Understanding ASP.NET Core Security

PROTECTING YOUR APPLICATION AGAINST COMMON ATTACKS

Roland Guijt
CONSULTANT | TRAINER | AUTHOR | MVP

@rolandguijt rolandguijt.com

Understanding ASP.NET Core

Overview

ConfArch introduction

Enforcing SSL

SQL Injection

Cross Site Request Forgery

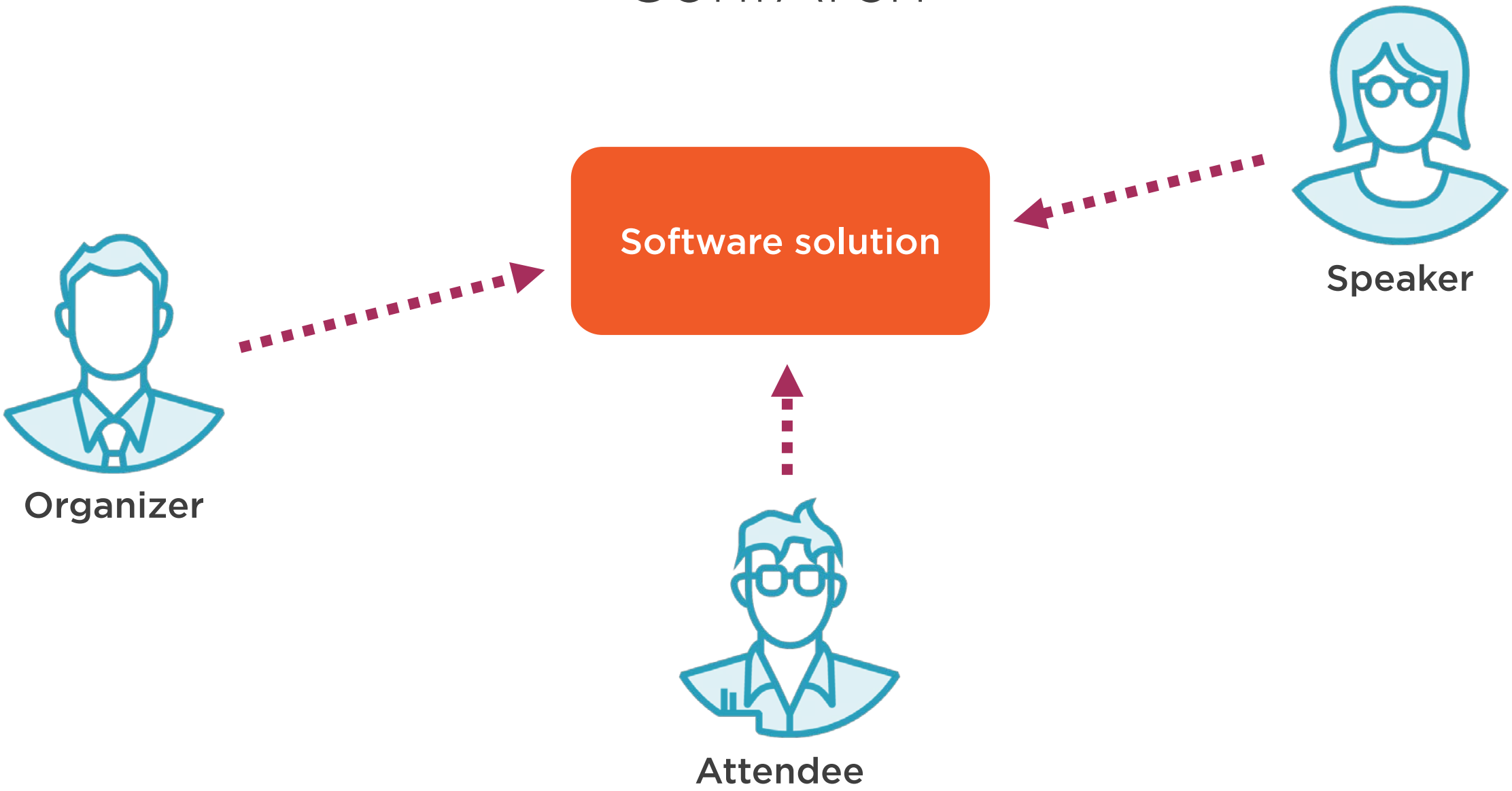
Cross Site Scripting

Open Redirection Attacks

Click jacking

Same Origin Policy

ConfArch



Enforcing SSL

Demo App

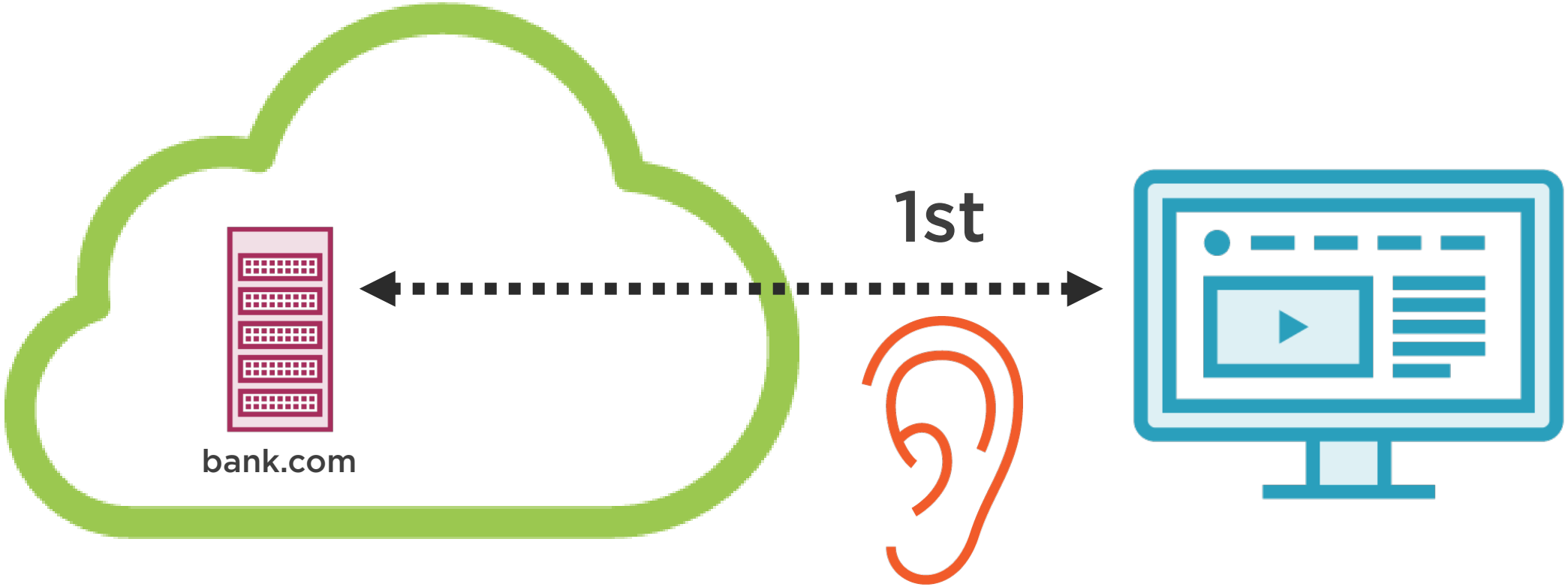
No extra security measures in place

No auth yet

Built with standard ASP.NET Core dependency injection pattern

Repositories

Enforcing SSL: HSTS



Enforcing SSL: HSTS

HSTS Header

```
Strict-Transport-Security: max-age=31536000;
```

There is no way back.

Enforcing SSL: HSTS

Enforcing SSL: HSTS

SQL Injection

```
public IActionResult Customer(string name)
{
   var query = $"select * from customers where name={name}";
   //execute query
}
```

SQL Injection

select * from customers where name='{name}'

What would be the output if username was:

' OR '1'='1' --

select * from customers where name='' or '1'='1' -- '

a'; DROP TABLE customers; --

select * from customers where name='a'; DROP TABLE customers-- '

SQL Injection

User input copied to SQL statement

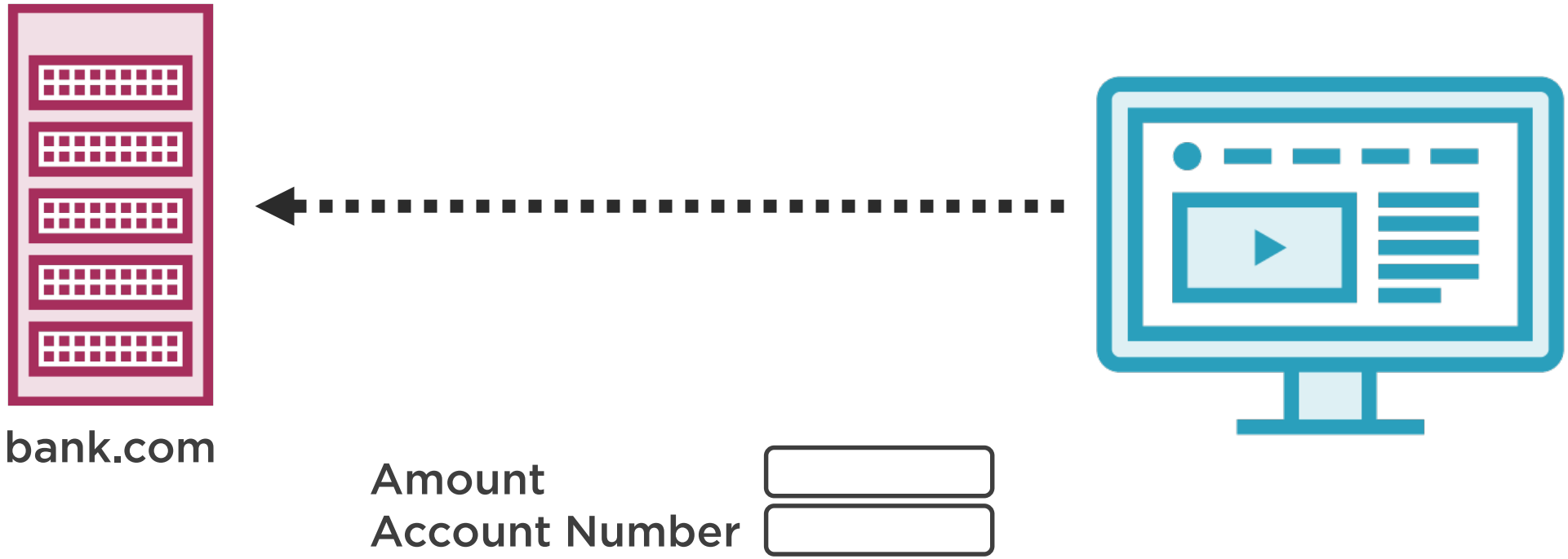
Attacker potentially has full control over database

Check the input

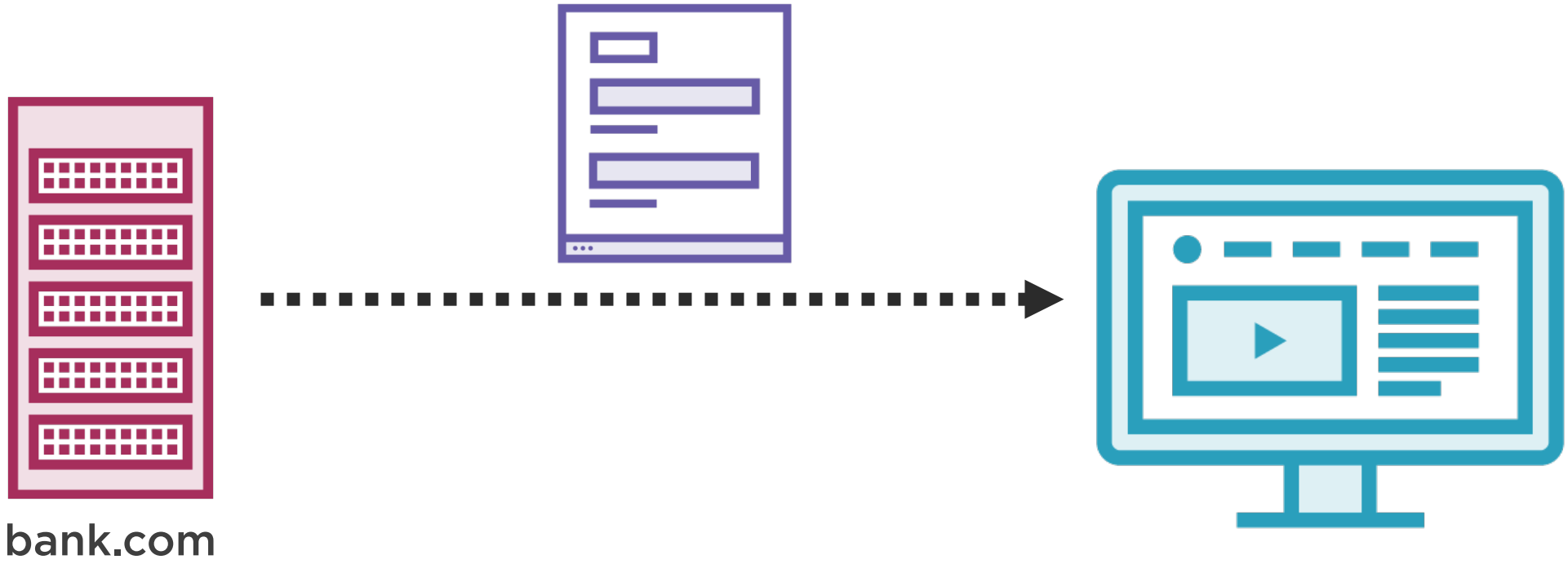
Use least privileged account

Use ORM like Entity Framework

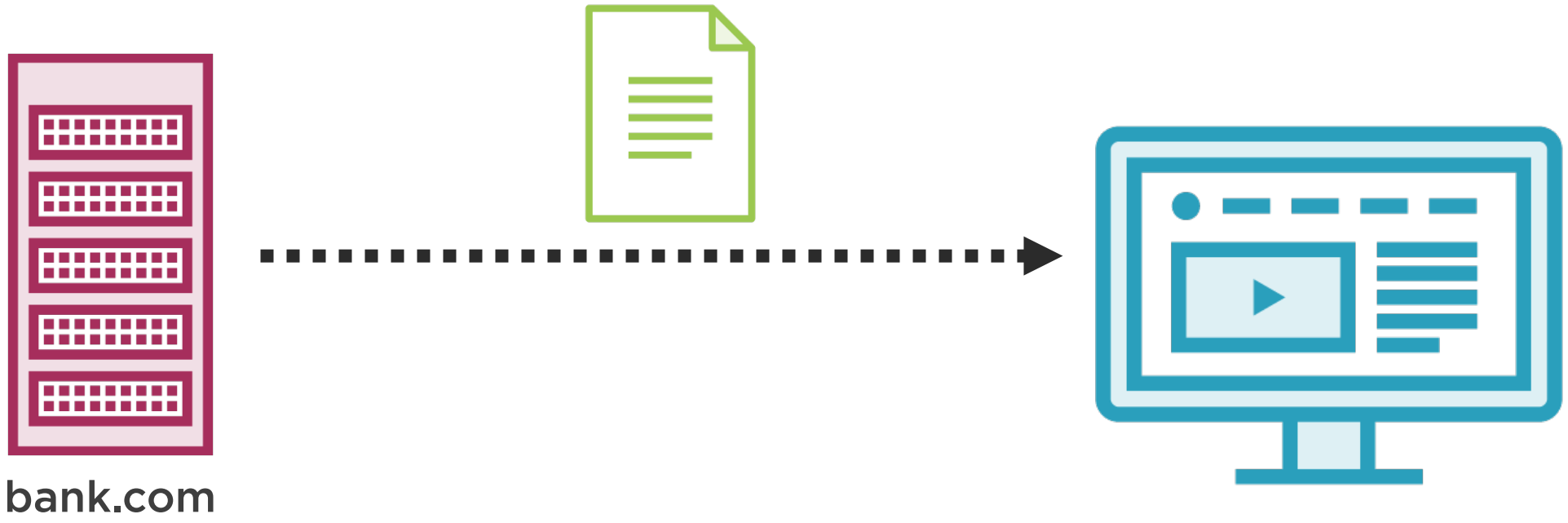
Cross Site Request Forgery (CSRF)



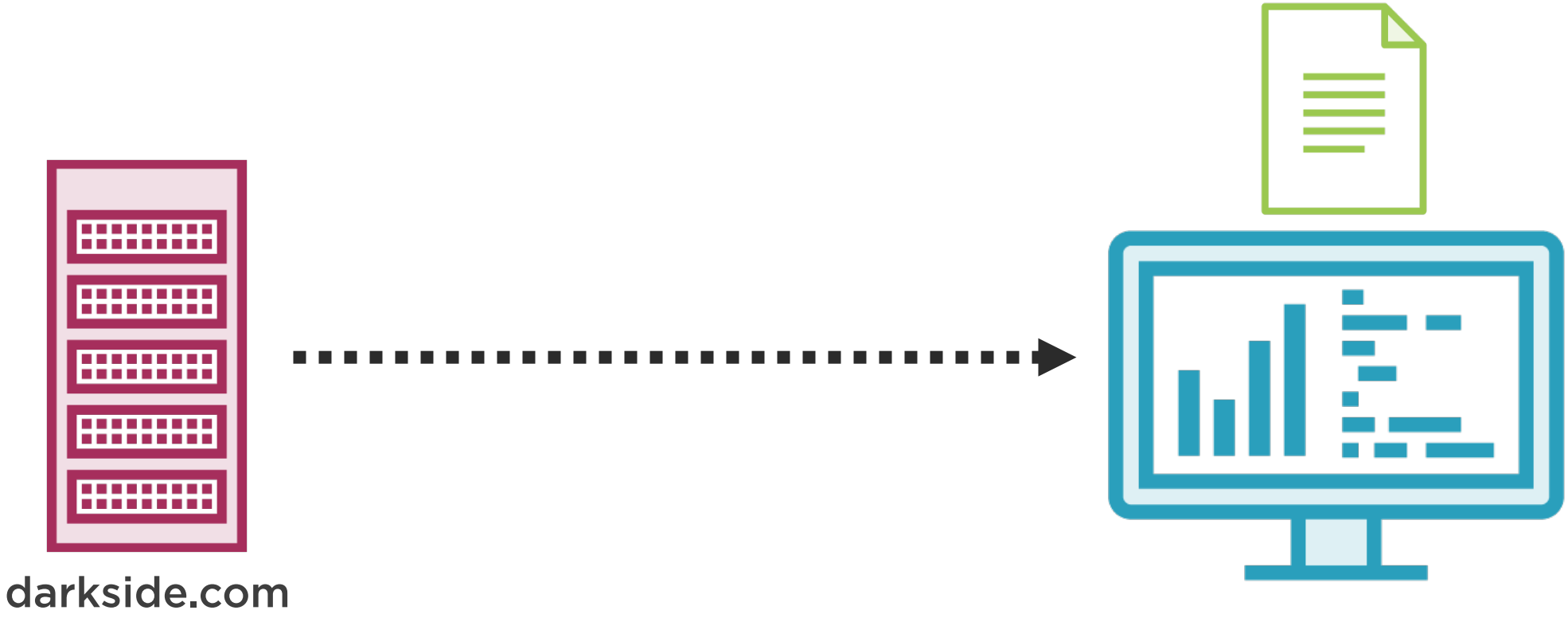
Cross Site Request Forgery (CSRF)



Cross Site Request Forgery (CSRF)



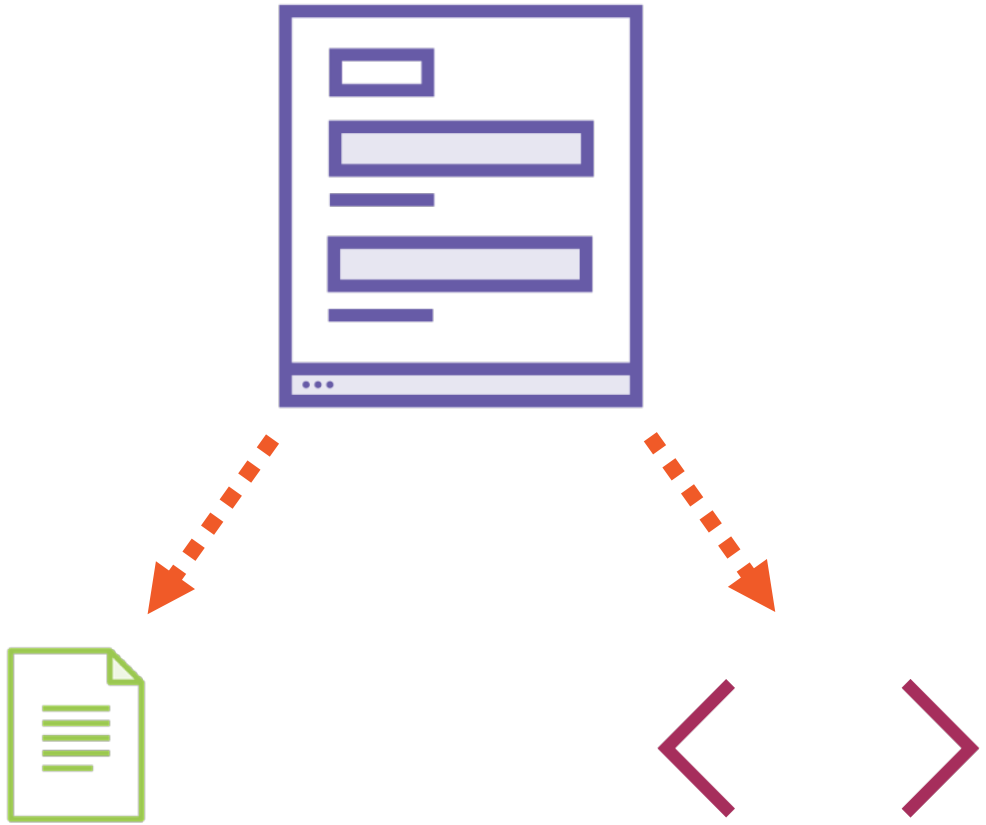
Cross Site Request Forgery (CSRF)



Cross Site Request Forgery (CSRF)

```
<h1>You have won $1000!</h1>
<form action="http://bank.com/api/transfer" method="post">
  <input type="hidden" name="Amount" value="1000000"/>
  <input type="hidden" name="AccountNumber" value="4356283"/>
  <input type="submit" value="Click here to redeem!!"/>
</form>
```

Cross Site Request Forgery (CSRF)



Cross Site Scripting (XSS)

Attacker places JavaScript into webpage

Provides attacker access to everything in the browser

Encoding replaces dangerous characters

MVC encodes everything that is in a variable automatically

Cross Site Scripting Solutions

Encode all the things

Use Content Security Policy (CSP)

Content Security Policy (CSP)

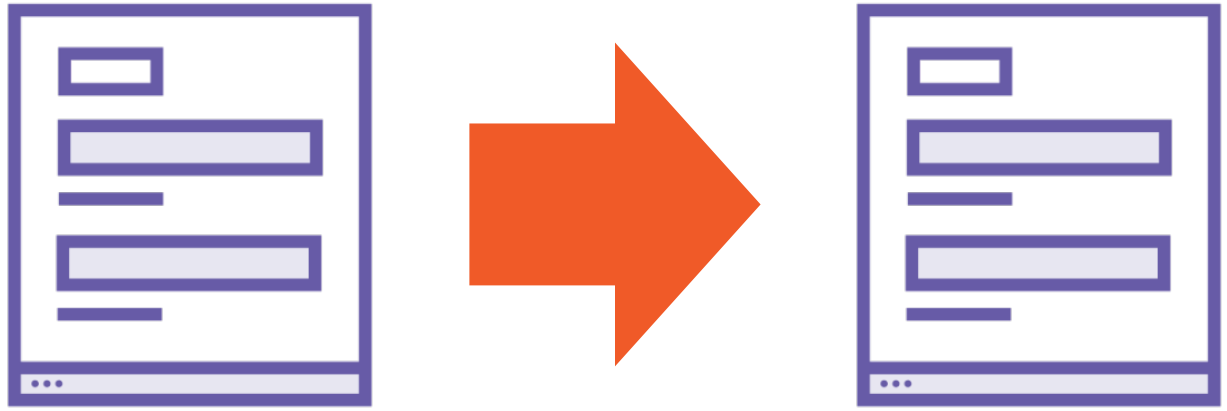
```
Content-Security-Policy: script-src 'self'
```

CSP Content Types

- script-src
- style-src
- img-src
- media-src
- frame-src
- font-src
- default-src

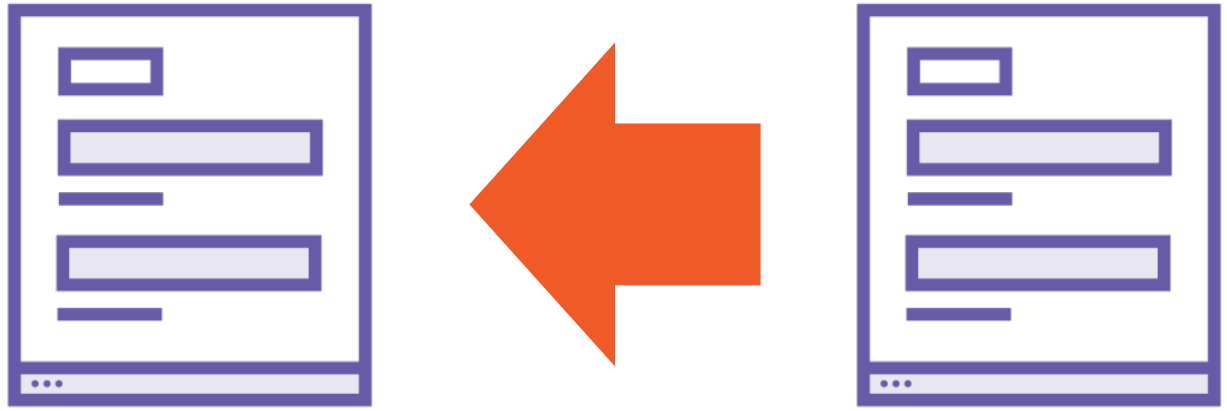
Open Redirection Attack

http://bank.com/Account/LogOn?returnUrl=http://bank.net/Account/LogOn



Open Redirection Attack

http://bank.com/Account/LogOn?returnUrl=http://bank.net/Account/LogOn



Open Redirection Attack

```
if (!Url.IsLocalUrl(returnUrl))
{
    //throw
}
```

Click-jacking

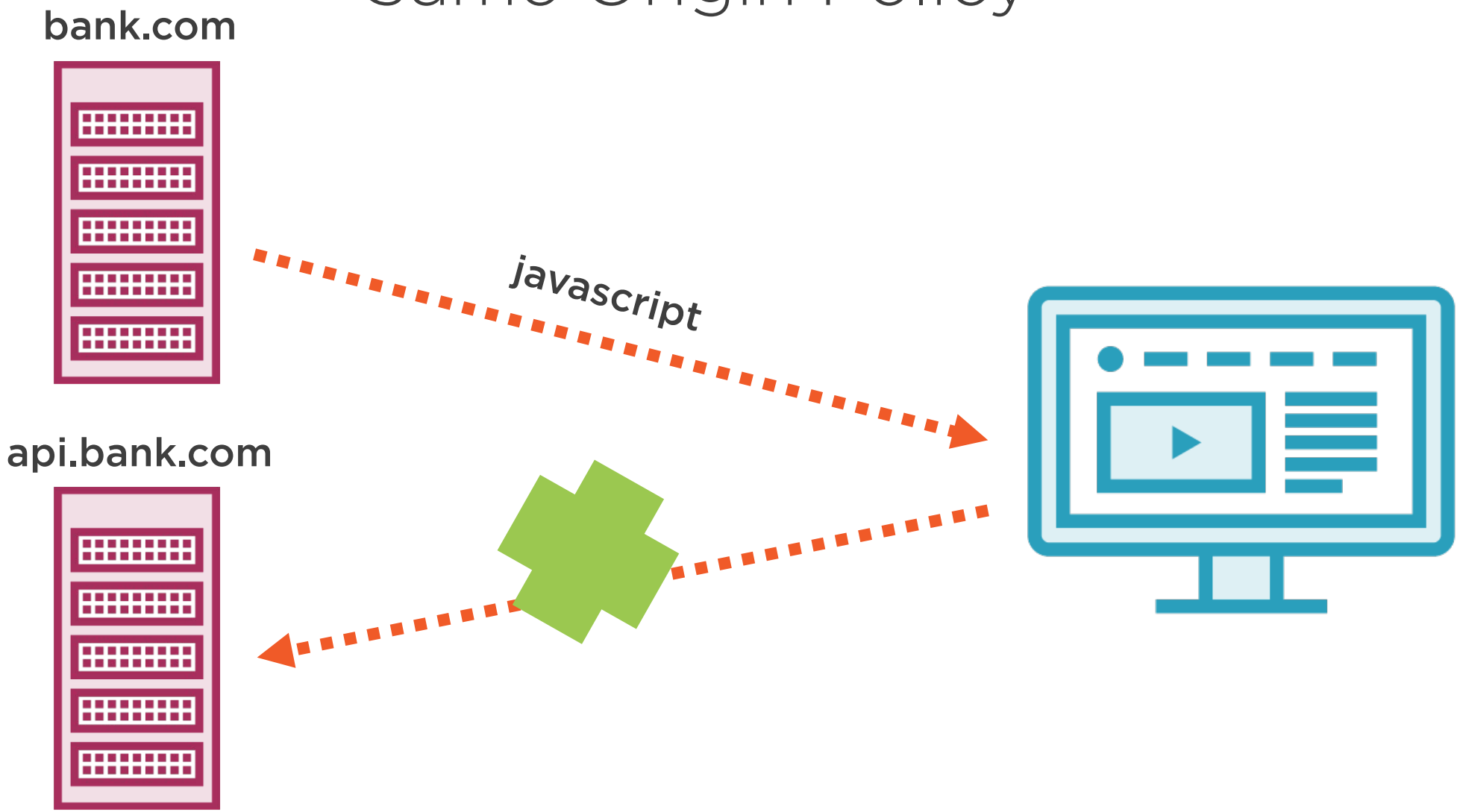
Click-jacking

```
X-Frame-Options: DENY

X-Frame-Options: SAMEORIGIN

X-Frame-Options: ALLOW-FROM https://example.com/
```

Same Origin Policy



CORS

Request

Origin: http://bank.com

Response

Access-Control-Allow-Origin: http://bank.com
or
Access-Control-Allow-Origin: *

Preflight Request Skip Conditions

The request method is GET, HEAD, or POST AND

The application does not set any request headers other than Accept, Accept-Language, Content-Language, Content-Type, or Last-Event-ID AND

The Content-Type header (if set) is one of the following:

- application/x-www-form-urlencoded
- multipart/form-data
- text/plain

Summary

- Adding protection from common attacks is easy
- The MVC framework and NWebSec are a great help
- CORS support allows configuration of cross-origin requests