Introduction to the OWASP Top Ten

•••

Kirk Jackson
RedShield
kirk@pageofwords.com
http://hack-ed.com
@kirkj

Recordings:
https://goo.gl/a2VSG2

OWASP NZ
https://www.meetup.com/OWASP-Wellington/
www.owasp.org.nz
@owaspnz

What is OWASP?

Open Web Application Security Project (OWASP) is a nonprofit foundation that works to improve the security of software.

- A website: owasp.org
- A bunch of cool tools: Zed Attack Proxy, Juice Shop, Proactive Controls, Software Assurance Maturity Model (SAMM), Application Security Verification Standard (ASVS)
- A global community of like-minded people, meetups and conferences

Who is the OWASP Foundation?

The Open Web Application Security Project (OWASP) is a nonprofit foundation that works to improve the security of software. Through community-led open source software projects, hundreds of local chapters worldwide, tens of thousands of members, and leading educational and training conferences, the OWASP Foundation is the source for developers and technologists to secure the web.

- Tools and Resources
- Community and Networking
- Education & Training

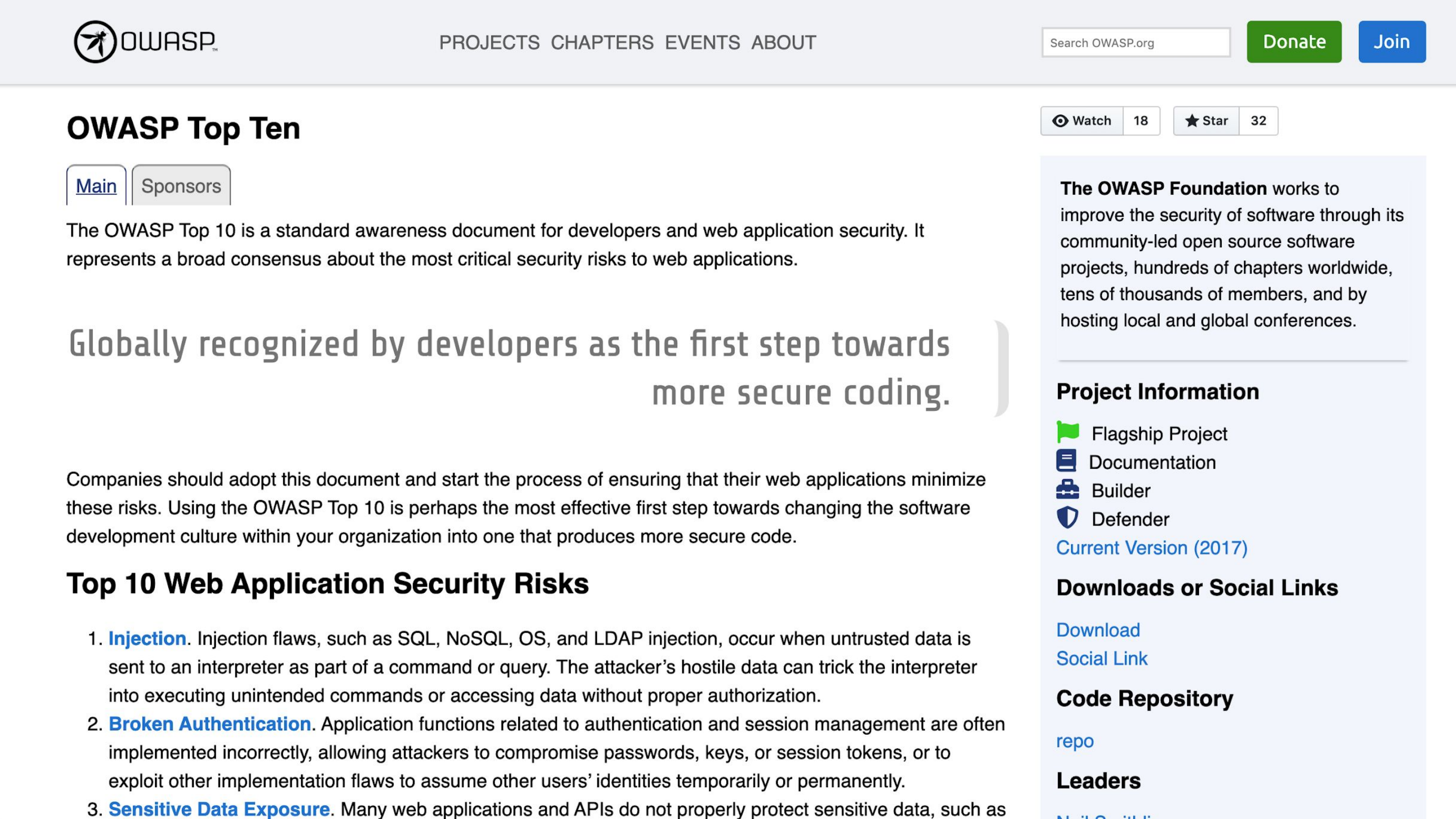
For nearly two decades corporations, foundations, developers, and volunteers have supported the OWASP Foundation and its work. Donate, Join, or become a Corporate Member today.

Project Spotlight: Zed Attack Proxy

Introducing the OWASP ZAP HUD

Simon Bennetts @psiinon
ZAP Project Leader
Mozilla Firefox Operations Security Team

Featured Chapter: Bay Area



OWASP Top Ten

Watch 18 Star 32

Main Sponsors

The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications.

> Globally recognized by developers as the first step towards more secure coding.

Companies should adopt this document and start the process of ensuring that their web applications minimize these risks. Using the OWASP Top 10 is perhaps the most effective first step towards changing the software development culture within your organization into one that produces more secure code.

Top 10 Web Application Security Risks

1. **Injection**. Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
2. **Broken Authentication**. Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.
3. **Sensitive Data Exposure**. Many web applications and APIs do not properly protect sensitive data, such as

The OWASP Foundation works to improve the security of software through its community-led open source software projects, hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

Project Information

Flagship Project
Documentation
Builder
Defender
Current Version (2017)

Downloads or Social Links

Download
Social Link

Code Repository

repo

Leaders

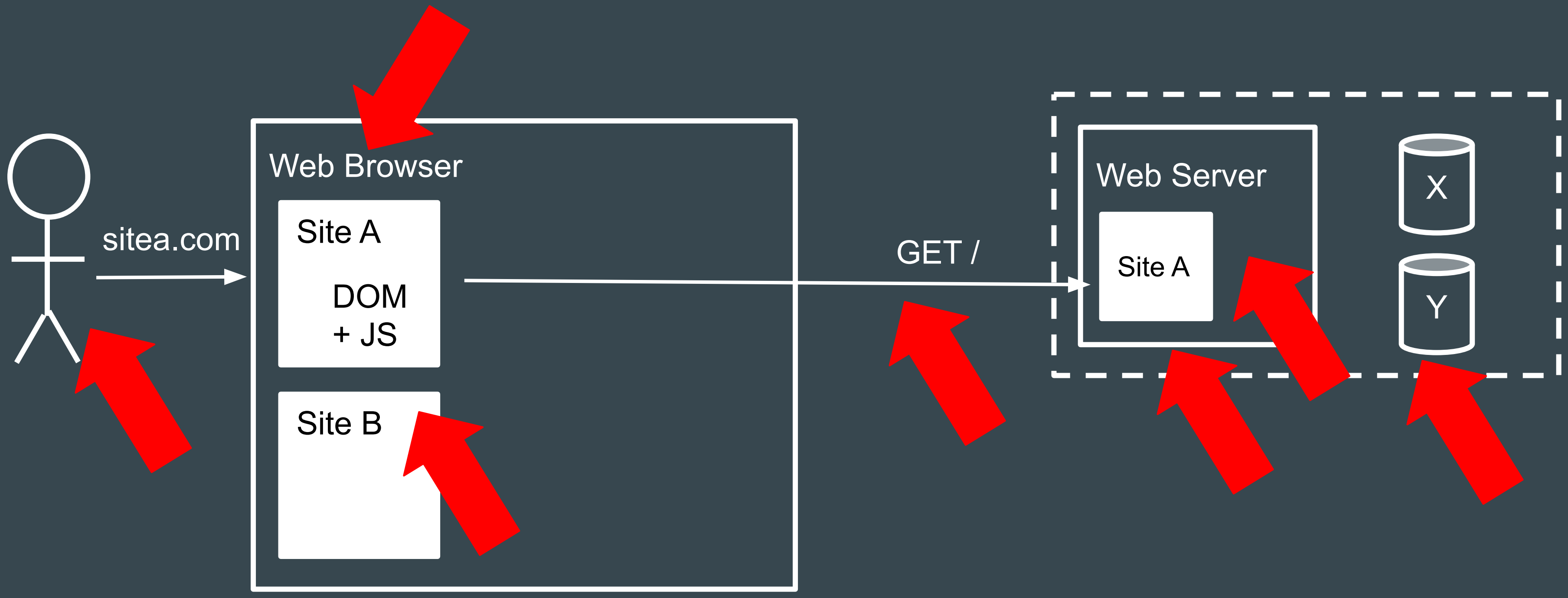
OWASP Top Ten

Globally recognized by developers as the first step towards more secure coding.

The *most critical* security risks to web applications.

Updated every 2-3 years from 2003 to 2017
(2020 is in progress)

Securing the user

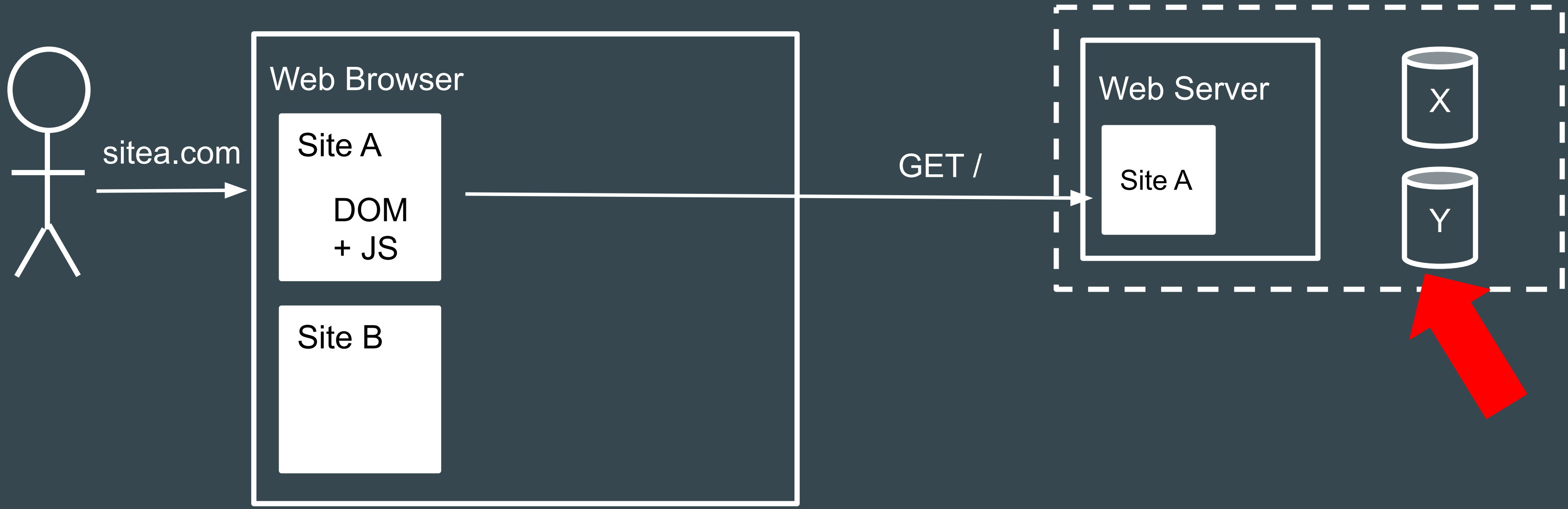


OWASP Top Ten 2017

A1	Injection
A2	Broken Authentication
A3	Sensitive Data Exposure
A4	XML External Entities (XXE)
A5	Broken Access Control
A6	Security Misconfiguration
A7	Cross-Site Scripting (XSS)
A8	Insecure Deserialization
A9	Using Components with Known Vulnerabilities
A10	Insufficient Logging & Monitoring

A1 Injection

Sending hostile data to an interpreter (e.g. SQL, LDAP, command line)



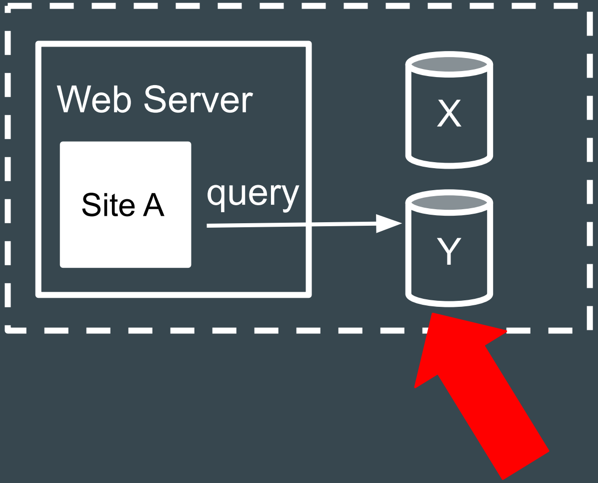
A1 Injection

Sending hostile data to an interpreter (e.g. SQL, LDAP, command line)

```
String query = "SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") + "'";

id = " '; drop table accounts -- "
```

SQL statements combine *code* and *data*



SQLi Demo

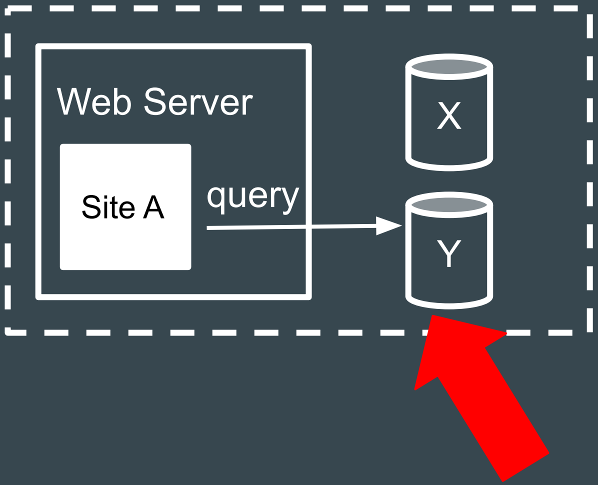
A1 Injection

Prevention:

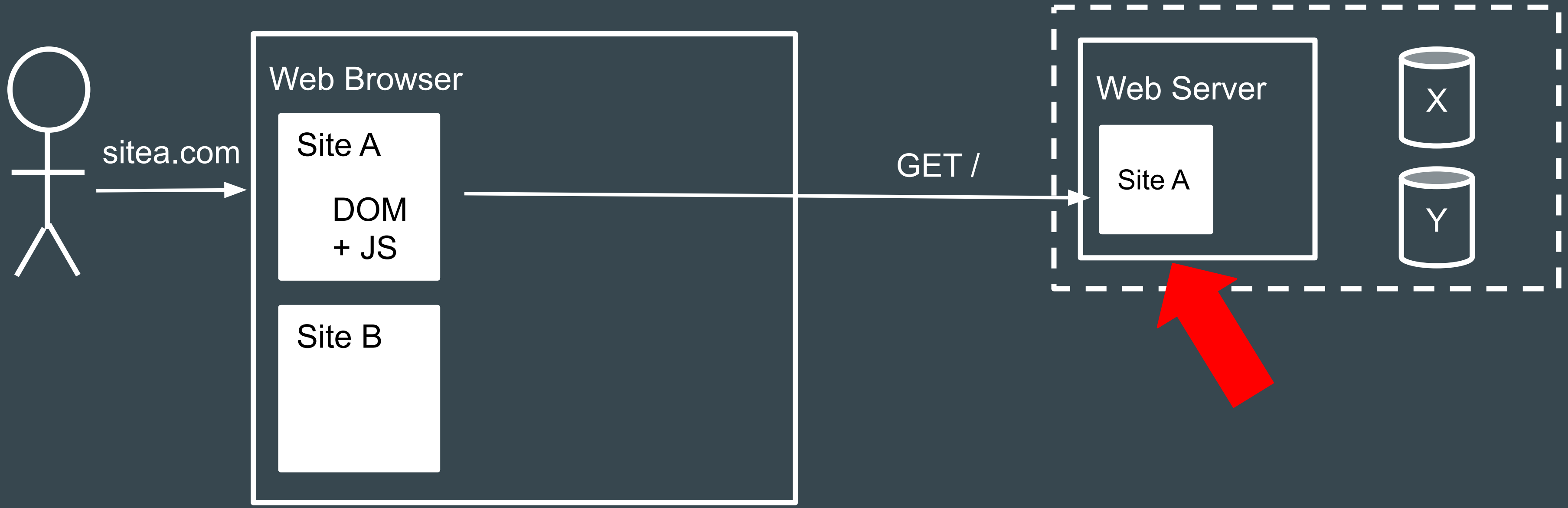
SQL statements combine *code* and *data*

=> Separate code and data

- Parameterise your queries
- Validate which data can be entered
- Escape special characters

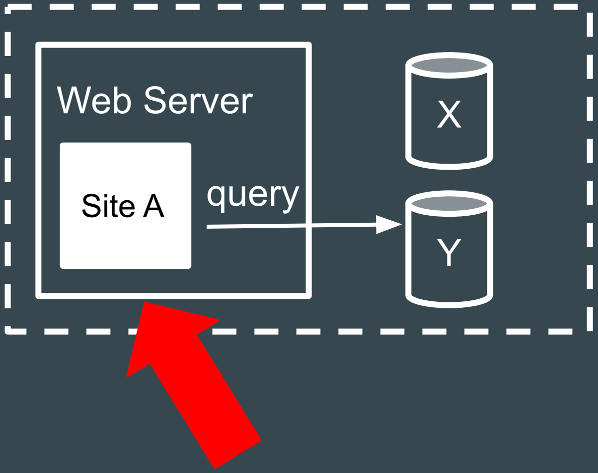


A2 Broken Authentication



A2 Broken Authentication

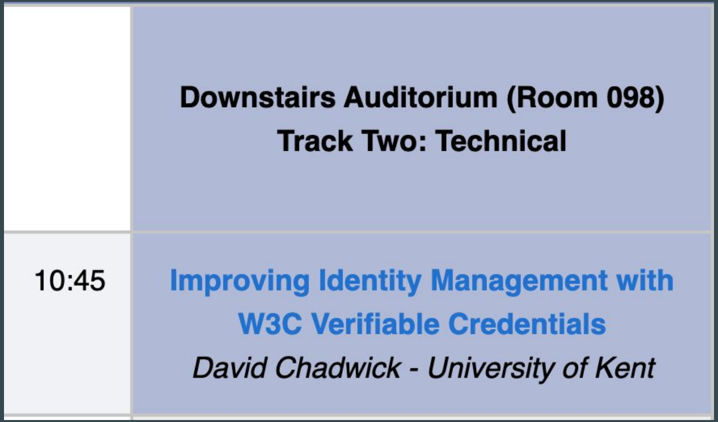
- Weak session management
- Credential stuffing
- Brute force
- Forgotten password
- No multi-factor authentication
- Sessions don’t expire



A2 Broken Authentication

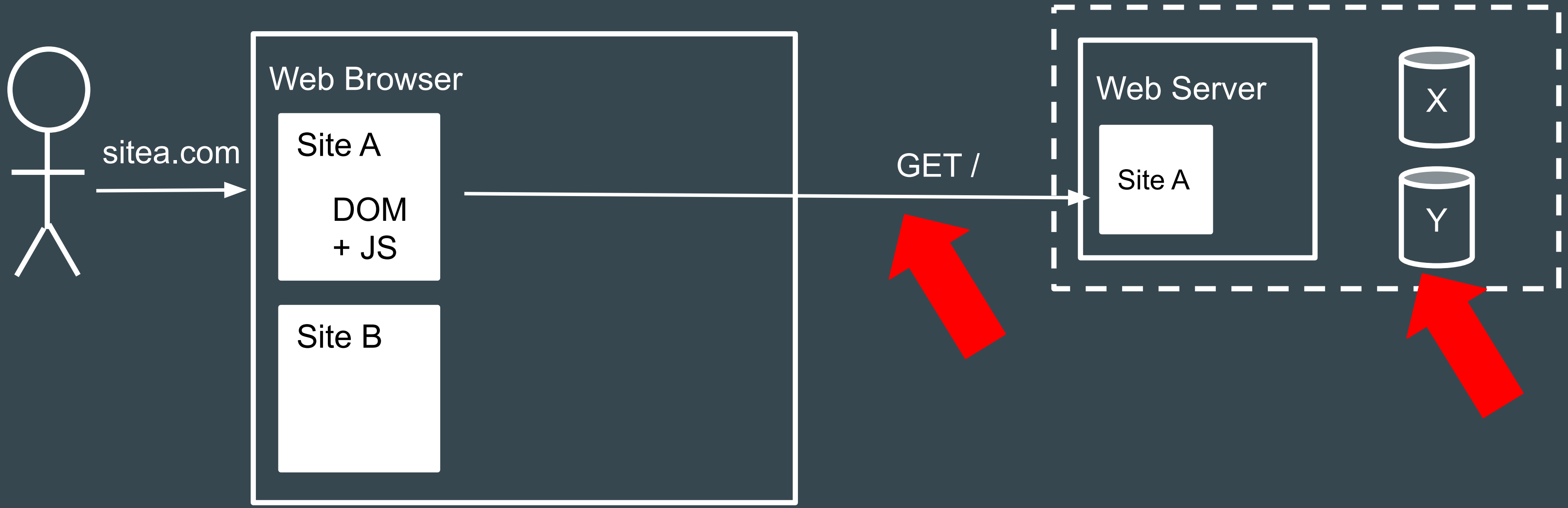
Prevention:

- Use good authentication libraries
- Use MFA
- Enforce strong passwords
- Detect and prevent brute force or stuffing attacks



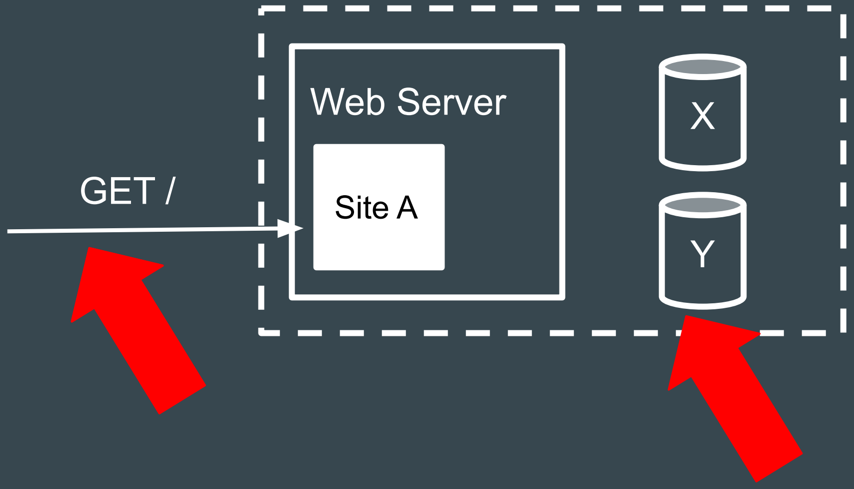
	Downstairs Auditorium (Room 098) Track Two: Technical
10:45	Improving Identity Management with W3C Verifiable Credentials *David Chadwick - University of Kent*

A3 Sensitive Data Exposure



A3 Sensitive Data Exposure

- Clear-text data transfer
- Unencrypted storage
- Weak crypto or keys
- Certificates not validated
- Exposing PII or Credit Cards

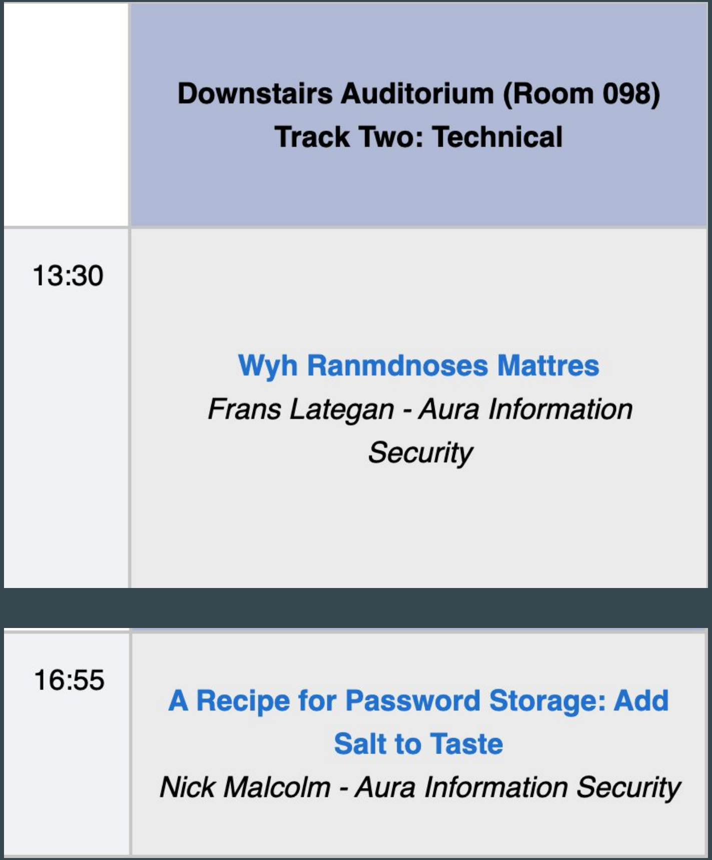


Data Exposure Demo

A3 Sensitive Data Exposure

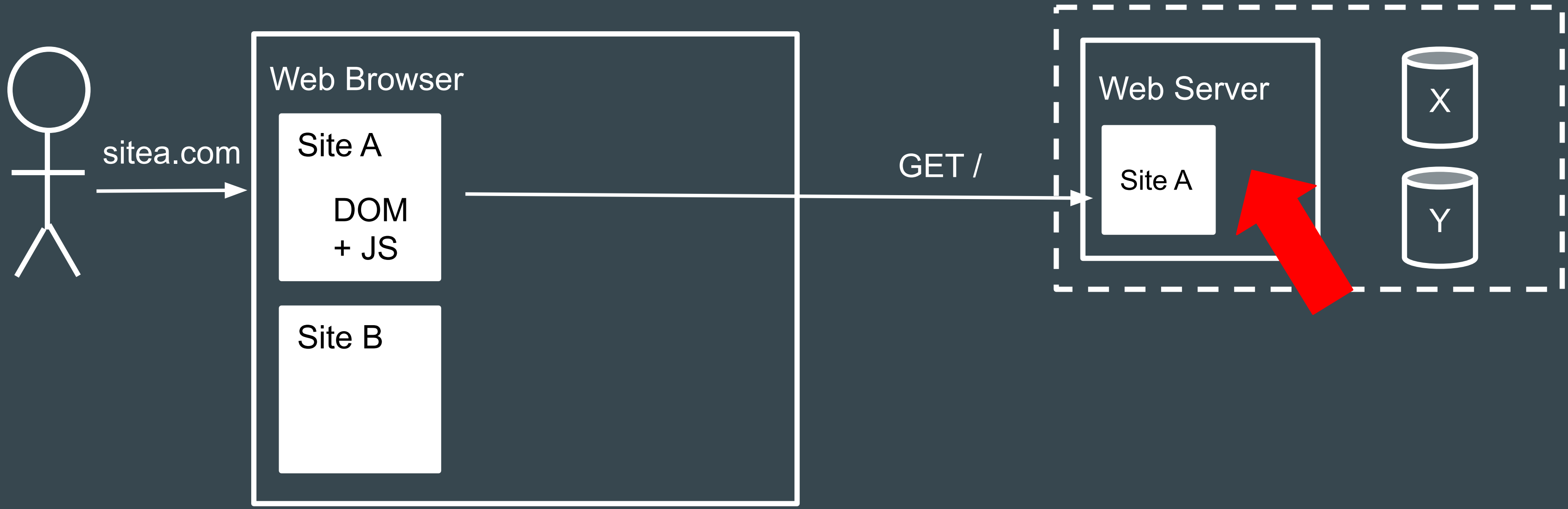
Prevention:

- Don't store data unless you need to!
- Encrypt at rest and in transit
- Use strong crypto



	Downstairs Auditorium (Room 098) **Track Two: Technical**
13:30	**Wyh Ranmdnoses Mattres** *Frans Lategan - Aura Information Security*
16:55	**A Recipe for Password Storage: Add Salt to Taste** *Nick Malcolm - Aura Information Security*

A4 XML External Entities (XXE)

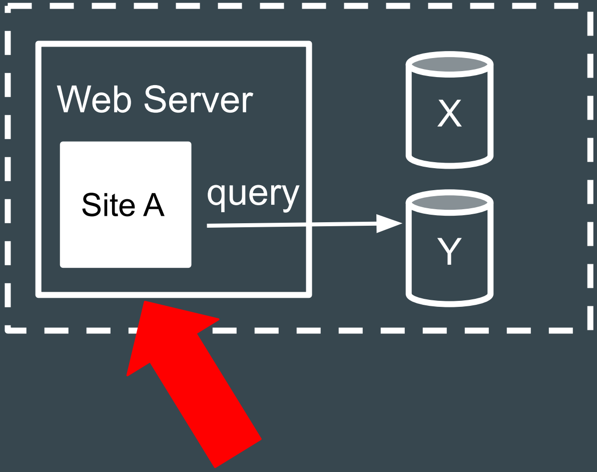


A4 XML External Entities (XXE)

The application accepts XML, and assumes it is safe

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

Can allow accessing sensitive resources, command execution, recon, or cause denial of service.

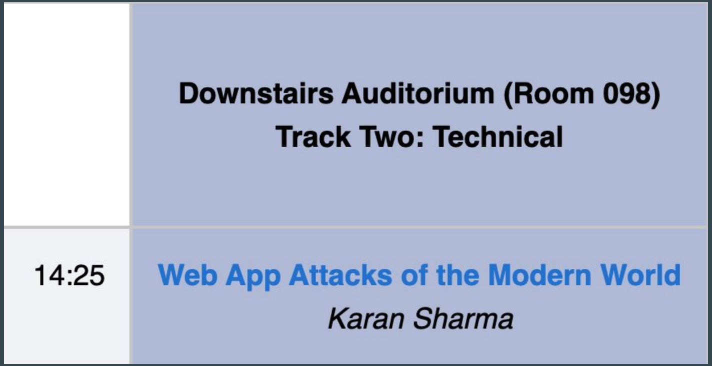


XXE Demo

A4 XML External Entities (XXE)

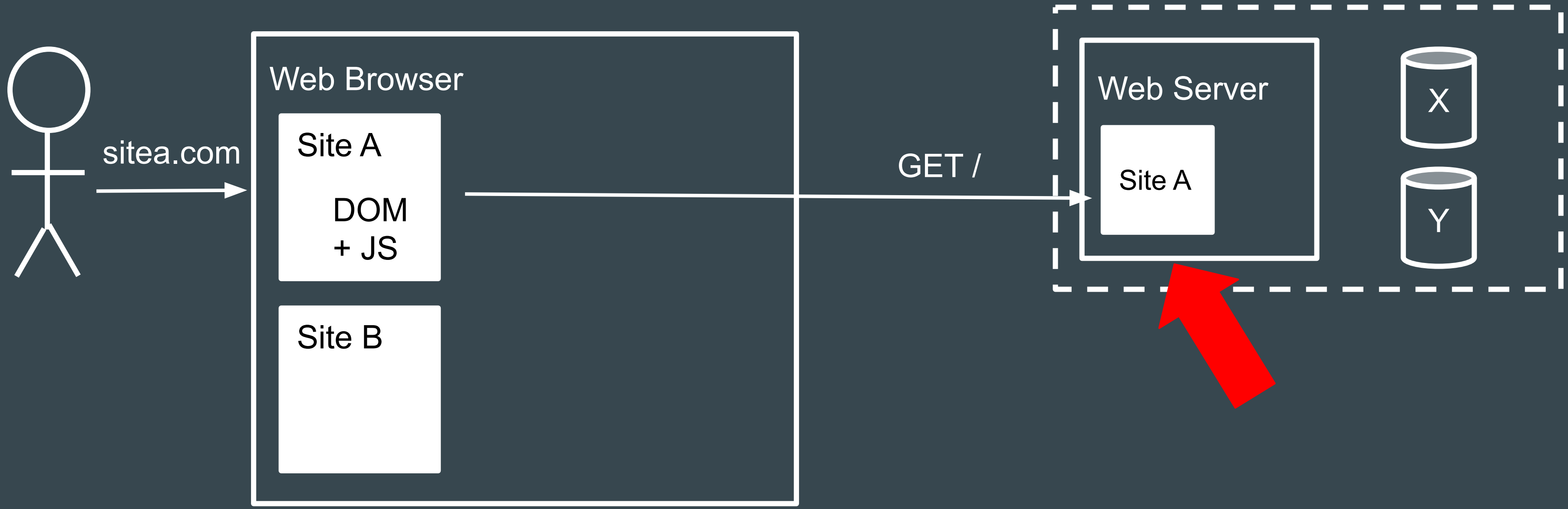
Prevention:

- Avoid XML
- Use modern libraries, and configure them well!
- Validate XML



	Downstairs Auditorium (Room 098) Track Two: Technical
14:25	Web App Attacks of the Modern World *Karan Sharma*

A5 Broken Access Control

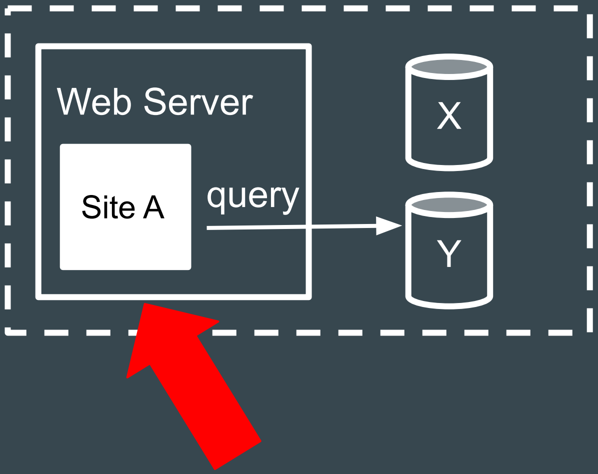


A5 Broken Access Control

- Access hidden pages

  `http://site.com/admin/user-management`
- Elevate to an administrative account
- View other people’s data

  `http://site.com/user?id=7`
- Modifying cookies or JWT tokens

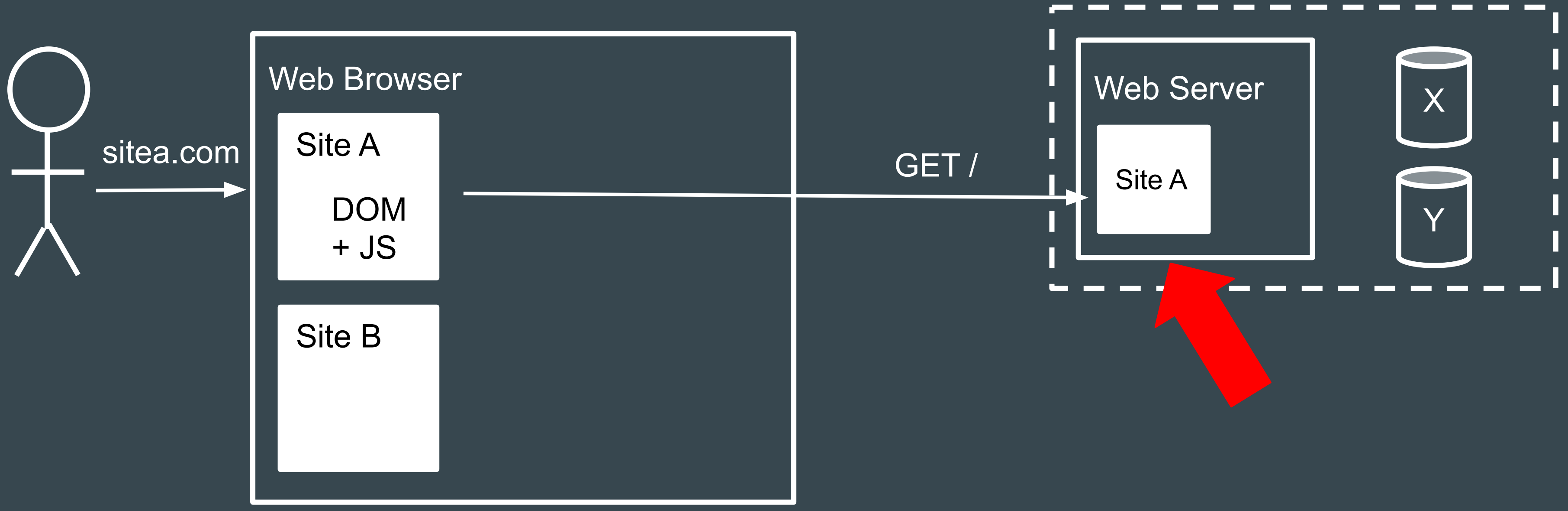


A5 Broken Access Control

Prevention:

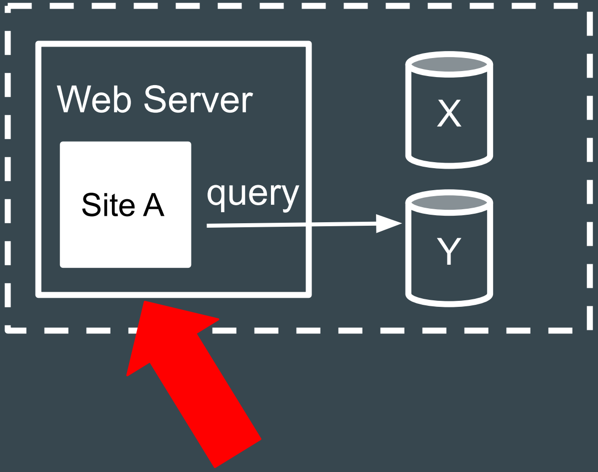
- Use proven code or libraries
- Deny access by default
- Log failures and alert
- Rate limit access to resources

A6 Security Misconfiguration



A6 Security Misconfiguration

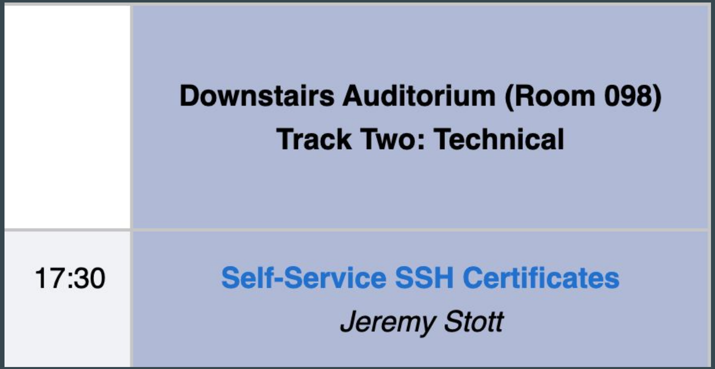
- Security features not configured properly
- Unnecessary features enabled
- Default accounts not removed
- Error messages expose sensitive information



A6 Security Misconfiguration

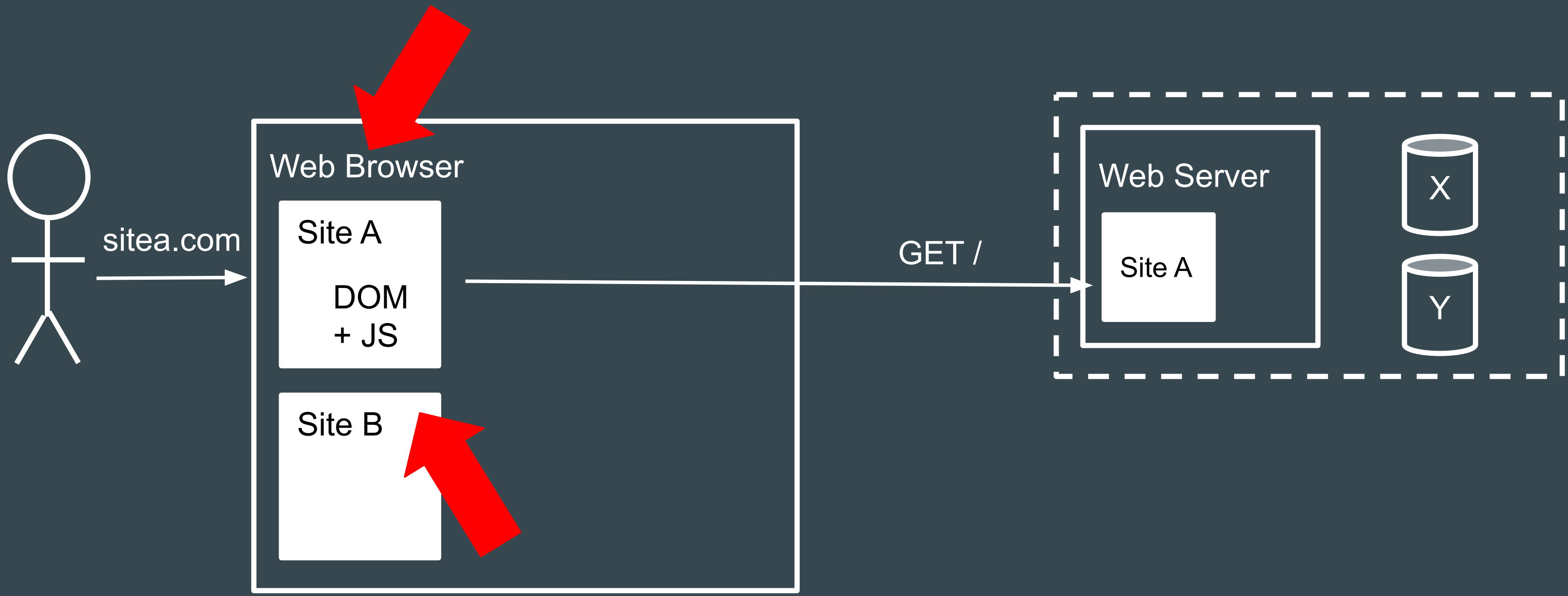
Prevention:

- Have a repeatable build process or “gold master”
- Disable all unused services
- Use tools to review settings



	Downstairs Auditorium (Room 098) Track Two: Technical
17:30	Self-Service SSH Certificates *Jeremy Stott*

A7 Cross-Site Scripting (XSS)

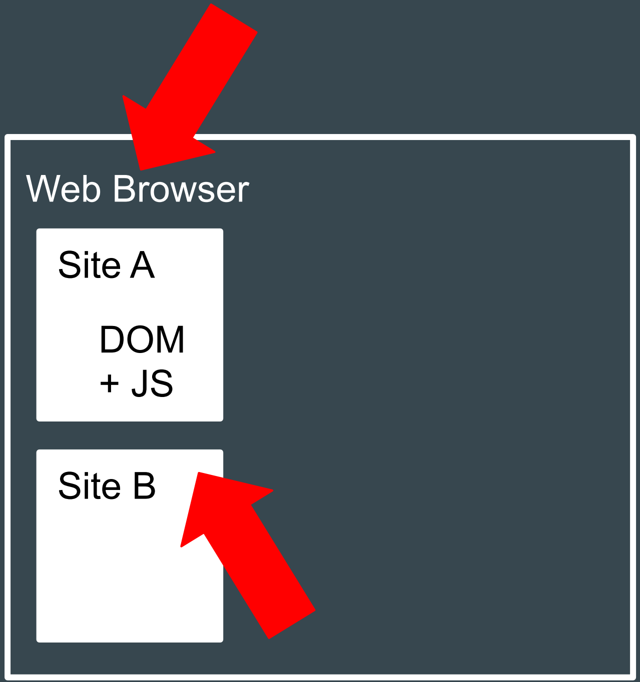


A7 Cross-Site Scripting (XSS)

HTML mixes content, presentation and code into one string (HTML+CSS+JS)

If an attacker can alter the DOM, they can do *anything* that the user can do.

XSS can be found using automated tools.



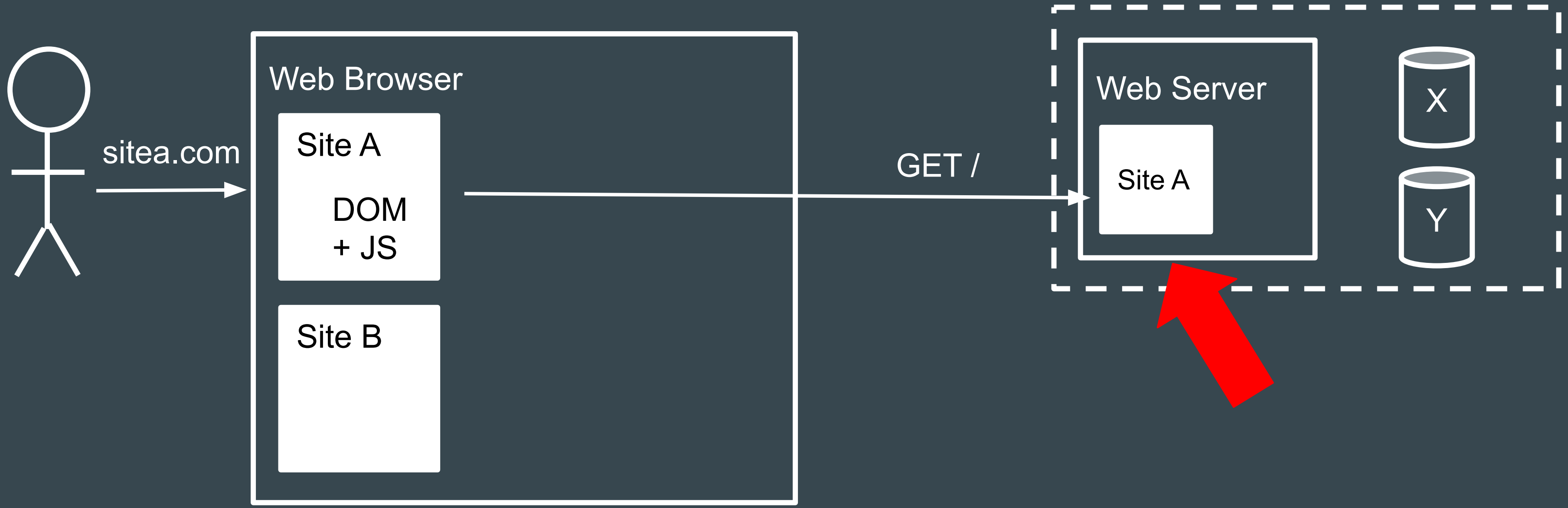
XSS Demo

A7 Cross-Site Scripting (XSS)

Prevention:

- Encode all user-supplied data to render it safe
  `Kirk <script> => Kirk &lt;script&gt;`
- Use appropriate encoding for the context
- Use templating frameworks that assemble HTML safely
- Use Content Security Policy

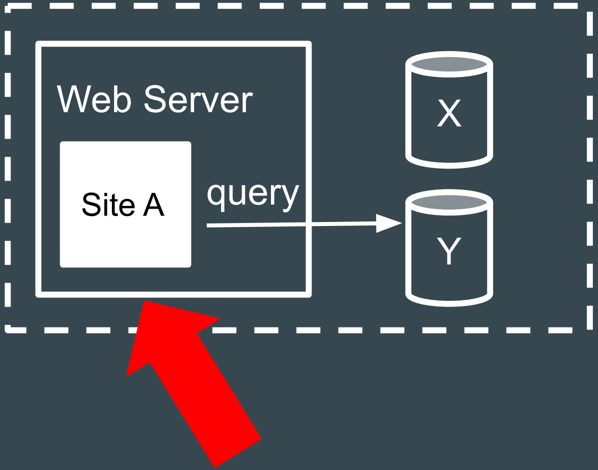
A8 Insecure Deserialization



A8 Insecure Deserialization

Programming languages allow you to turn a tree of objects into a string that can be sent to the browser.

If you *deserialise* untrusted data, you may allow objects to be created, or code to be executed.



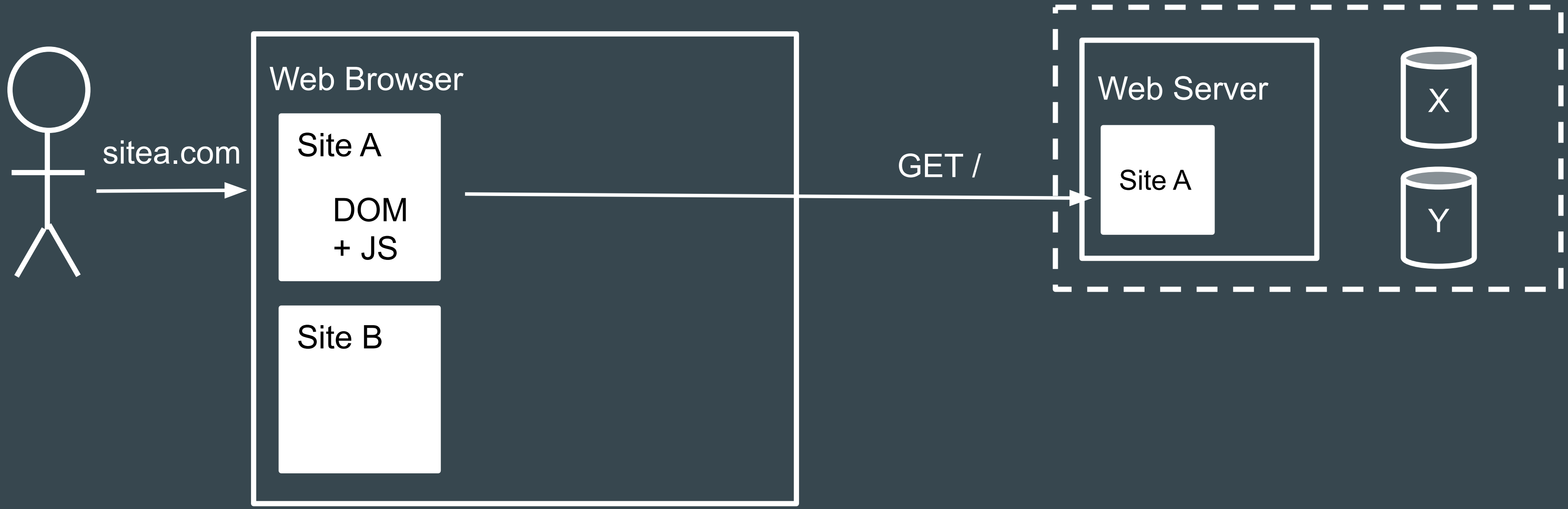
Deserialisation Demo

A8 Insecure Deserialization

Prevention:

- Avoid serialising and deserialising objects
- Use signatures to detect tampering
- Configure your library safely
- Check out the OWASP Deserialisation Cheat Sheet

A9 Using Components with Known Vulnerabilities



A9 Using Components with Known Vulnerabilities

Modern applications contain a *lot* of third-party code.

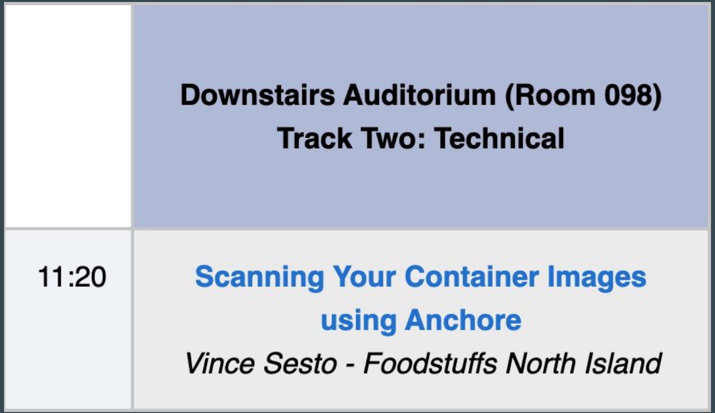
It’s hard to keep it all up to date.

Attackers can enumerate the libraries you use, and develop exploits.

A9 Using Components with Known Vulnerabilities

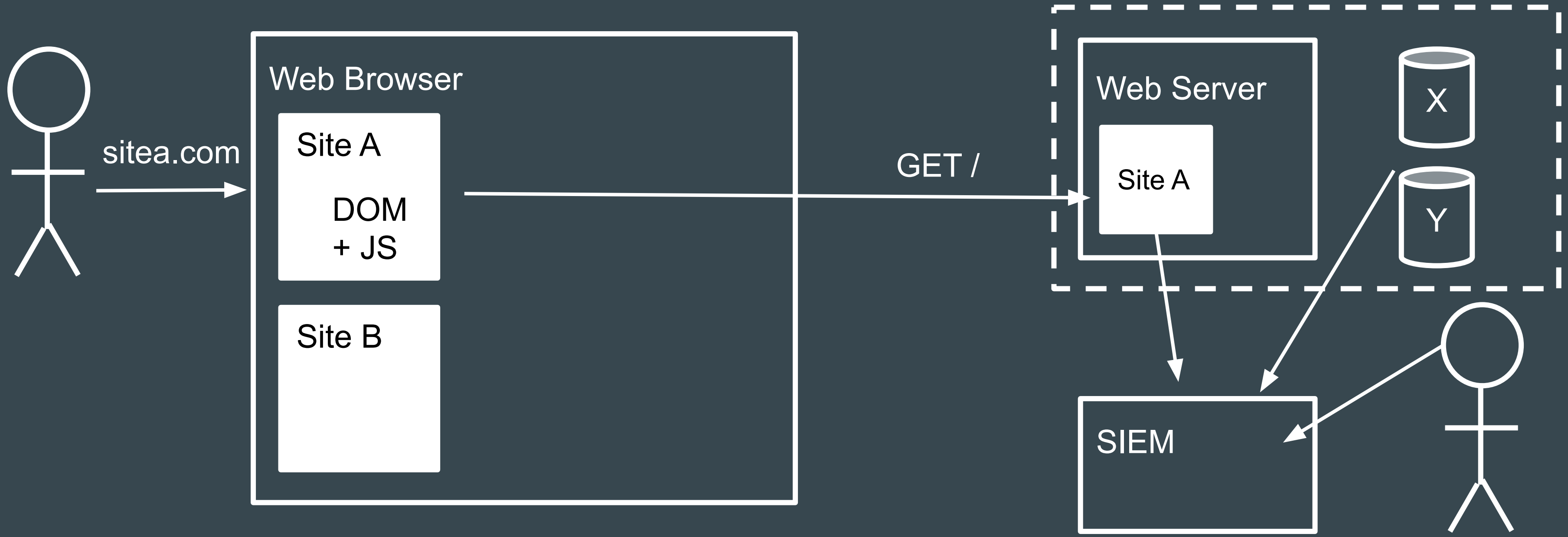
Prevention:

- Reduce dependencies
- Patch management
- Scan for out-of-date components
- Budget for ongoing maintenance for all software projects



	Downstairs Auditorium (Room 098) **Track Two: Technical**
11:20	**Scanning Your Container Images using Anchore** *Vince Sesto - Foodstuffs North Island*

A10 Insufficient Logging & Monitoring

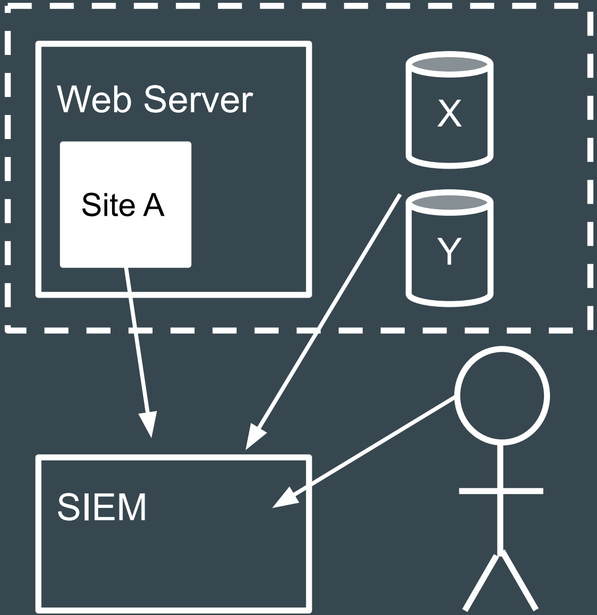


A10 Insufficient Logging & Monitoring

You can't react to attacks that you don't know about.

Logs are important for:

- Detecting incidents
- Understanding what happened
- Proving who did something



OWASP Top Ten 2017

A1	Injection
A2	Broken Authentication
A3	Sensitive Data Exposure
A4	XML External Entities (XXE)
A5	Broken Access Control
A6	Security Misconfiguration
A7	Cross-Site Scripting (XSS)
A8	Insecure Deserialization
A9	Using Components with Known Vulnerabilities
A10	Insufficient Logging & Monitoring

Next Steps

Next Steps

- Attend OWASP events
- Search for OWASP Top Ten category names and your framework
  E.g. “C# XSS protection”
- Watch youtube or Pluralsight videos
- Use the terms when discussing bugs with colleagues
- Keep track of which issues affect you the most
- Go beyond the Top Ten

Introduction to the OWASP Top Ten

•••

Kirk Jackson
RedShield
kirk@pageofwords.com
http://hack-ed.com
@kirkj

Recordings:
https://goo.gl/a2VSG2

OWASP NZ
https://www.meetup.com/OWASP-Wellington/
www.owasp.org.nz
@owaspnz