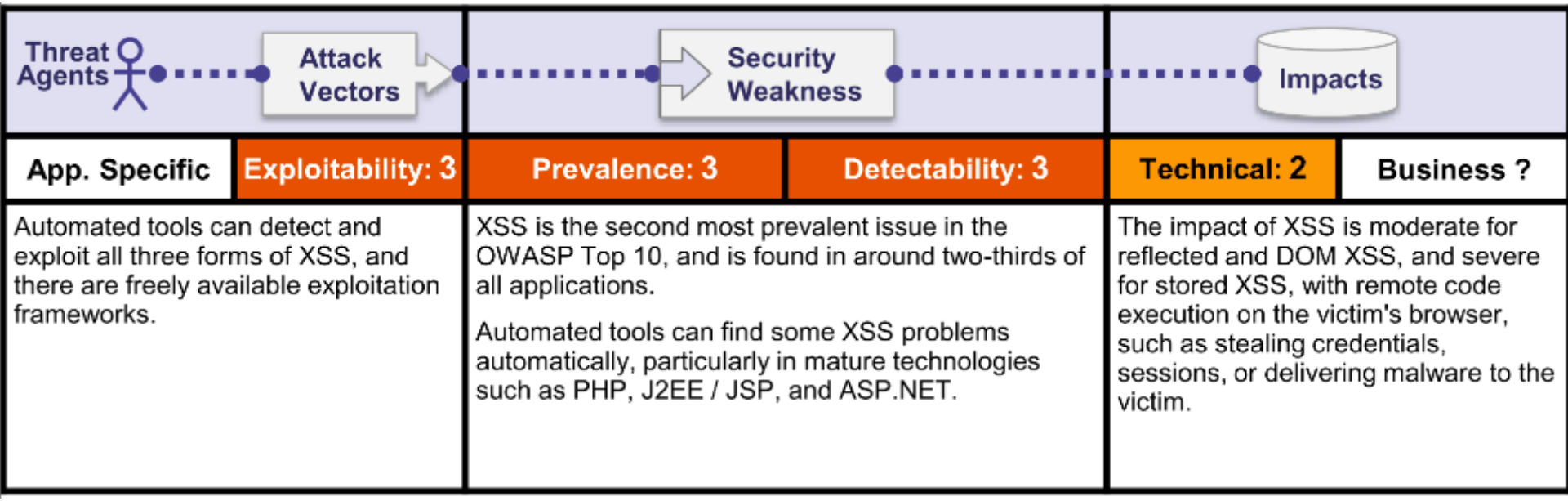
- Berechtigungszuweisung nach «Deny by default» (Ausnahme: öffentliche Ressourcen)
- Zentrale Prüfung von Zugriffsrechten
- Indirekte / zufällige Objektreferenzen wählen
- Jeden Zugriff auf ein Objekt autorisieren
- Token und Referenzen nach Logout invalidieren

A6:2017 – Security Misconfiguration



- Auf Anwendungsstufe Verantwortung des Entwicklers (z.B. Libraries bei Mobile-App).

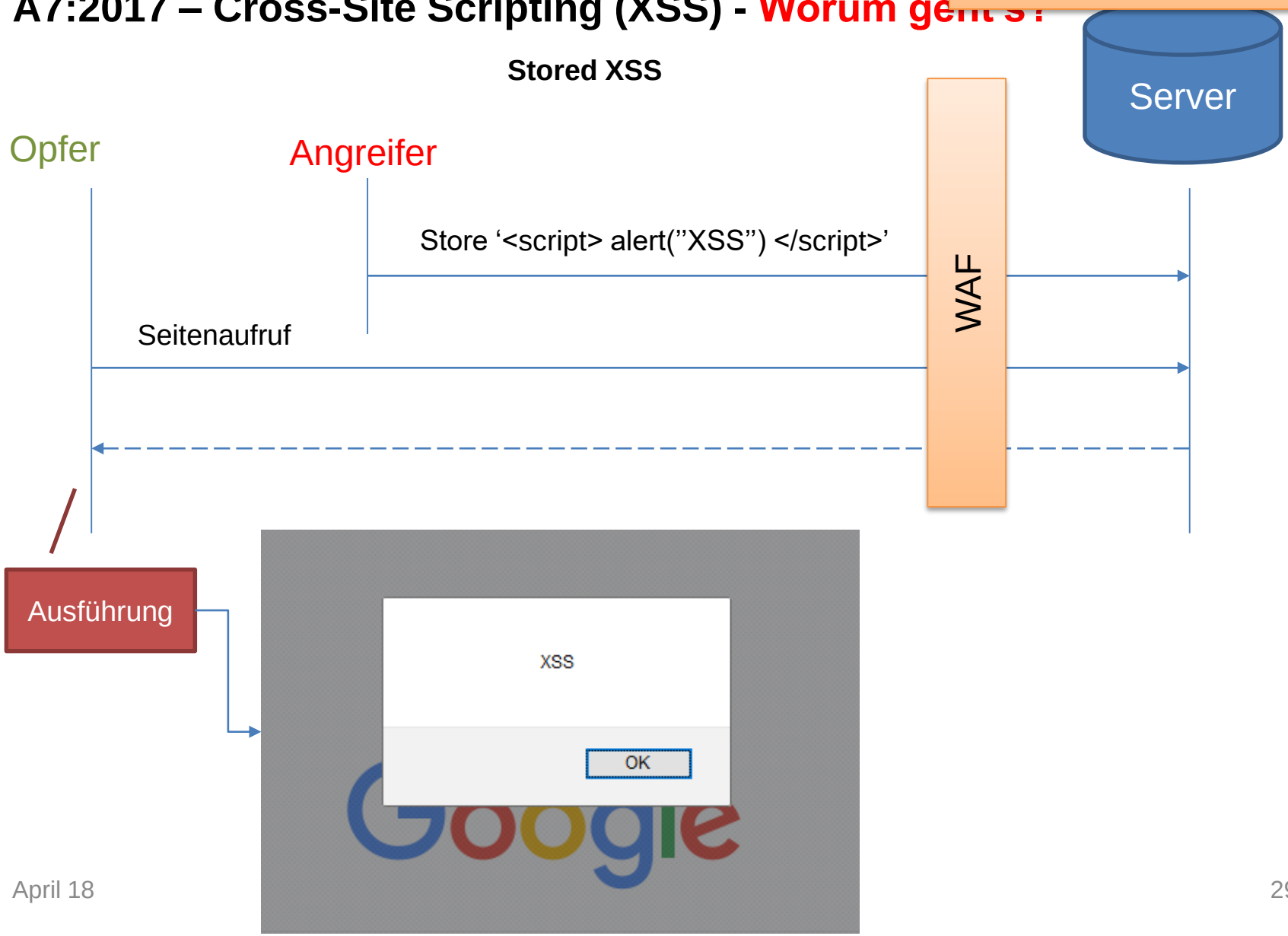
A7:2017 – Cross-Site Scripting (XSS) - Worum geht's?



- Drei Arten XSS:
  - Reflektierte XSS
  - Persistente XSS
  - DOM-basierte XSS

Ein guter WAF kann heikle Pattern erkennen

A7:2017 – Cross-Site Scripting (XSS) - Worum geht's?



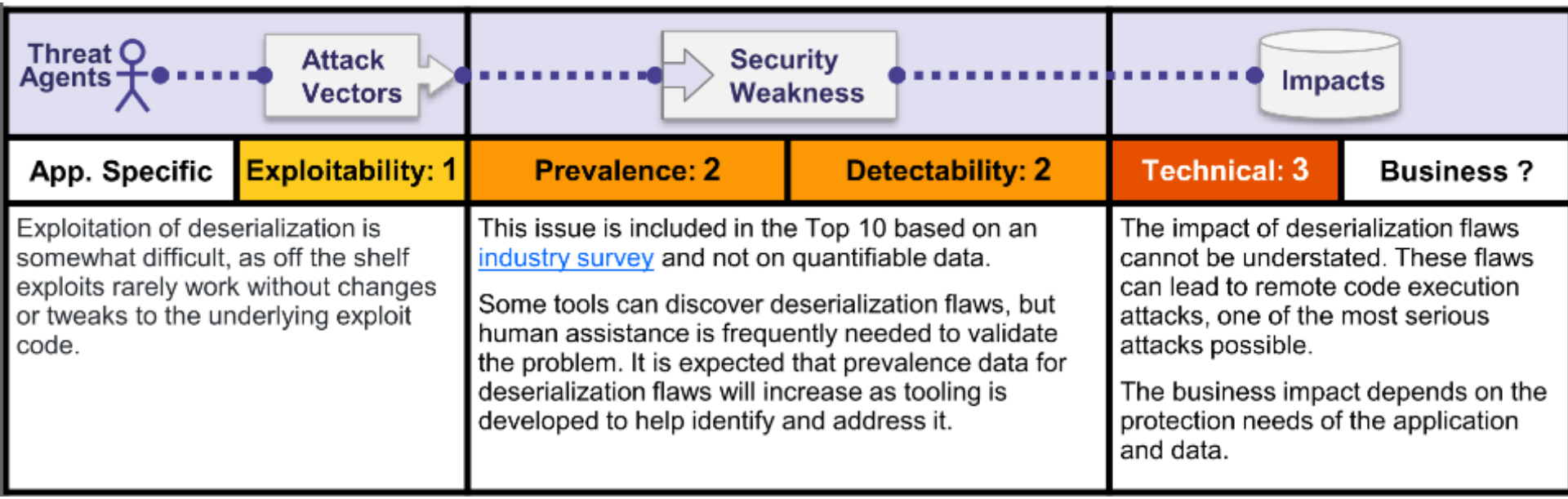
A7:2017 – Cross-Site Scripting (XSS) - Wie finden?

- Automatische Tools (z.B. Burp)
- Code-Analyse

A7:2017 – Cross-Site Scripting (XSS) - Was dagegen tun?

- Benutzereingaben prüfen (nach Möglichkeit)
- Ausgabe maskieren (immer)
- Content Security Policy (CSP) aktivieren
- Benutzung von Framework mit automatischem XSS-Schutz, z.B.:
  - Ruby on Rails
  - React JS

A8:2017 – Insecure Deserialization - Worum geht's?



A8:2017 – Insecure Deserialization - Wie finden?

- Scanner (mit Einschränkungen)
- Deserialisierung von ungeprüften Daten bedeutet Anfälligkeit

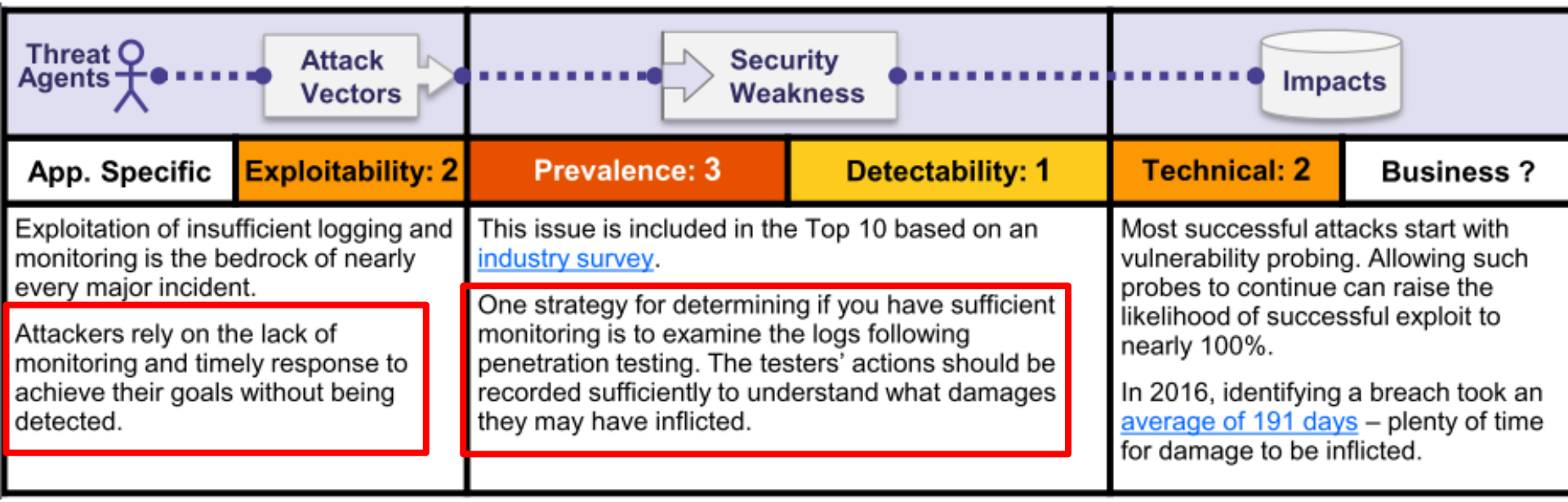
A8:2017 – Insecure Deserialization - Was dagegen tun?

- Unrealistisch:
  - Keine serialisierten Daten aus unbekannten Quellen akzeptieren
- Realistisch:
  - Integritätsprüfungen von serialisierten Objekten (digitale Signatur)
  - Enforce Type Constraints
  - Deserialisierung in isolierte Umgebung mit wenig Rechten
  - Logging und Monitoring

A9:2017 – Using Components with Known Vulnerabilities

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.	

A10:2017 – Insufficient Logging & Monitoring - Worum geht's?



- Fehlgeschlagene Logins nicht geloggt.
  → Sind aber häufig interessant

A10:2017 – Insufficient Logging & Monitoring - Wie finden?

- Prüfen der Logs nach einem Penetrationstest
  - Sind die Aktionen in den Log-Dateien abgebildet?
  - Werden sinnvolle Alarme ausgelöst?
- Werden Logins / gescheiterte Logins geloggt?
- Werden Transaktionen geloggt?
- Warnungen und Fehler generieren keine / unklare Log-Einträge
- Logs von API und Applikation werden nicht überwacht
- Log-Dateien werden nur lokal gespeichert
- Fehlende / inadäquate Alarmierung
- Aktive Attacken werden nicht zeitnah detektiert und alarmiert

A10:2017 – Insufficient Logging & Monitoring - Was dagegen tun?

- Festlegen, welche Events geloggt werden müssen (nicht abschliessend):
  - Authentisierungs-/Autorisierungsversuche (erfolgreich und fehlgeschlagen)
  - Fehler (Input- / Output-Validierung, Session Management)
  - Aktivitäten auf hoch privilegierten Funktionen (z.B. von Admins)
- Log-Format zur zentralen Überwachung definieren und einhalten
- Audit-Trail sicherstellen
- Effektive Überwachung und Alarmierung
- Pläne für den Ereignisfall
- Wiederherstellungsplan

What's Next for Developers

- OWASP Top 10 **nicht** abschliessend
- Was ist Sicherheit für «meine» Applikation?
- Security by Design
- Standard-Kontrollmechanismen[1]
- Sichere Entwicklungsstrategie (Secure Development Lifecycle)
- Verwendung von entsprechenden Frameworks
- Üben → heute: WebGoat

  «HackingLab» als Alternative (https://www.hacking-lab.com)
- Achtung: gemäss Art. 143bis Abs. 1 StGB[2] ist das «unbefugte Eindringen in ein Datenverarbeitungssystem» («Hacken») strafbar und kein Kavaliersdelikt.

1. https://www.owasp.org/index.php/OWASP_Proactive_Controls
2. https://www.admin.ch/opc/de/classified-compilation/19370083/index.html#a143bis

OWASP Top 10 – Wie kann ich auf die Anfälligkeit testen?

No.	automatisch	manuell
A1 Injection	(✔)	✔ source code review
A2 Broken Authentication		✔
A3 Sensitive Data Exposure		✔
A4 XML External Entities (XXE)	✔	✔
A5 Broken Access control	(✔) fehlende Authentisierung wird erkannt	✔
A6 Security Misconfiguration	✔	✔
A7 Cross-Site Scripting (XSS)	✔	✔
A8 Insecure Deserialization	(teilweise)	✔
A9 Using Components with Known Vulnerabilities	(teilweise)	✔
A10 Insufficient Logging & Monitoring		✔

Online-Übungen

- Die WebGoat-Übungen sind auch online verfügbar.
  https://webgoat.cnlab.ch/WebGoat
  username: webgoat
  password: cnlab

Weitere Informationen

- OWASP Top 10 für Entwickler: https://www.owasp.org/index.php/Category:OWASP_Top_10_fuer_Entwickler
- OWASP Cheat Sheets: https://www.owasp.org/index.php/OWASP_Cheat_Sheet_Series
- OWASP Dependency Check (A9): https://www.owasp.org/index.php/OWASP_Dependency_Check
- OWASP Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls
- OWASP Application Security Verification Standard Project: https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project

Danke

Thomas Lüthi
thomas.luethi@cnlab.ch
+41 55 214 33 41

Stephan Verbücheln
stephan.verbuecheln@cnlab.ch
+41 55 214 33 36

Martina Minges
martina.minges@cnlab.ch
+41 55 214 33 42