Ajilon

OWASP
The Open Web Application Security Project

OWASP Top 10 - 2013

Featuring

and

Ulf Larson

ulf.larson@ajilonconsultants.se

Oct 24, 2013

OWASP top ten 2013

- Based on risk data from eight firms that specialize in application security,
- This data spans over 500,000 vulnerabilities across hundreds of organizations and thousands of applications.
- The Top 10 items are selected and prioritized according to this prevalence data, in combination with consensus estimates of exploitability, detectability, and impact estimates.

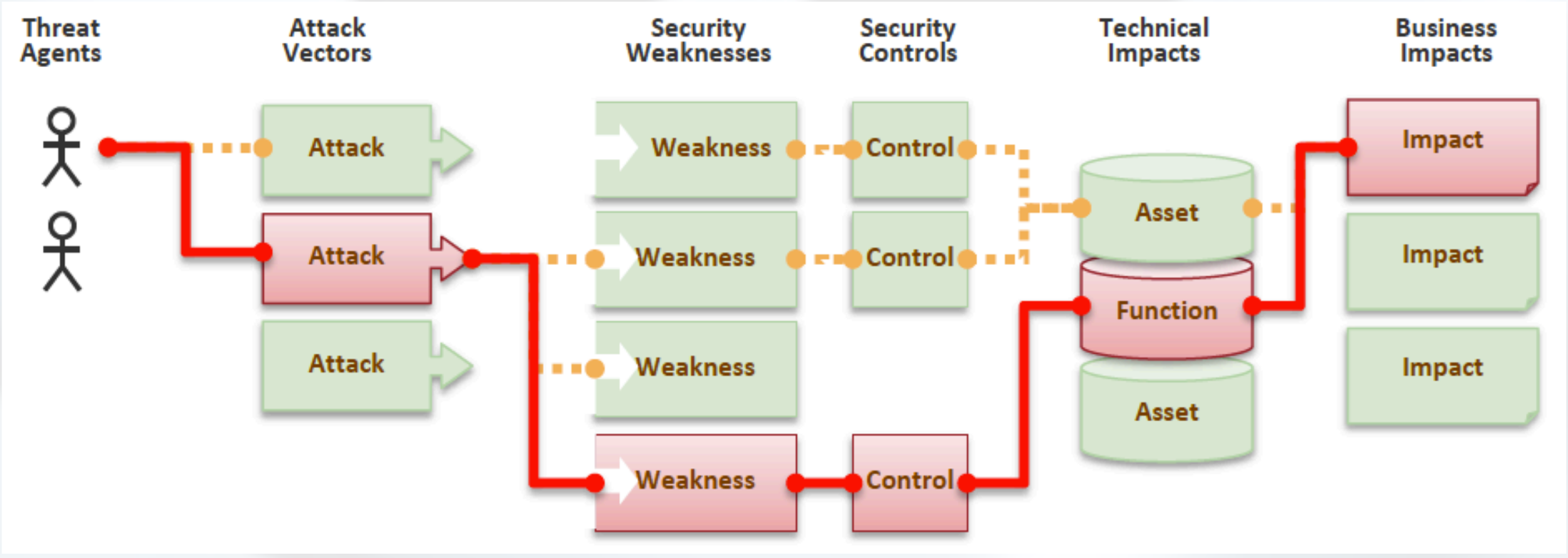
Aim

- The primary aim of the OWASP Top 10 is to educate developers, designers, architects, managers, and organizations about the consequences of the most important web application security weaknesses.

- The Top 10 provides basic techniques to protect against these high risk problem areas – and also provides guidance on where to go from here.

It's all about risk, but what *is* risk?

Each path (red) carries a risk. Some of the risks may warrant attention



So, how big is *my* risk?

Threat Agent	Attack Vector	Weakness Prevalence	Weakness Detectability	Technical Impact	Business Impact
?	Easy	Widespread	Easy	Severe	?
	Average	Common	Average	Moderate	
	Difficult	Uncommon	Difficult	Minor	

Why are the Threat Agent and Business Impact fields empty?

Before we begin. Some very basics of web communication

Before we begin. Some very basics of web communication

1. Someone has written code and put it here
(html, javascript, aspx, C#)

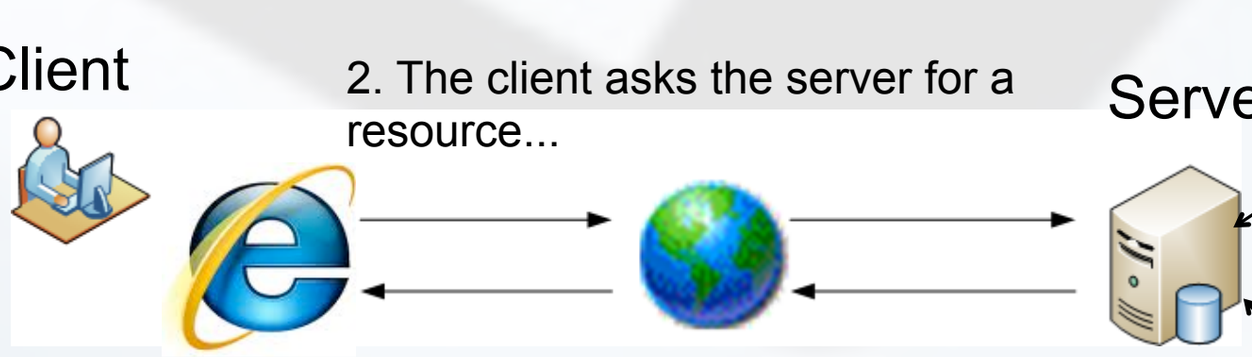
Before we begin. Some very basics of web communication

1. Someone has written code and put it here (html, javascript, aspx, C#)

Client

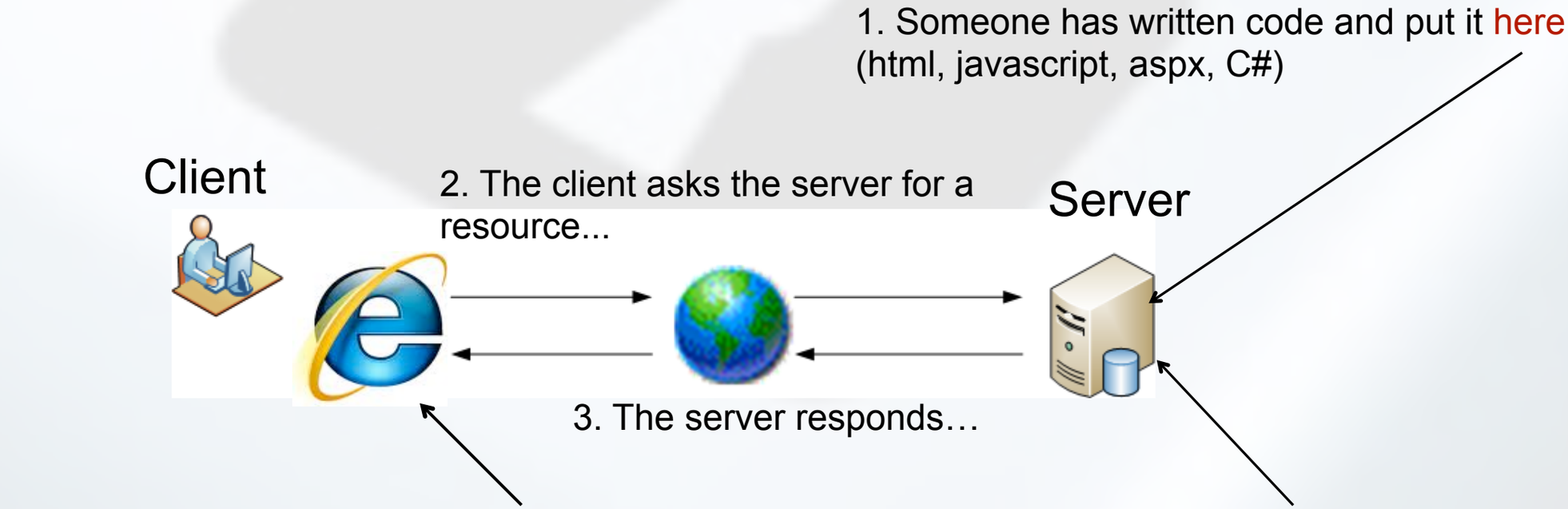
2. The client asks the server for a resource...

Server

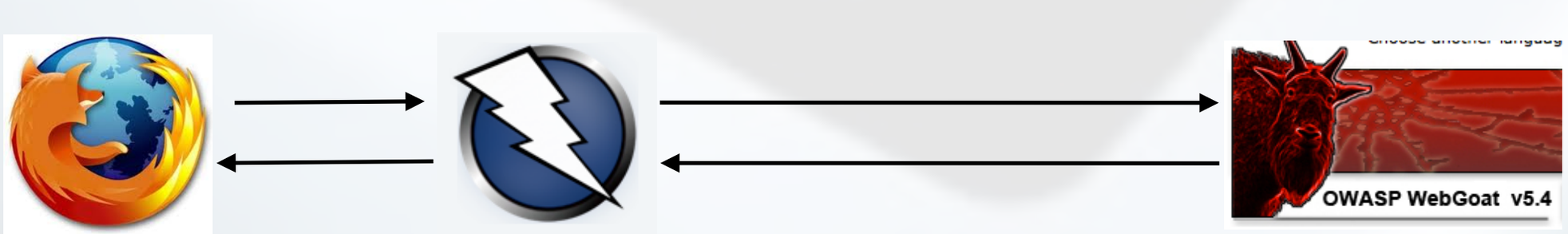


… and part of the code interprets and services the request

Before we begin. Some very basics of web communication



(The same model represented as) Demo setup



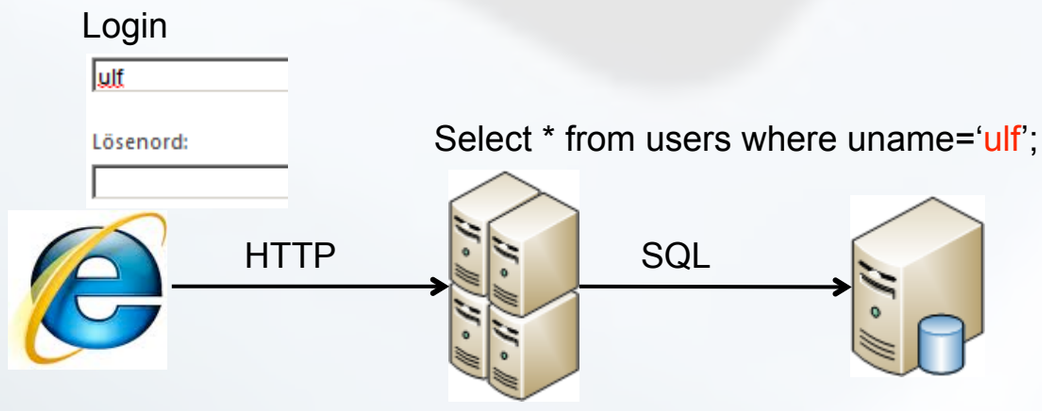
Top ten 2013

OWASP Top 10 – 2013 (New)
A1 – Injection
A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References
A5 – Security Misconfiguration
A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control
A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards

A1 Injection

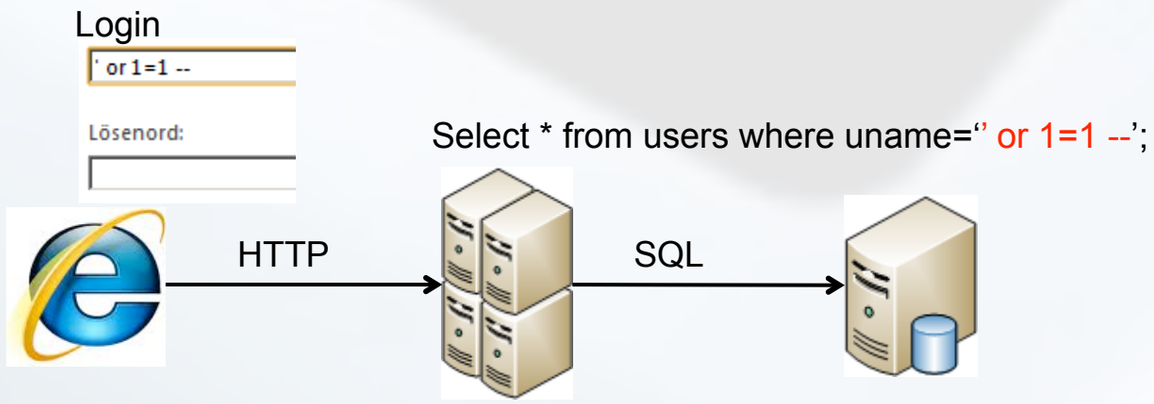
Input data from untrusted source (external system, user through web browser) is interpreted as code or part of query (SQL, Xpath), rather than as data

Exemple (intended use)

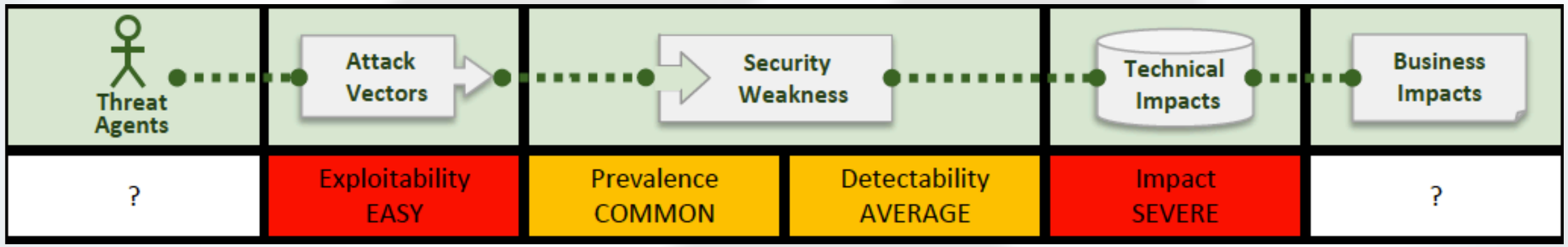


A1 Injection

Example (not-so-intended use)



A1 Injection



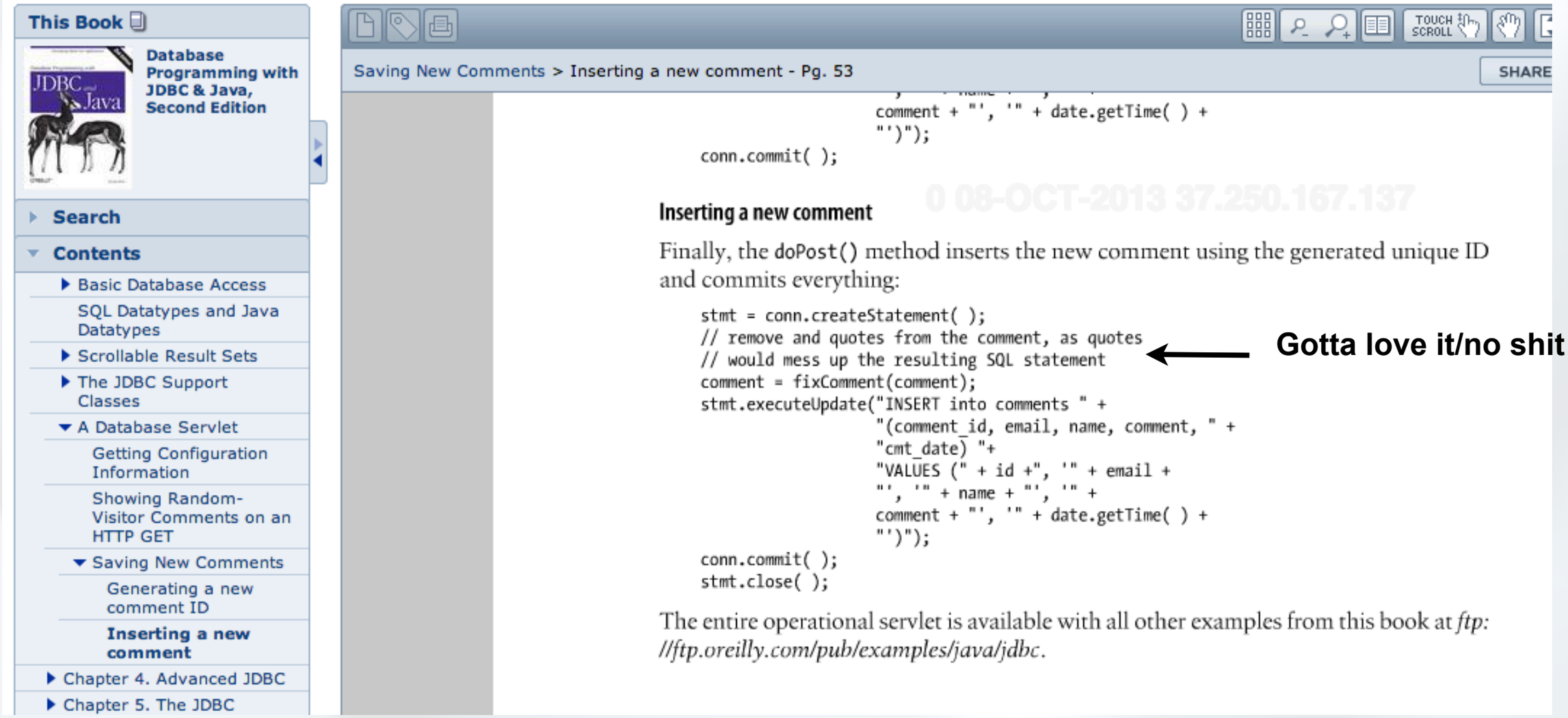
Creating text data in front of a browser is simple, i.e., `' or 1=1`

Many types, SQL, Xpath, LDAP…

Tools, error messages to screen, sad error handling in apps

Data loss, arbitrary code execution, denial of service (all users)

Wonder why people still get it wrong? Ah. Sorry. My bad...

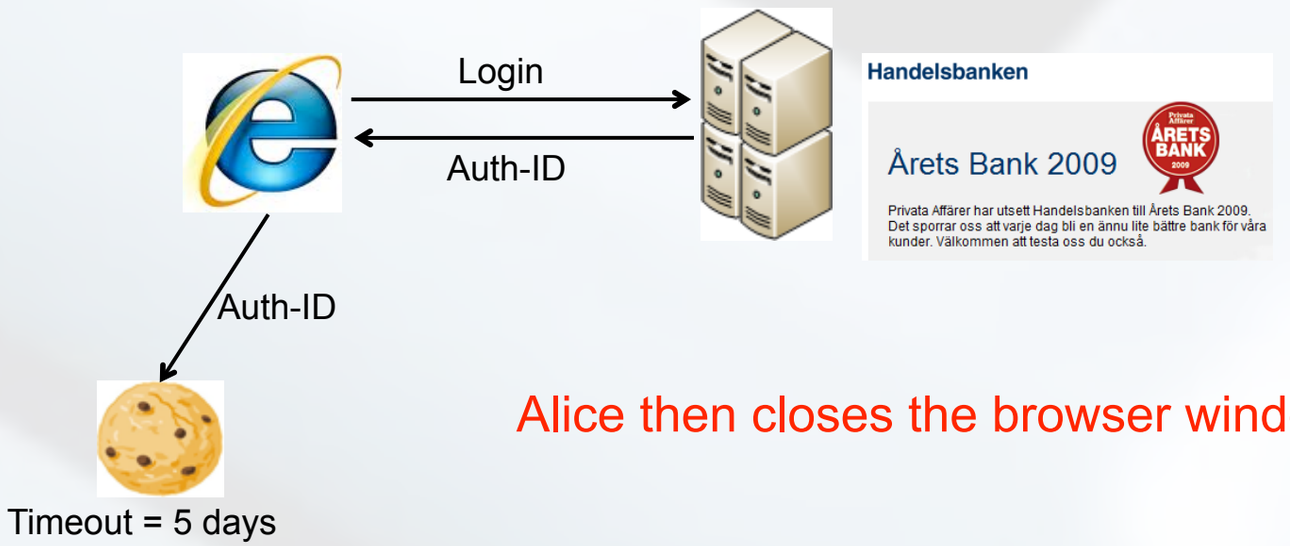


Let's see an example from Webgoat

A2 Broken Authentication and Session Management

Authentication and session handling functions are incorrectly implemented. THis leads to that the attacker can find/figure out passwords, session IDs, and thus steal the users' identities

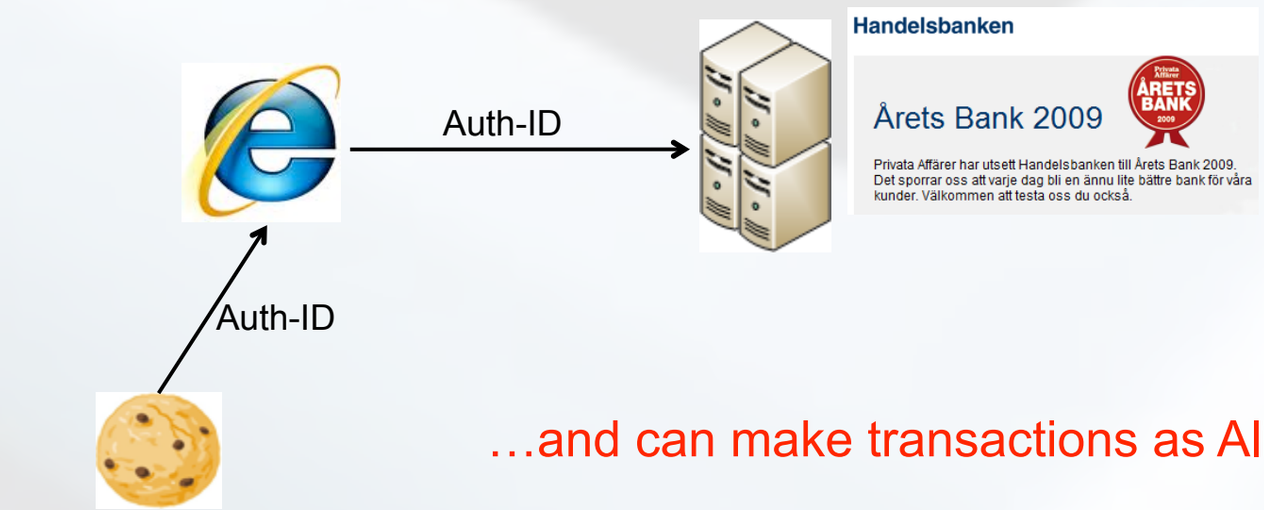
Example: Public computer, user Alice logs in to the bank



Alice then closes the browser window and walks away

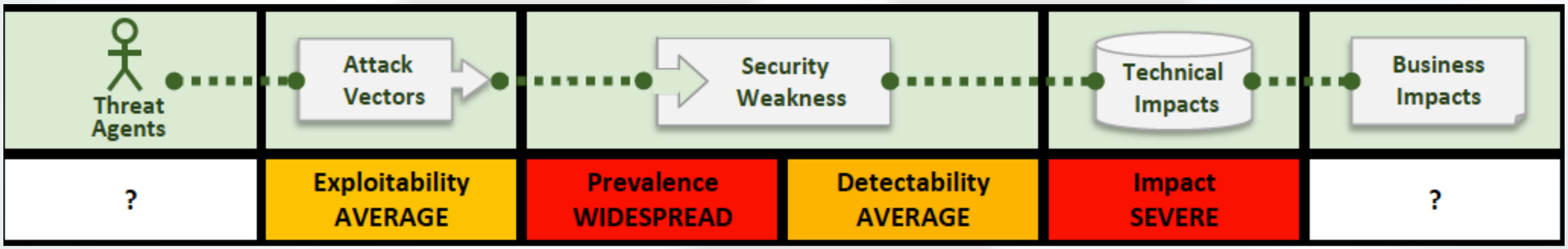
A2 Broken Authentication and Session Management

Example: Eve navigates to the bank on the same computer, a few days later...



…and can make transactions as Alice, without having to log in

A2 Broken Authentication and Session Management



Publicly accessible computers, proxy to determine randomness in ID generation

Developers tend to create their own functions for session management

Different solutions makes it difficult to find a template for what a vulnerability looks like

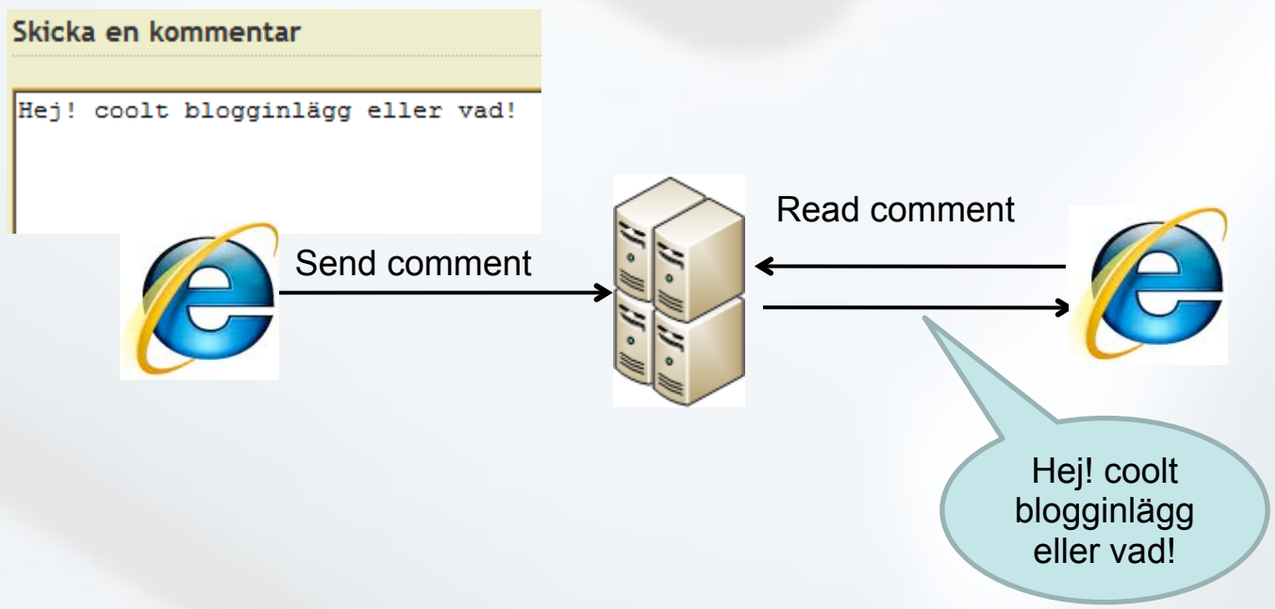
Usernames, passwords, session IDs may be stolen. Bad for single users

Let's see an example from Webgoat

A3 Cross-Site Scripting (XSS)

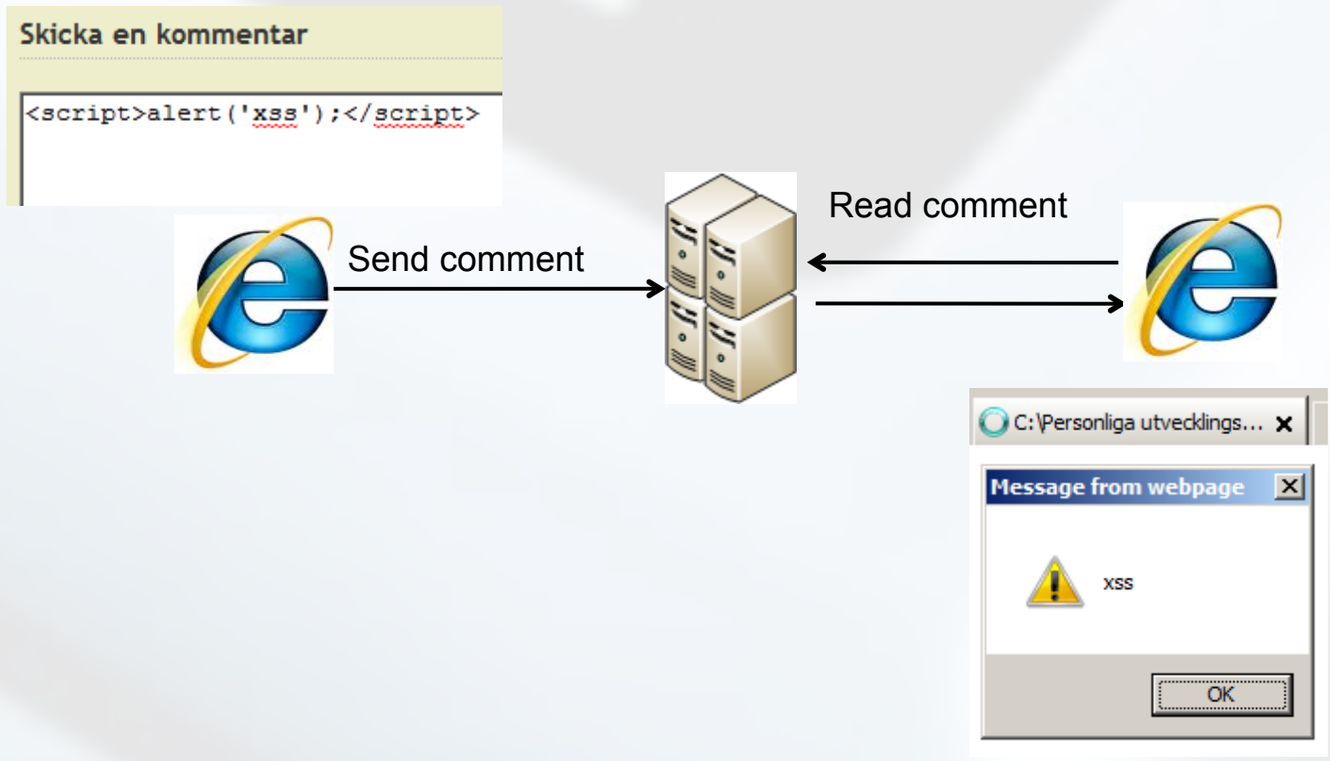
Data is sent from server to web browser without input validation or output data encoding. Sent data is then interpreted by the browser as script code

Example (intended use)

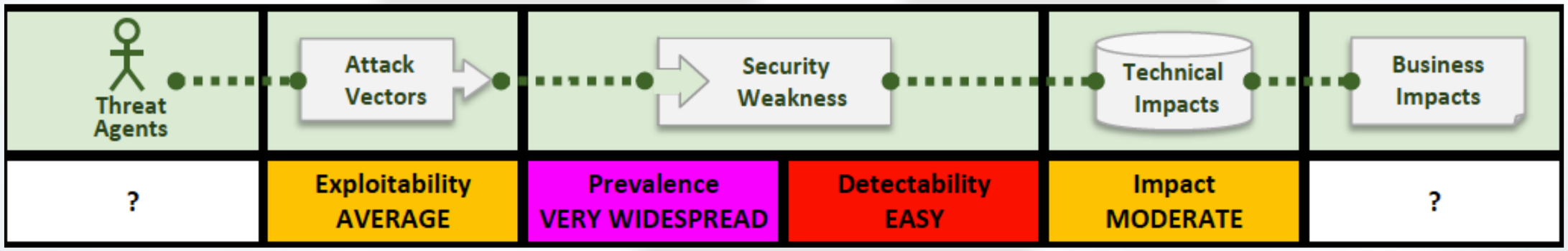


A3 Cross-Site Scripting (XSS)

Example (not-so-intended use)



A3 Cross-Site Scripting (XSS)



Could be difficult to create text data with correct syntax (<script>...)

Data is passed between systems all the time

Send data, see what happens, look at source code, fairly easy to spot tags that becomes out of place, < or < in source code

Session hijacking, traffic redirection

Ungefär 11 300 000 resultat (0,18 sekunder)

Let's see an example from Webgoat

And guys, Let's face it

- Unless we do the last seven really quick, this won't work.

A4 – Insecure Direct Object References	•A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.	
A5 – Security Misconfiguration	•Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults. This includes keeping all software up to date.	
A6 – Sensitive Data Exposure	•Many web applications do not properly protect sensitive data, such as credit cards, tax ids, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.	Was A7 and A9!
A7 – Missing Function Level Access Control	•Virtually all web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access unauthorized functionality.	Was A8
A8 - Cross-Site Request Forgery (CSRF)	•A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.	Was A4
A9 - Using Components with Known Vulnerabilities	•Vulnerable components, such as libraries, frameworks, and other software modules almost always run with full privilege. So, if exploited, they can cause serious data loss or server takeover. Applications using these vulnerable components may undermine their defenses and enable a range of possible attacks and impacts.	New!
A10 – Unvalidated Redirects and Forwards	•Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.	

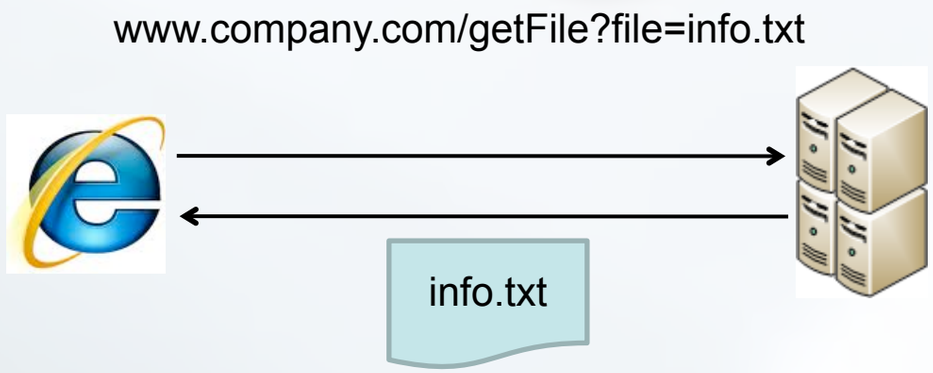
And finally...


THE BITTERNESS OF POOR QUALITY
REMAINS LONG AFTER THE SWEETNESS
OF LOW PRICE IS FORGOTTEN.

A4 Insecure Direct Object References

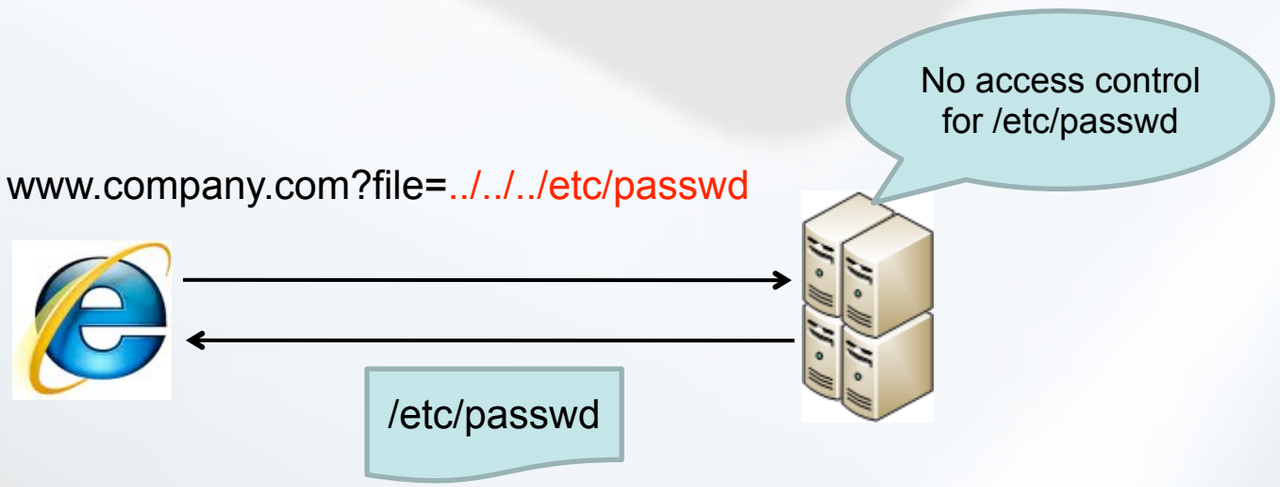
An object reference, e.g., a file, is made visible to the user. If access control is not enforced when the object is accessed, the user can try accessing other objects through the visible reference

Example (intended use)

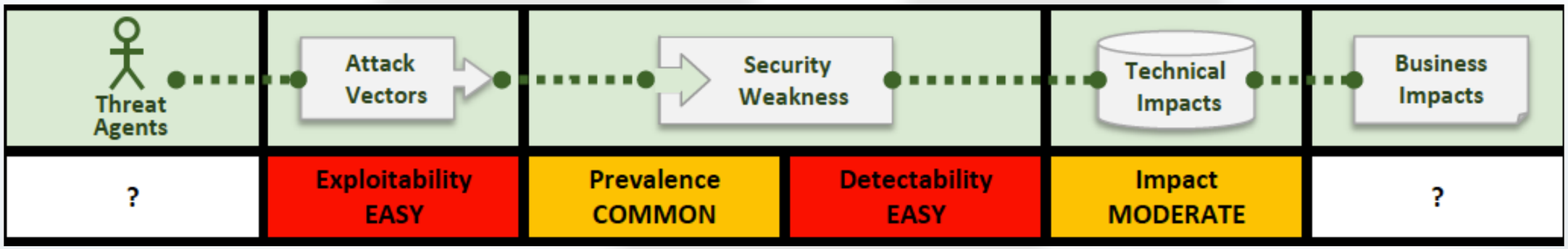


A4 Insecure Direct Object References

Example: (not-so-intended use)



A4 Insecure Direct Object References



Look in the parameter field during object access. Try to change name/ value on the denoted object

Direct object references are common

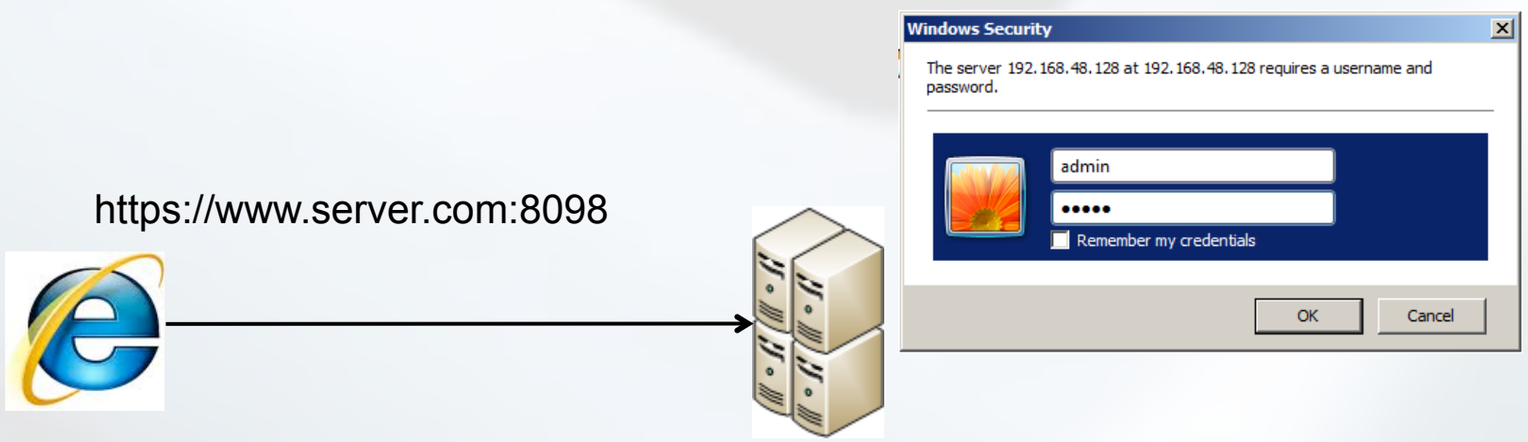
Simple to “see” that a potential vulnerability exist. Just look at the link

Mot probably you will access similar filetypes. Normally you won’t be able to escape the web server directory... but you’ll never know

A5 Security Misconfiguration

Some or several components (application, framework, web server, application server...) in a system is not correctly configured. An attacker uses some or many of these configuration mistakes

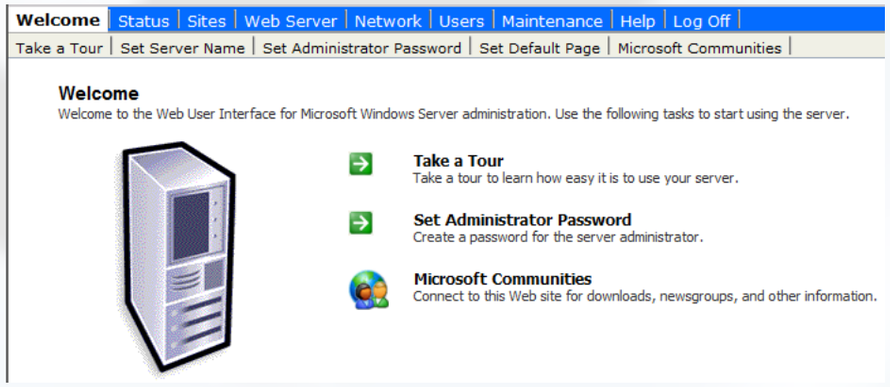
Example: IIS remote administration application running on port 8098 (default)

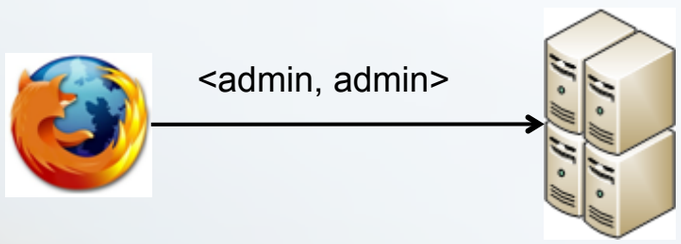


<admin:admin> perhaps. Will that work?

A5 Security Misconfiguration

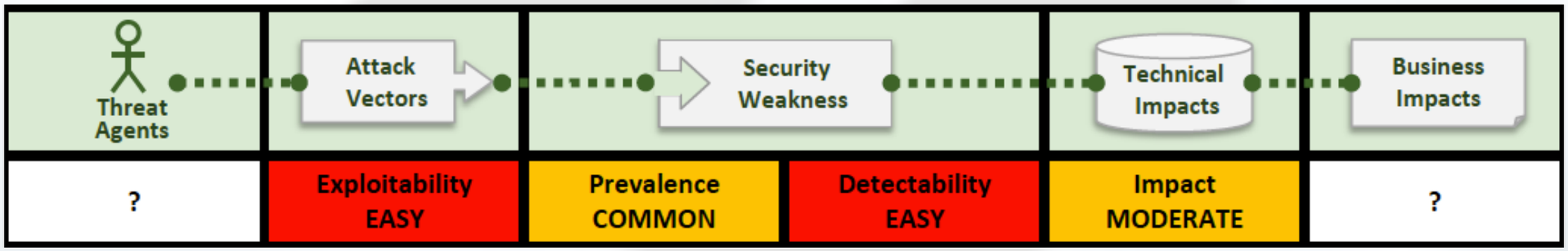
Example: Of course





Then what? – Change admin password, create users…

A5 Security Misconfiguration



The attacker uses tools with input data such as lists with known usernames/passwords

Many possible attack points, (server, application, framework)

Run tools, scan server and application for open ports and known files

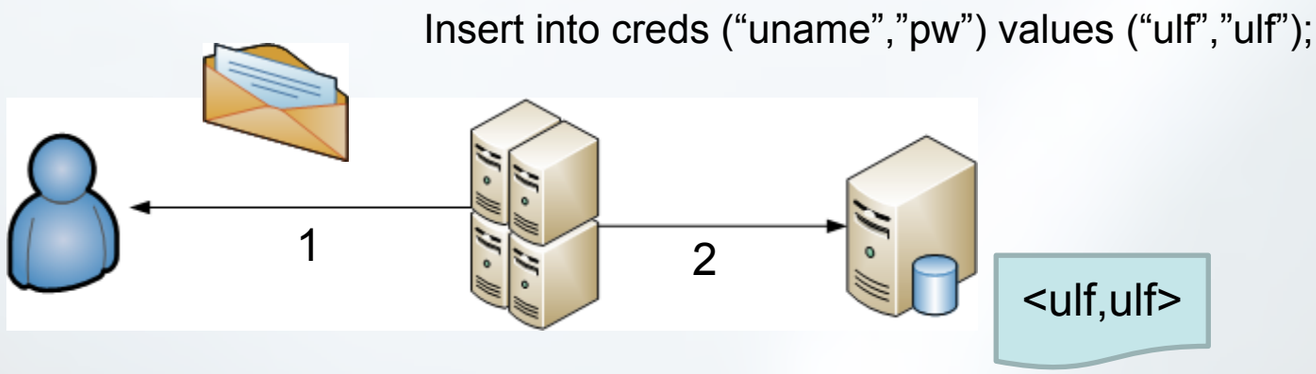
Some attacks may provide admin or root privileges, other only access to forgotten dummy files

A6 Sensitive Data Exposure

Was A7 and A9 from 2010

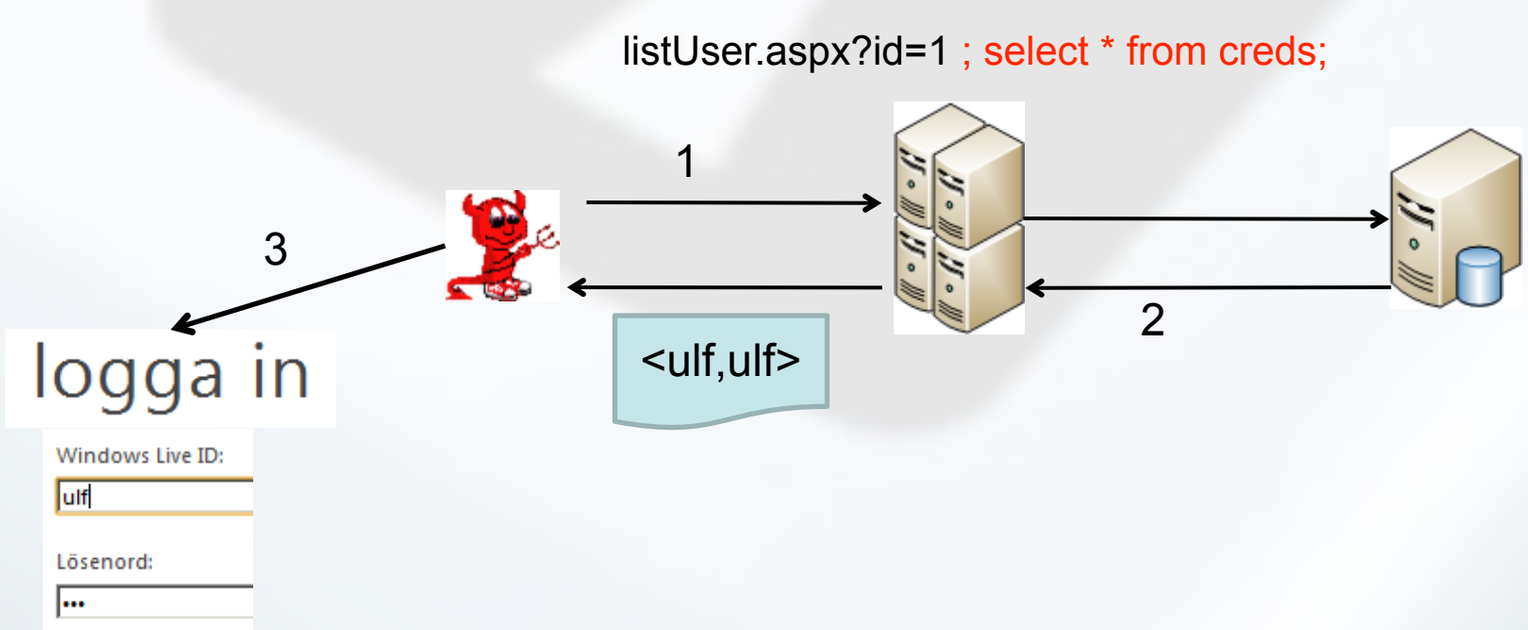
A web application stores sensitive information (credit card details, PII, patient journals) in plaintext or with insufficient cryptography or hash. An attacker can then access and use the information.

Example #1: password is stored on disk in plain text



A6 Sensitive Data Exposure

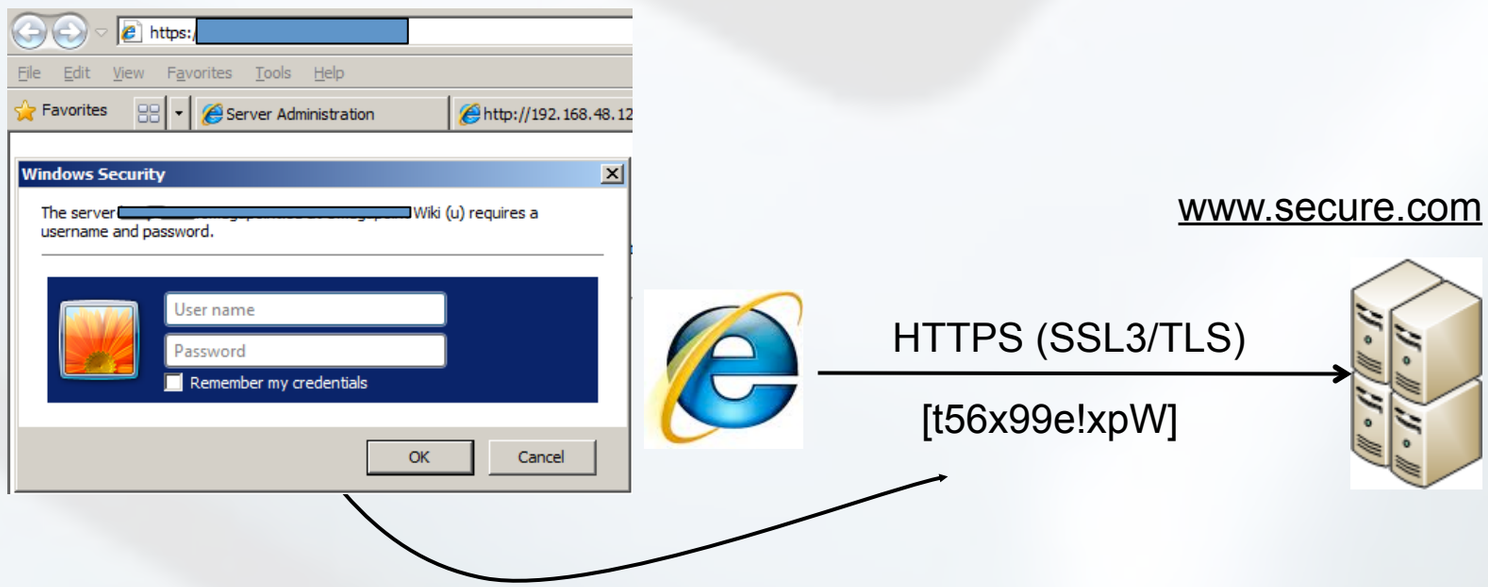
...and are then stolen at a successful intrusion



A6 Sensitive Data Exposure

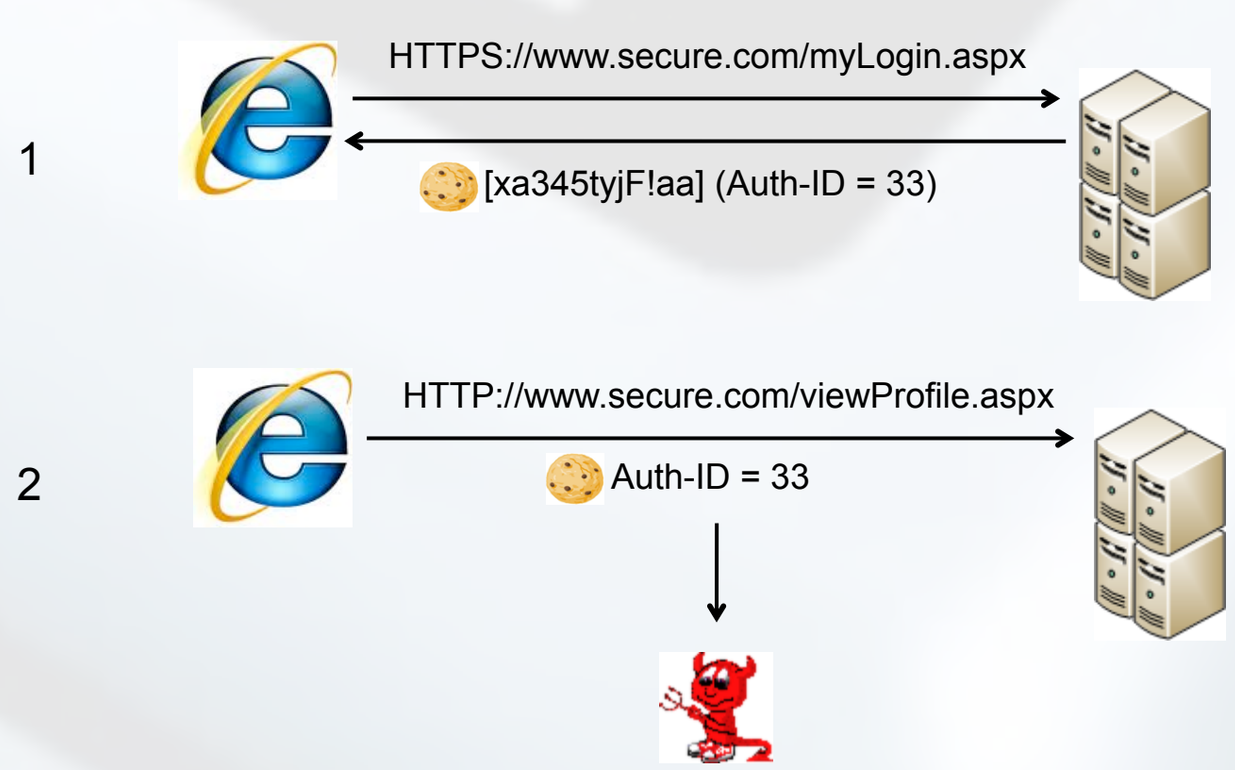
Sensitive information (e.g., user credentials) are transmitted between client and server. If the transmission channel is unprotected, uses weak encryption or erroneous certificates, the attacker can use this.

Example: (intended use)

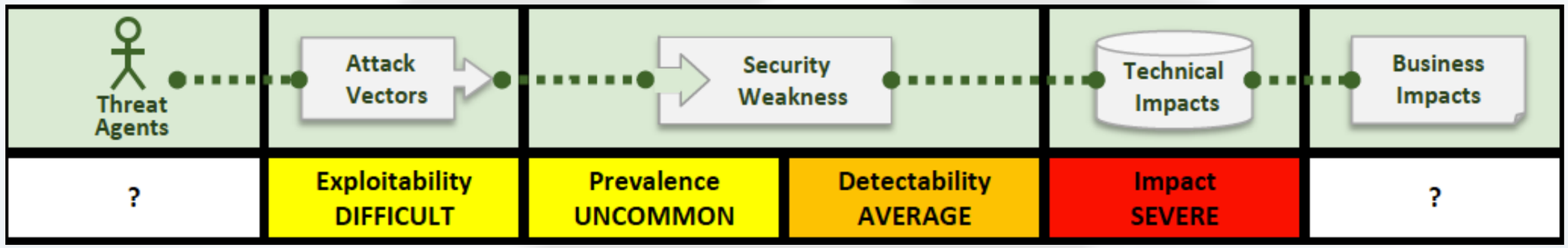


A6 Sensitive Data Exposure

Example (misconfiguration -> inconsistent use of HTTP/HTTPS)



A6 Sensitive Data Exposure



Should be quite difficult to break cryptos, attackers resort to steal data from server or capture plaintext in transit

Laws and regulations enforces data protection of sensitive data. However, exceptions are always present. Weak key generation and management, and weak algorithm usage is common, particularly weak hashing solutions to protect passwords.

External attackers have difficulty detecting most of these types of flaws due to limited access and they are also usually hard to exploit.

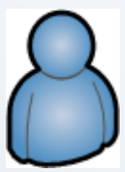
Failure frequently compromises all data that should have been protected. Typically this information includes sensitive data such as health records, credentials, personal data, credit cards, etc.

A7 Missing Function Level Access Control

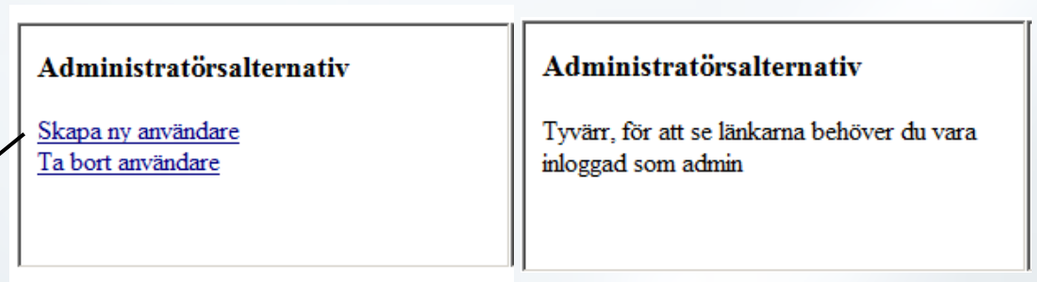
Was A8 from 2010

An access control is performed before a link is rendered in the browser. If a user rather than clicking the link directly navigates to the address the link points to, the control is not executed

Example: Access control is enforced before the links are rendered



Logged in as admin

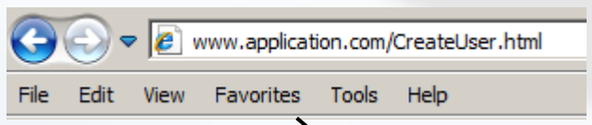


<a href="CreateUser.html">Skapa …

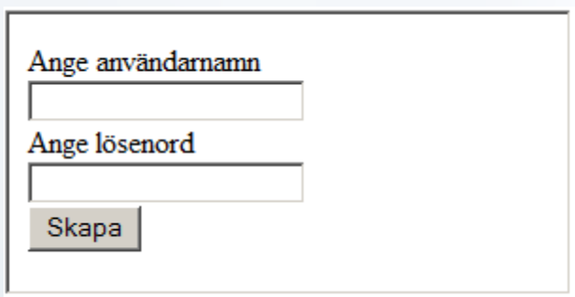
Not logged in as admin

A7 Missing Function Level Access Control

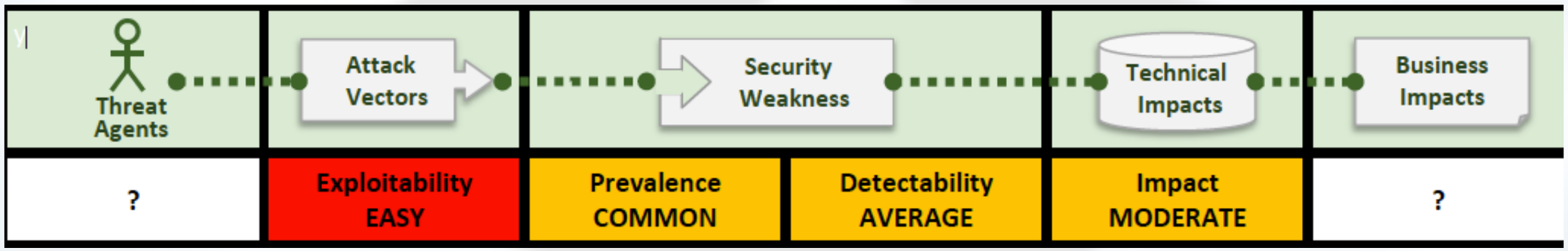
Example: What happens if you instead…



…directly in the address field submits the address to the page where the link points?



A7 Missing Function Level Access Control



The attacker inputs different addresses. It might work, or not...

Faulty configurations and proramming flaws exist

Depends on whether the pages have logical names or not. You might be able to guess that a page that creates users is called createUser.html

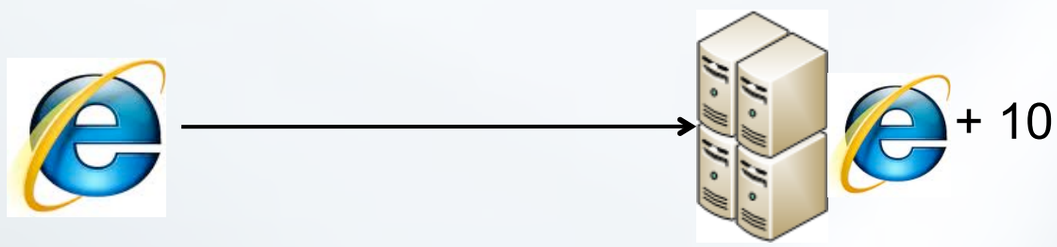
If you manage to get access to administrative functions, you can most likely affect users and possibly also the system

A8 Cross-Site Request Forgery (CSRF)

An attacker creates a normal server request and then tricks a logged in user to make this request. The request is made using the logged in user's credentials

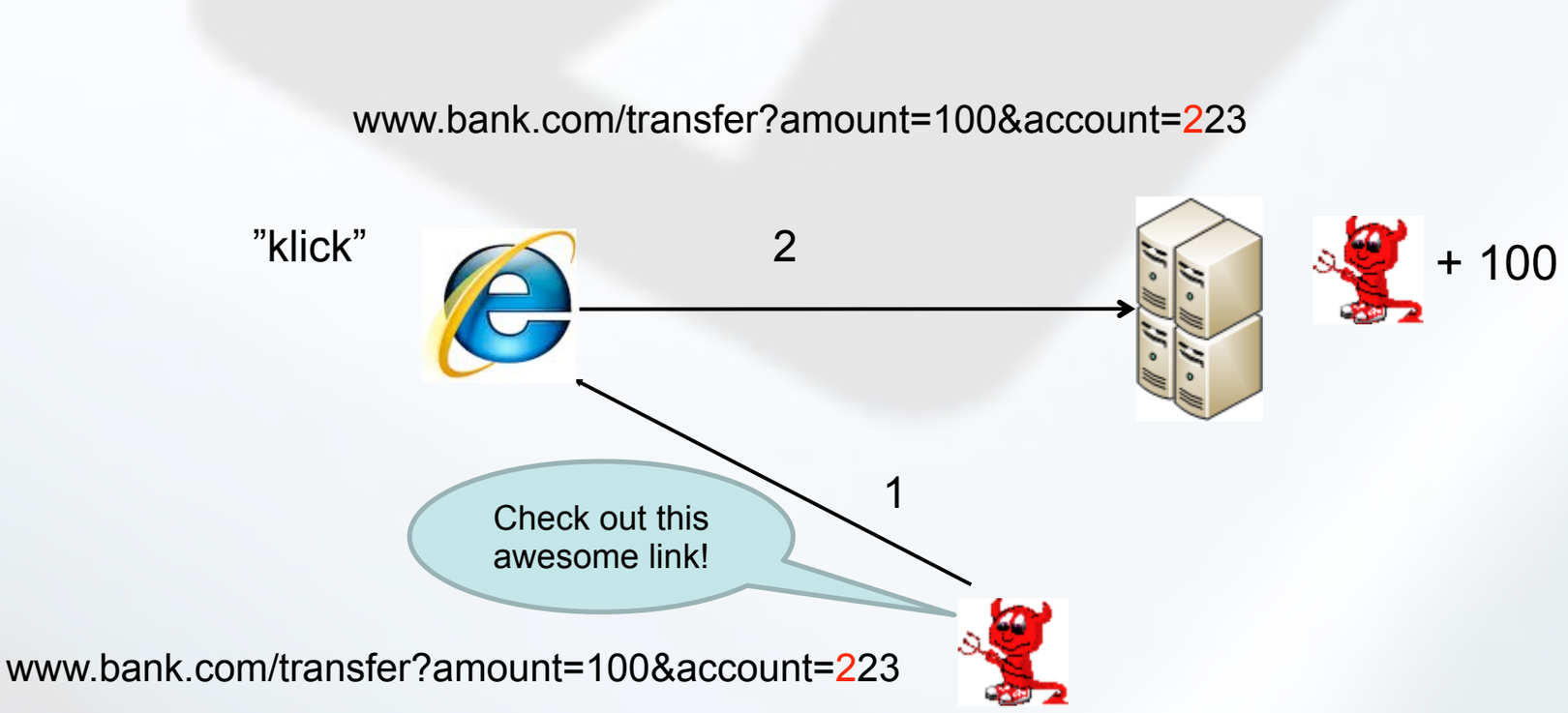
Example (normal use)

www.bank.com/transfer?amount=10&account=123

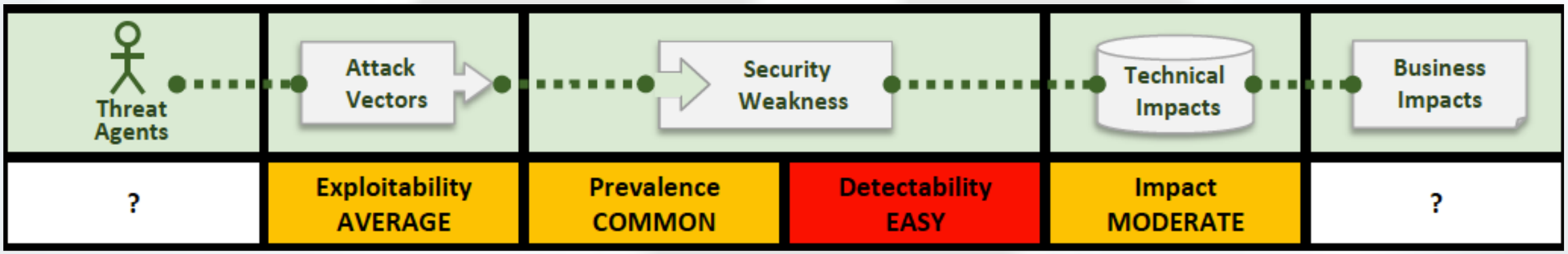


A8 Cross-Site Request Forgery (CSRF)

Example: (not-so-intended use)



A8 Cross-Site Request Forgery (CSRF)



The attacker must trick the user, the user must be logged in

Requests between client and server are made... often

Check calls for random values, so called CSRF tokens, If they are not present it should be straight forward to create a correct query

It is possible to exploit vulnerabilities within the permission bounds that the tricked user has

A9 Using Components with Known Vulnerabilities

An application contains a library or perhaps “googled” code with known vulnerabilities. This may expose a flaw in the application even if the surrounding framework consequently adapts a specific protection technique

A9 Using Components with Known Vulnerabilities

Output

Lasse

<XSS>

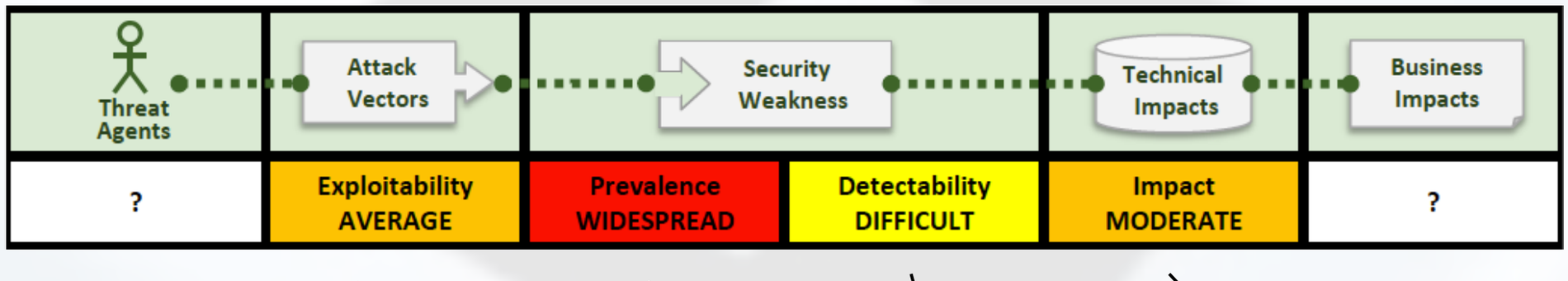
Source code

```
<script type="text/javascript">

    $(function() {
        var availableTags = [
    "<XSS> --[redacted]"

                ];
            $( "#ent" ).autocomplete({
                source: availableTags
            });
        });
</script>
```

A9 Using Components with Known Vulnerabilities



Attacker identifies a weak component through scanning or manual analysis. They customize the exploit as needed and execute the attack.

Virtually every application has these issues because most development teams don't focus on ensuring their components stay up to date.

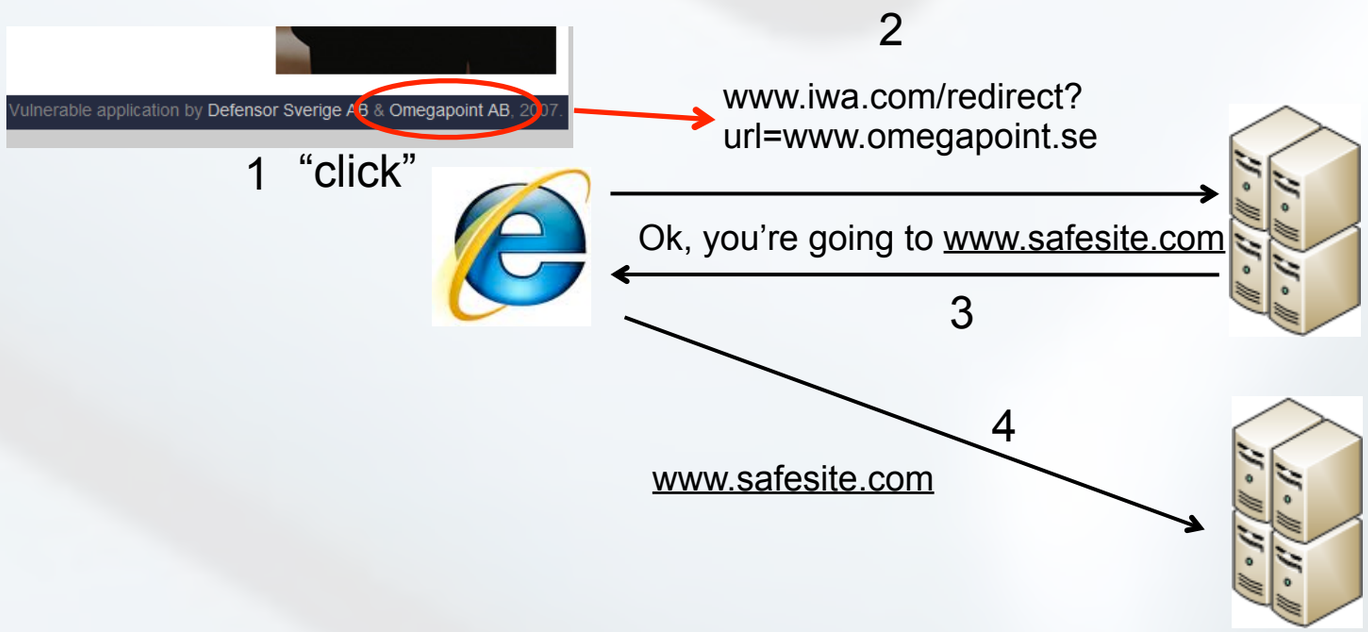
The deeper in the application the component is used, the more difficult it gets to exploit it.

The full range of weaknesses is possible, including injection, broken access control, XSS, etc. The impact could be minimal, up to complete host takeover and data compromise.

A10 Unvalidated Redirects and Forwards

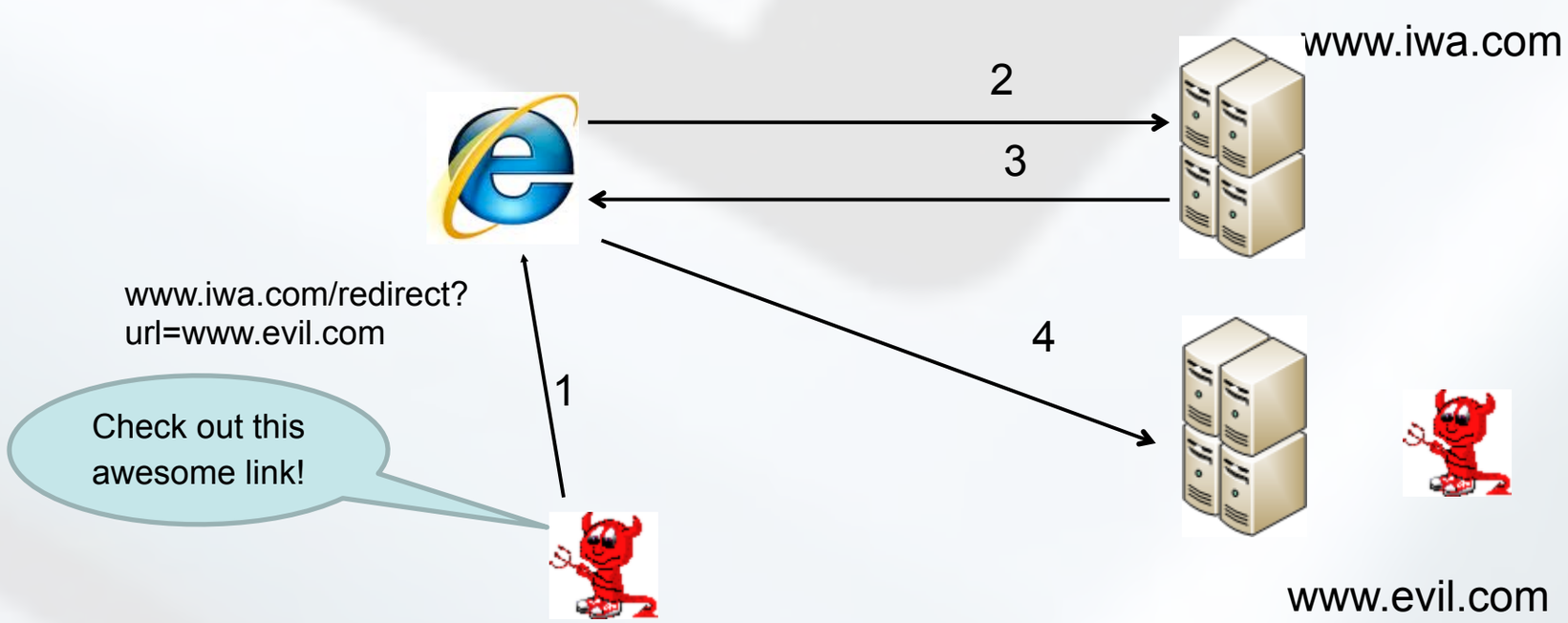
An application performs a redirect or a forward of a request, based on data that the attacker can affect. The attacker can then redirect the call to a target it selects, such as a phishing site

Example: (normal use)

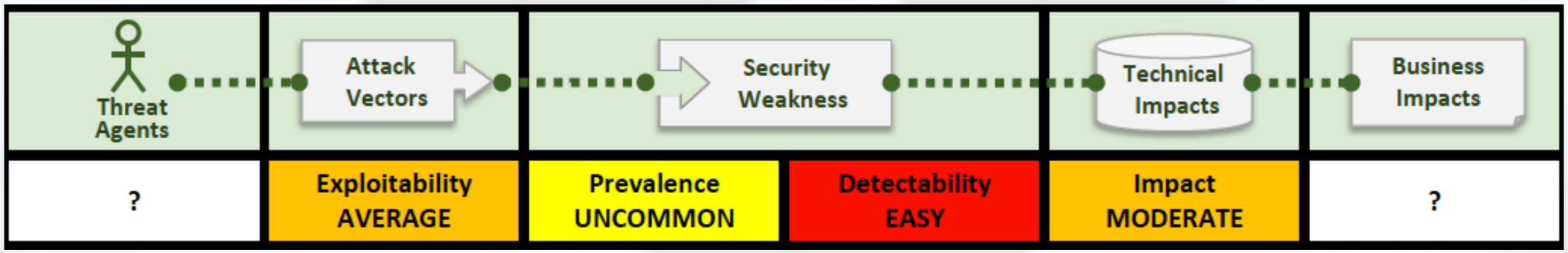


A10 Unvalidated Redirects and Forwards

Example: (not-so-intended use)



A10 Unvalidated Redirects and Forwards



The user must be tricked into believing that the link is legit. The receiving site must thus be credible, i.e, not www.myevilsite.com

Redirects and forwards are fairly common. However, less common that they are directly based on request parameters

Find a redirect, set a URL, and check to see if you get transferred to the provided URL

Possibility that a user is tricked into downloading malware. Possibility to bypass access controls