BECOME A SECURITY NINJA

Don’t let your code be low-hanging fruit

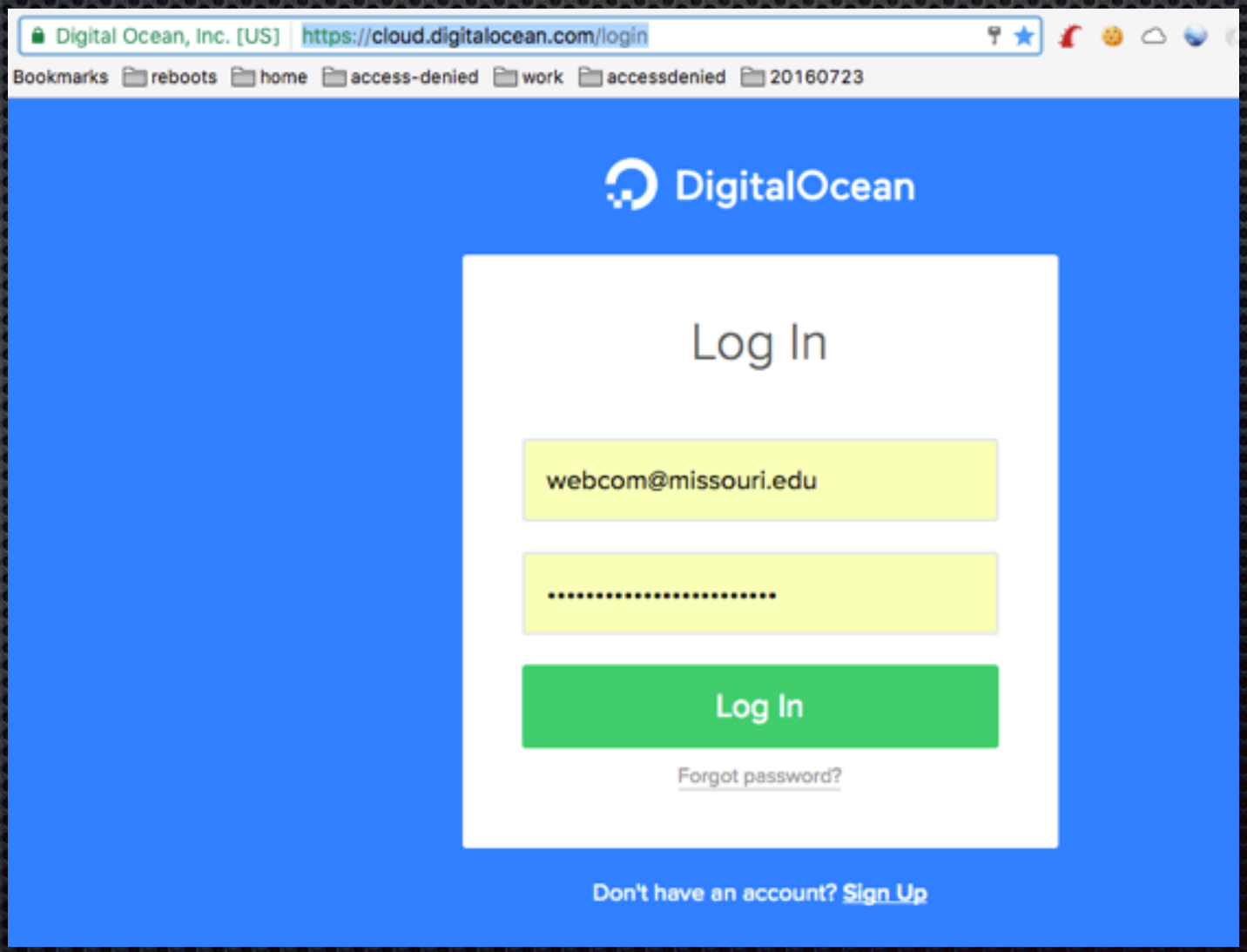
Paul Gilzow
gilzow@missouri.edu
Twitter: @gilzow
Facebook: /gilzow
https://www.linkedin.com/in/gilzow

What we will talk about

- Get our example files up and running
- What is OWASP
- What is the OWASP Top 10
- The Current Top 10 Web Security Risks
- ATTACK!

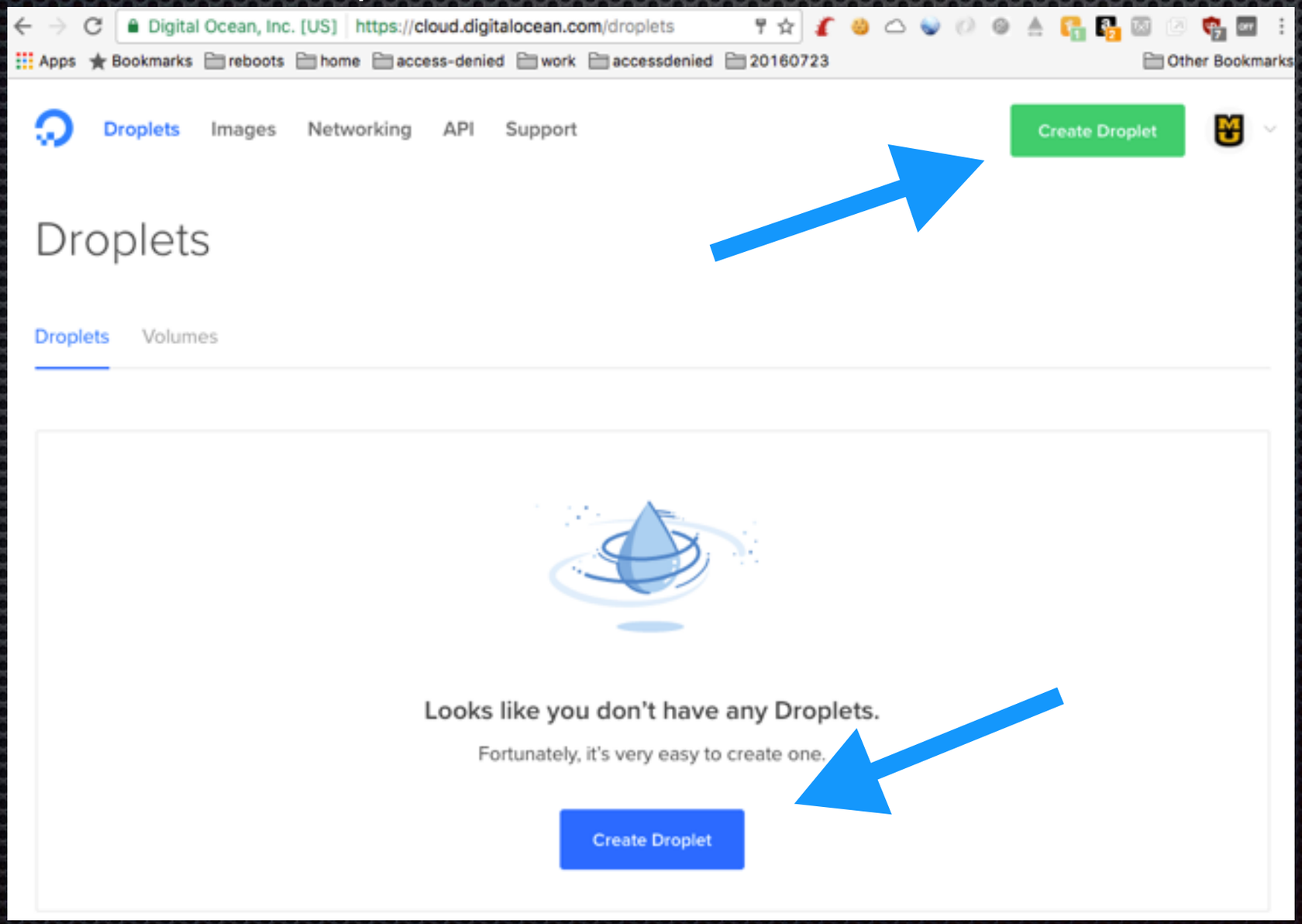
Set Up Our Exercise files

- Log into your Digital Ocean account https://cloud.digitalocean.com/login



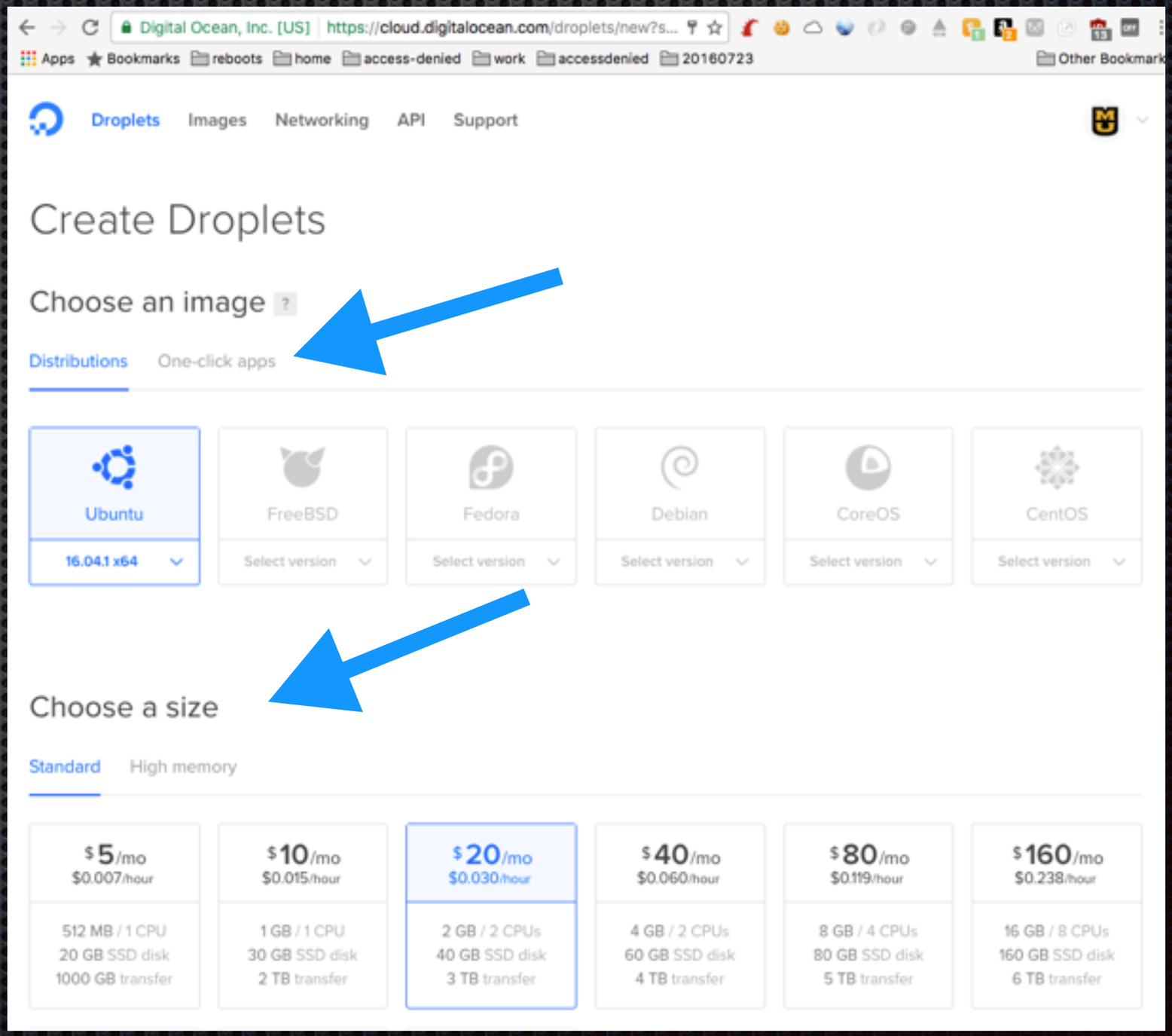
Set Up Our Exercise files

- Click on “Create Droplet”



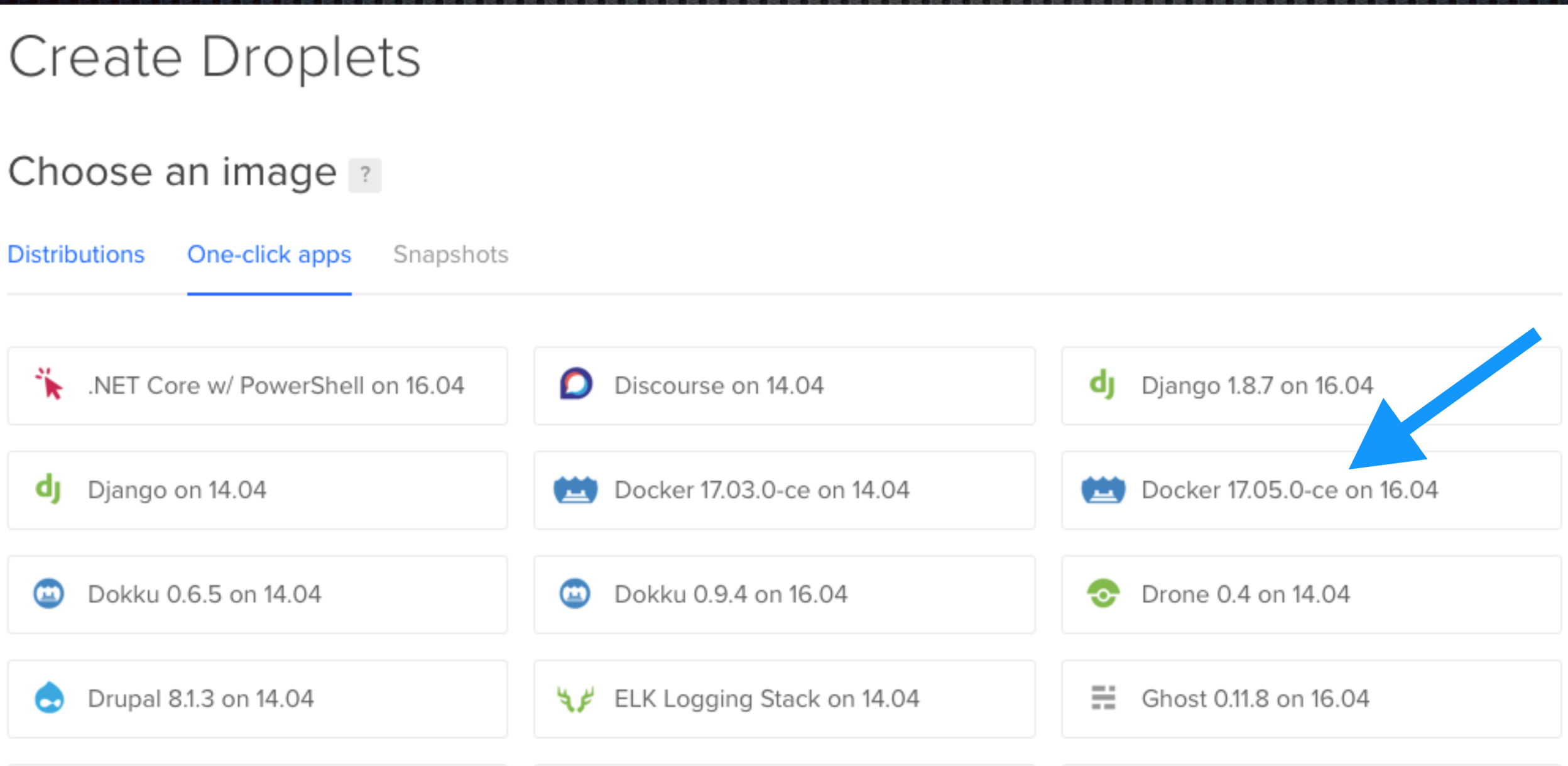
Set Up Our Exercise files

- Choose a size
- Click on “One-click apps”



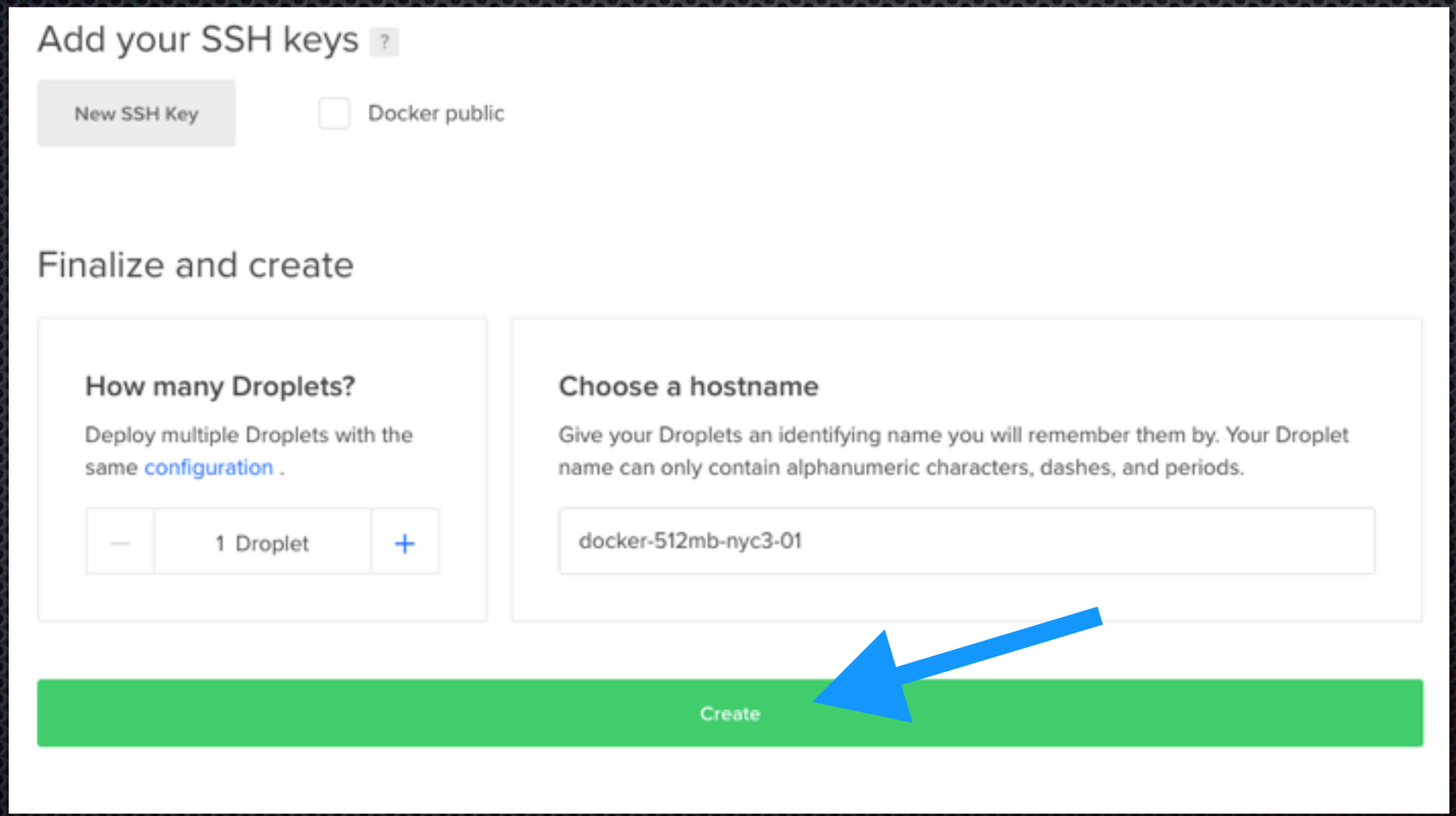
Set Up Our Excersice files

- Click on Docker



Set Up Our Excersice files

- Scroll to the bottom of the page
- Click the “Create” button



Set Up Our Excersice files

- Once the instance is created, check your email for the credentials
- In your ssh client, log into your new instance

```
mu-039068:/ gilzowp$ ssh root@45.55.167.245
The authenticity of host '45.55.167.245 (45.55.167.245)' can't be established.
ECDSA key fingerprint is SHA256:5SJVchJnbhXxcXAwGboadquUazi176J6m+Bz3PrBA+k.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '45.55.167.245' (ECDSA) to the list of known hosts.
root@45.55.167.245's password:
You are required to change your password immediately (root enforced)
Welcome to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-38-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

0 packages can be updated.
0 updates are security updates.


--------------------------------------------------------------------------------
Thank you for using DigitalOcean's Docker Application.

Docker has been preinstalled and configured per Docker's Recommendations.
```

Set Up Our Excersice files

- You'll need to change your password the first time you login

```
--------------------------------------------------------------------------------
Thank you for using DigitalOcean's Docker Application.

Docker has been preinstalled and configured per Docker's Recommendations.

"ufw" has not been enabled, however it has been configured. To enable it,
run "ufw enable".

Let's Encrypt has been pre-installed for you. If you have a domain name, and
you will be using it with this 1-Click app, please see: http://do.co/le-apache

--------------------------------------------------------------------------------

You can learn more about using this image here: http://do.co/docker

--------------------------------------------------------------------------------
To delete this message of the day: rm -rf /etc/update-motd.d/99-one-click
Last login: Wed Sep 21 20:47:49 2016 from 128.206.39.68
Changing password for root.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
root@docker-512mb-nyc3-01:~#
```

Set Up Our Excersice files

- At the prompt type in

  docker run -d -p 8899:80 opendns/security-ninjas

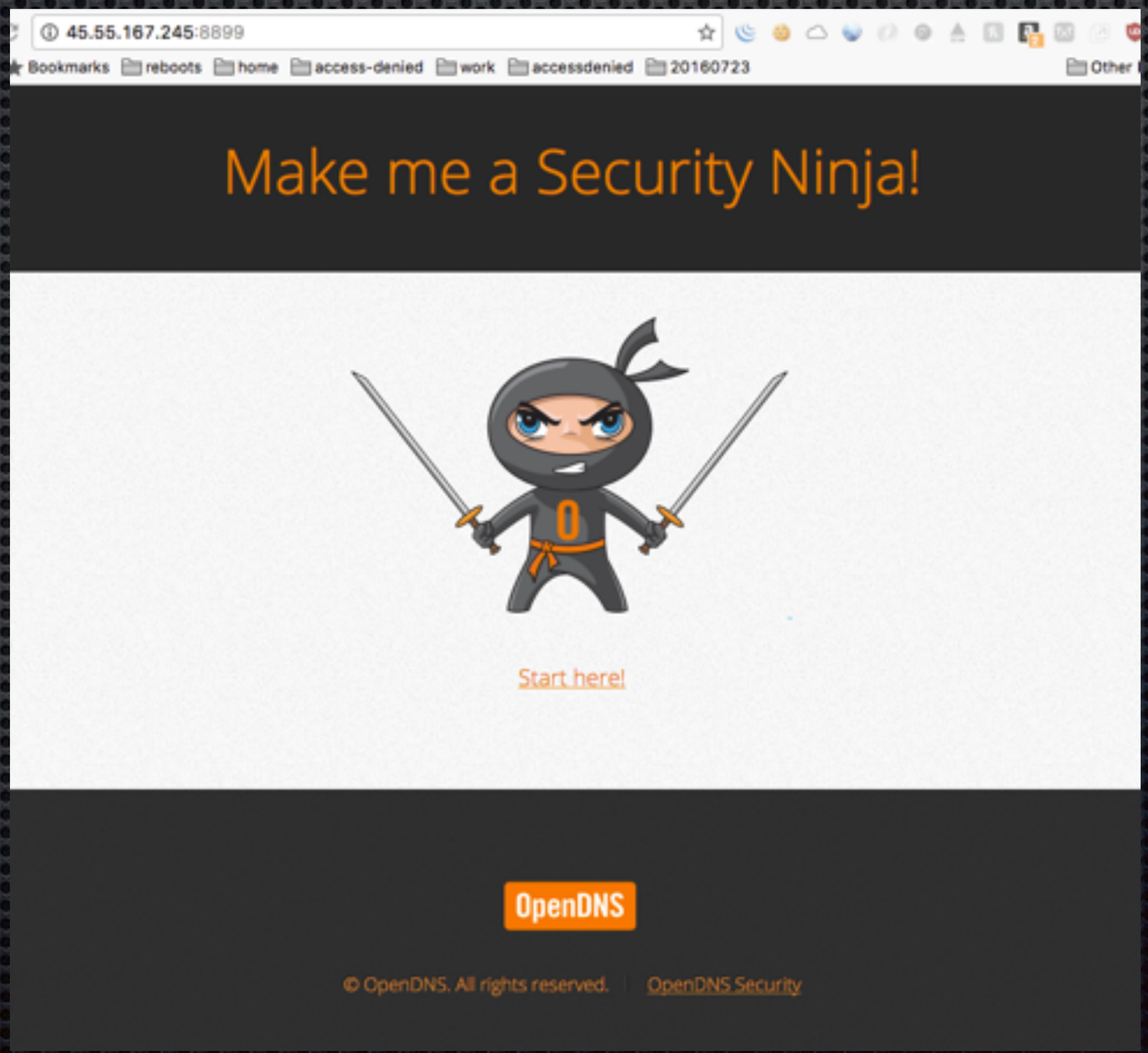
- Docker will download and install the security-ninjas container

```
root@docker-512mb-nyc3-01:~# docker run -d -p 8899:80 opendns/security-ninjas
Unable to find image 'opendns/security-ninjas:latest' locally
latest: Pulling from opendns/security-ninjas

e190868d63f8: Pull complete
909cd34c6fd7: Pull complete
0b9bfabab7c1: Pull complete
a3ed95caeb02: Pull complete
f0df993c3aef: Pull complete
09358b20e8bc: Pull complete
5b0e1b98c45e: Pull complete
9c00e77d73a9: Pull complete
fe8e16791cf6: Pull complete
ac8b0ffa2b98: Pull complete
94051df6e066: Pull complete
Digest: sha256:aae15b7ca2827f1d4fcc5b38a238e8a207f1ebc1bd5eef921cef97f5f6262994
Status: Downloaded newer image for opendns/security-ninjas:latest
c20a58e1686fcd6d98045ebc5f360149d57a5f590f00504652c497d916df9849
root@docker-512mb-nyc3-01:~#
```

Set Up Our Excersice files

- Open your browser and go to your droplet ip address, port 8899
  e.g. http://45.55.167.245:8899



What is OWASP

- Open Web Application Security Project
- Worldwide non-project organization focused on improving software security by raising awareness
- Started in 2001
- Hundreds of local chapters throughout the world, with over a hundred in the U.S.

What is the OWASP Top 10

- Not a standard, but an awareness document
- 10 Most Critical Web Application Security **Risks**
- For each risk, it includes
  - Description
  - Example vulnerabilities
  - Example Attacks
  - Guidance on how to avoid
- First released in 2003, refreshed every 3 years
- Current version is from 2013, next version (2017) is in Release Candidate status

Current State of Web App security

- 35% increase in websites hosting malware between 2016 and 2017[1]
- 80% of web apps have at least one vulnerability[2]
- Average of 45 vulnerabilities per application[2]
- 25% of web applications vulnerable to 8 of the OWASP Top 10[2]

1 - Google Transparency Report, 2017
2 -Contrast Security Labs Web Application Vulnerability report 2017

Current Top 10 (as of 2013)

1. Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)
4. Insecure Direct Object Reference
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery (CSRF)
9. Using Components with Known Vulnerabilities
10. Unvalidated Redirects and Forwards

A1. Injection

What is it?

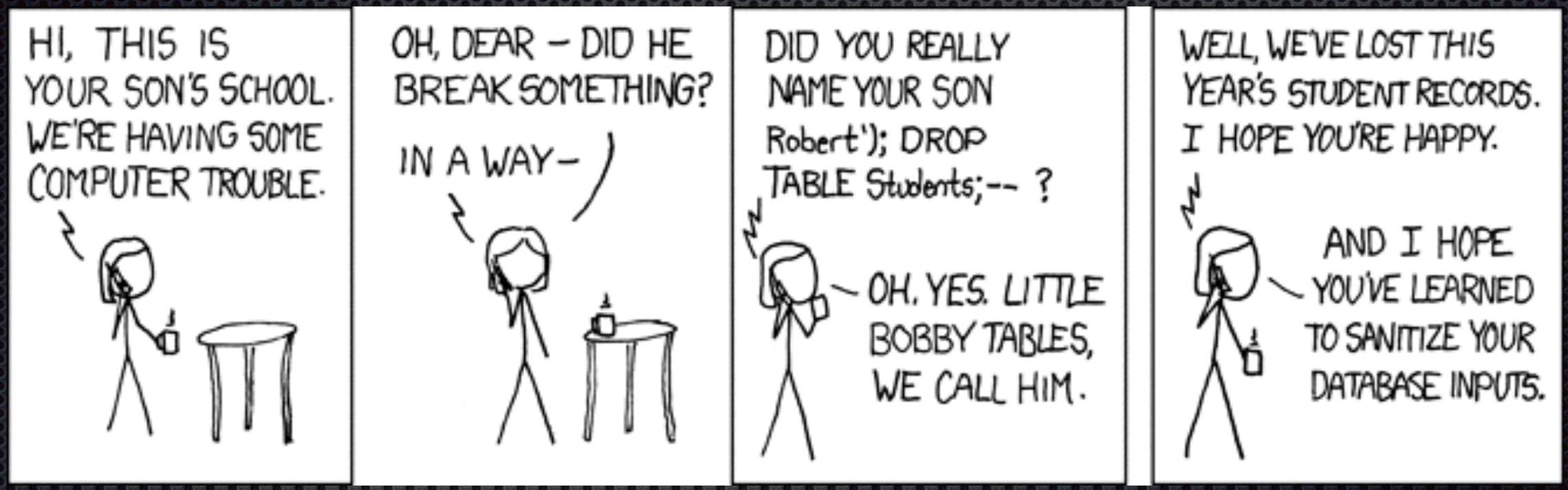
- Untrusted/tainted data is sent to an interpreter as part of a command or query.
- Interpreter is tricked into executing unintended commands

A1. Injection

What is susceptible?

- HTML/Javascript
- SQL
- LDAP
- Xpath/Xquery
- Anything that accepts input and executes

A1. Injection



A1. Injection

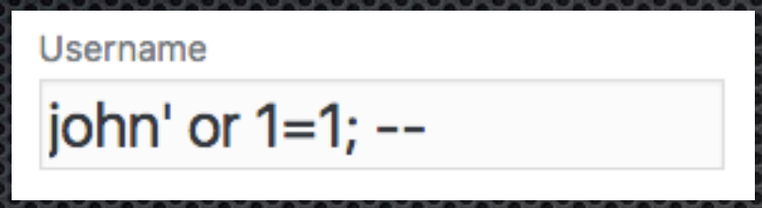
Why does it happen?

- Failure to separate untrusted data with commands sent to an interpreter
- Lack of input validation and/or input sanitation
- Attacker is able to change execution context

A1. Injection

Examples

- SQL Injection




```
SELECT UserID, Password from Users WHERE UserName = '$input';
```

Becomes

```
SELECT UserID, Password from Users WHERE UserName = 'john' or 1=1; -- ';
```

A1. Injection

ATTACK!

A1 A2 A3 A4 A5 A6 A7 A8 A9 A10

A1 : Injection

Whois Lookup Service

Domain Name

Go

There can be many types of Injection. This page has an OS Command Injection flaw.

Hint 1

Hint 2

Solution

A1. Injection

Prevention

- Use APIs that provide parameterized/sanitized interfaces
- Validate input against a whitelist (do NOT use a blacklist)
- Escape any special characters that you whitelist

A2. Broken Authentication and Session Management

What is it?

- Failure to protect credentials and session tokens throughout the lifecycle of the session
- Weak authentication logic
- Imperfect implementation

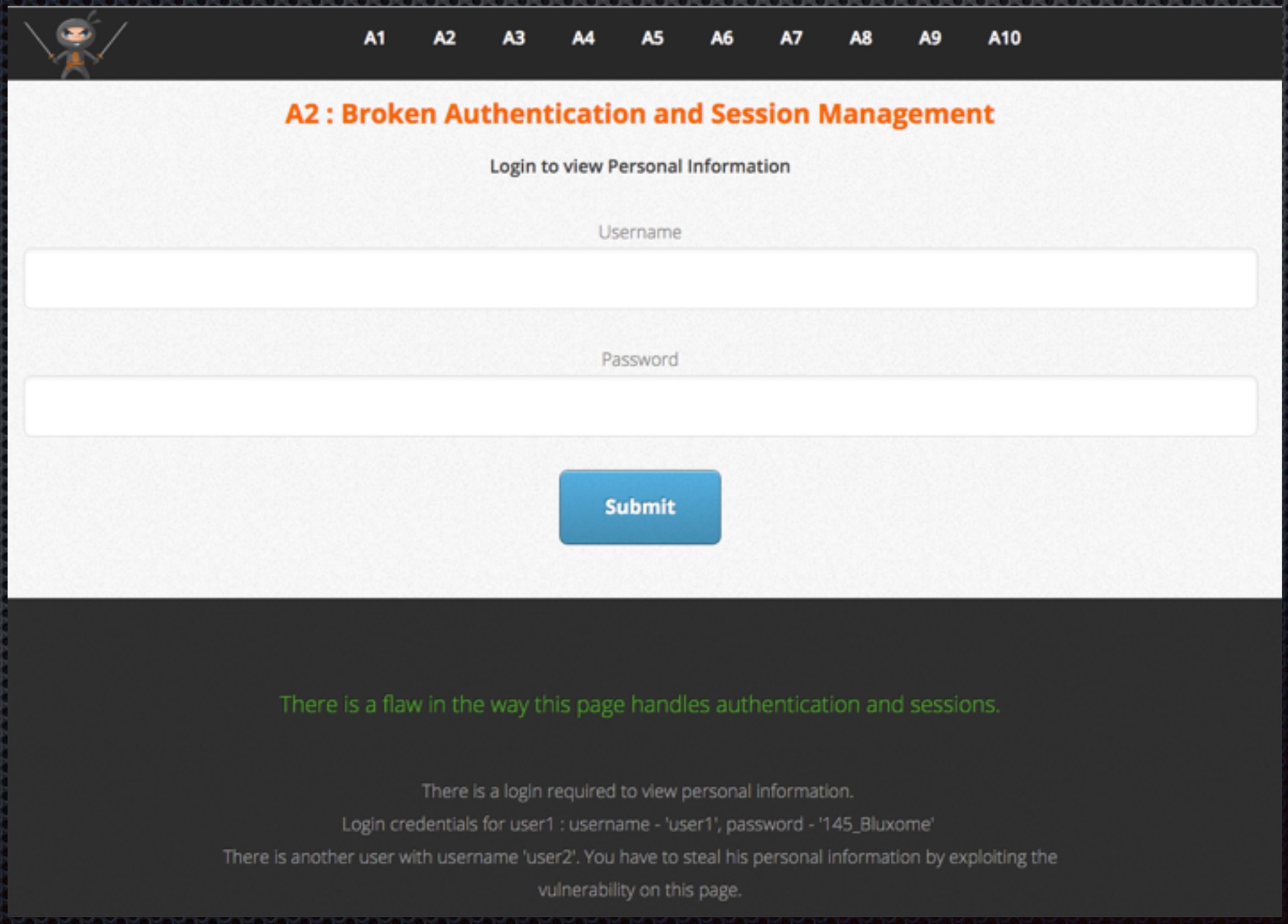
A2. Broken Authentication and Session Management

Why does it happen?

- Passing the session ID as a parameter in the URL
- Using a predictable value for the session ID
- User credentials passed unencrypted
- Poor username and password recovery functions

A2. Broken Authentication and Session Management

ATTACK!



A2. Broken Authentication and Session Management

Prevention

- Never send credentials/sensitive data in the clear; use SSL
  - Make sure *Secure* flag is set on your cookies **especially** the session cookie
  - Add Http Strict Transport Security header to your web/app server
- Protect secrets throughout their lifecycle
  - Don’t expose credentials (including session IDs) in the URL **or logs**
- Use centralized authentication and session management APIs, if available
- Implement strong account management functions (e.g., account creation, change password, recover password, etc.)

A2. Broken Authentication and Session Management

Prevention

- Use strong algorithms to generate (random) secrets
- Do not rely on untrustworthy information for authentication (e.g. IP address, referrer, etc.)
- Use a different session ID after authentication or whenever authorization/ permission levels change
- Destroy/invalidate the session ID **completely** after the log out method has completed
- Enforce session timeouts
- Protect against Cross-Site Scripting (XSS)

A3. Cross-Site Scripting (XSS)

What is it?

- Similar to injection, but the intended target is a user's browser: the user's browser is tricked into executing unintended commands
- Usually in the form of HTML code or client-side scripts
- Occurs when an application (client or server-side) takes unstrusted data from the user and uses it without validation or encoding
- Exploits the trust the user has for a site

If an attacker controls your browser, it is no longer *your* browser

A3. Cross-Site Scripting

Why does it happen?

- Failure to separate untrusted data with commands
- Lack of input validation and/or input sanitation
- Attacker is able to change execution context

A3. Cross-Site Scripting

Types

- Reflected
- Stored (aka persistent)
- Dom based

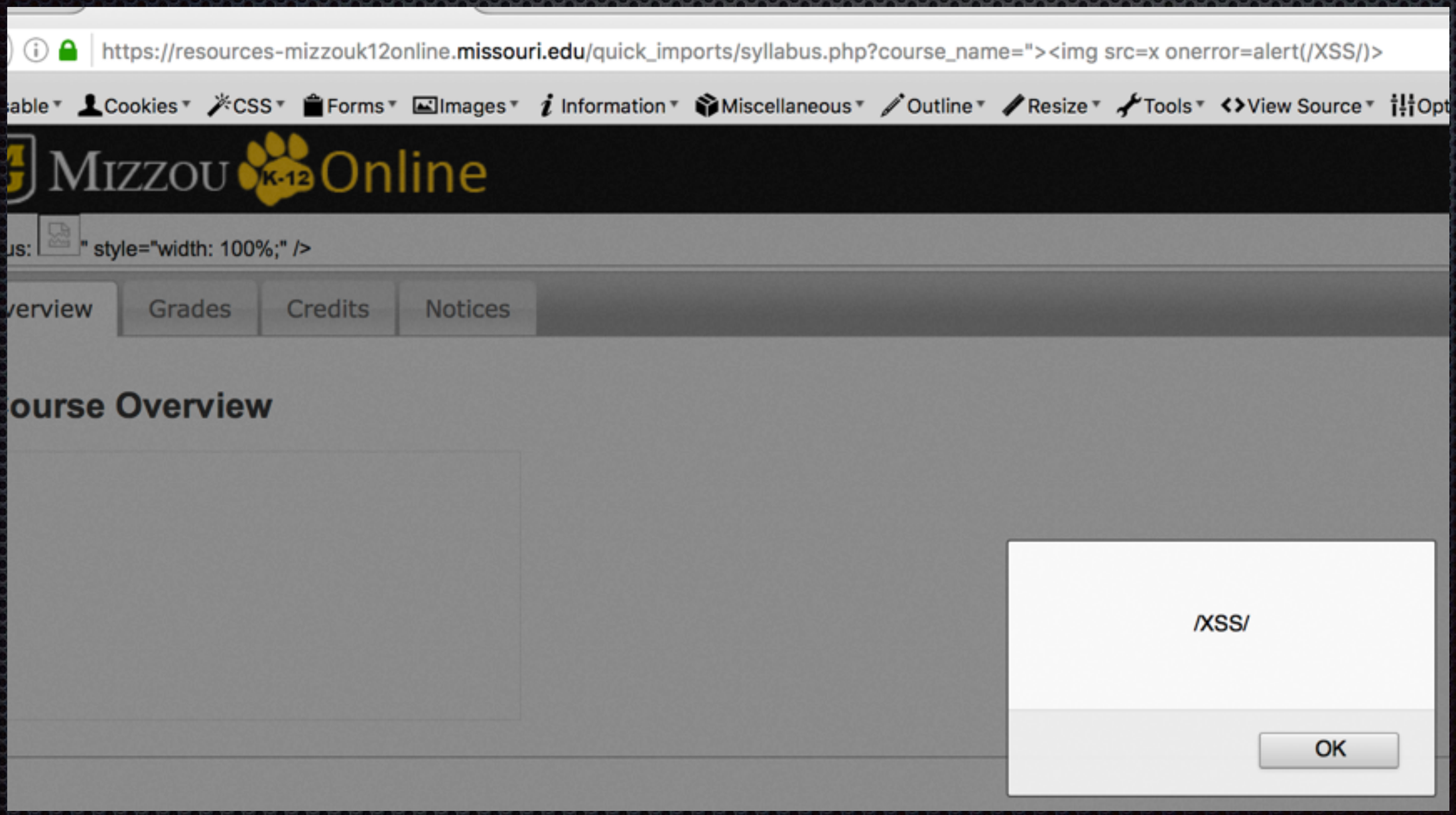
A3. Cross-Site Scripting

Types: Reflected

- Data from the URL is used in the creation of the page
- Attacker tricks/entices user to click the exploited URL
- Web application reflects the payload back to the user's browser where it is executed as if it came from the originating site

A3. Cross-Site Scripting

Types: Reflected Example



A3. Cross-Site Scripting

Types: Stored/Persistent

- Injected script is stored on the target server
- The injection script is then sent to visitors of the site
- Vicitim's browser then executes the injected script as it believes it came from the originating site
- Delayed (aka "Blind") refers to the payload being executed at a later time, possibly in a completely different system (e.g. Log viewers, customer service apps, etc.)

A3. Cross-Site Scripting

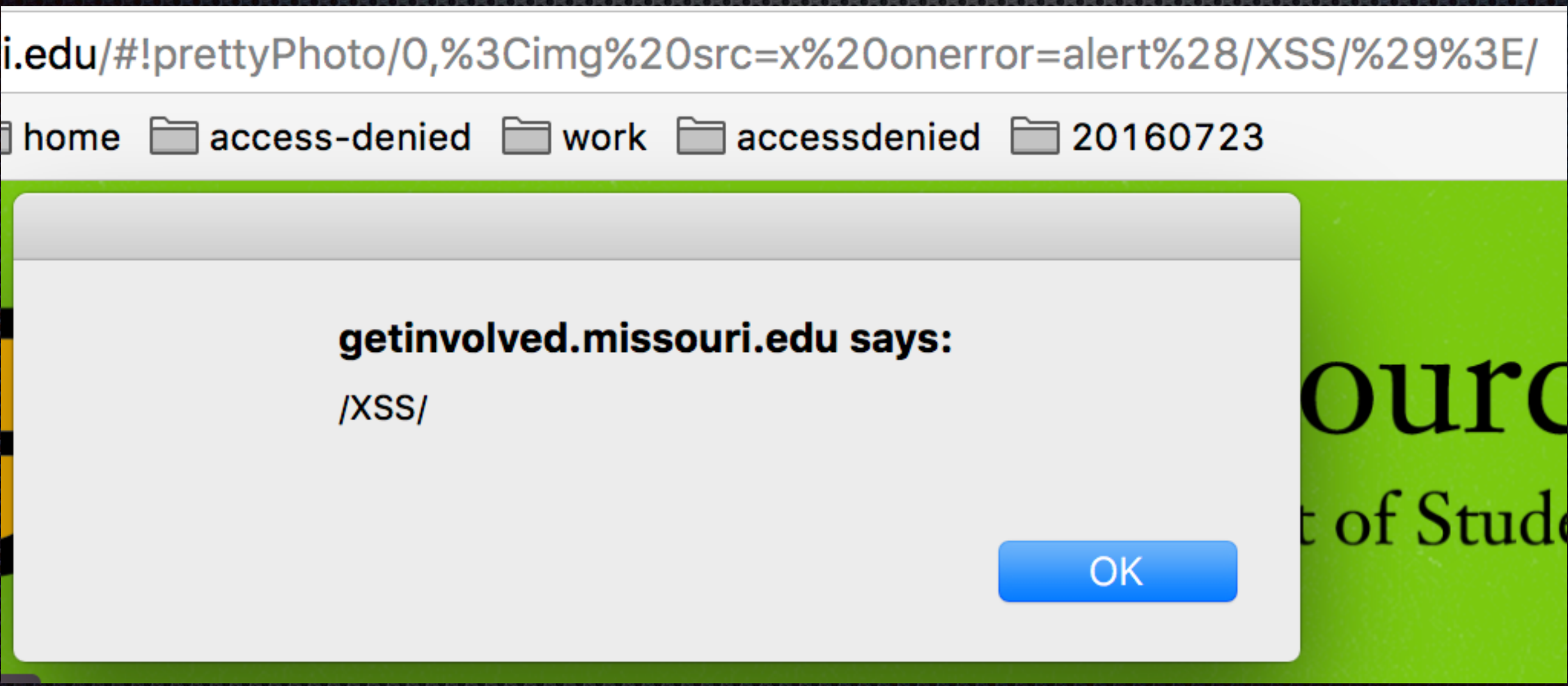
Types: Dom-based (aka Client Side)

- Similar to reflected but instead of tricking the user's browser to exectute script, tricks legitimate client side script into executing the payload
- The attack payload is executed as a result of modifying the DOM environment in the victim's browser used by the original client side script, so that the client side code runs in an unexpected manner
- More prevalent now than in the past due to the increase use in javascript and vulnerable javascript libraries (almost 200% increase between 2015 and 2016[1])

1 - Acunetix web application vulnerability report 2016, pg 5

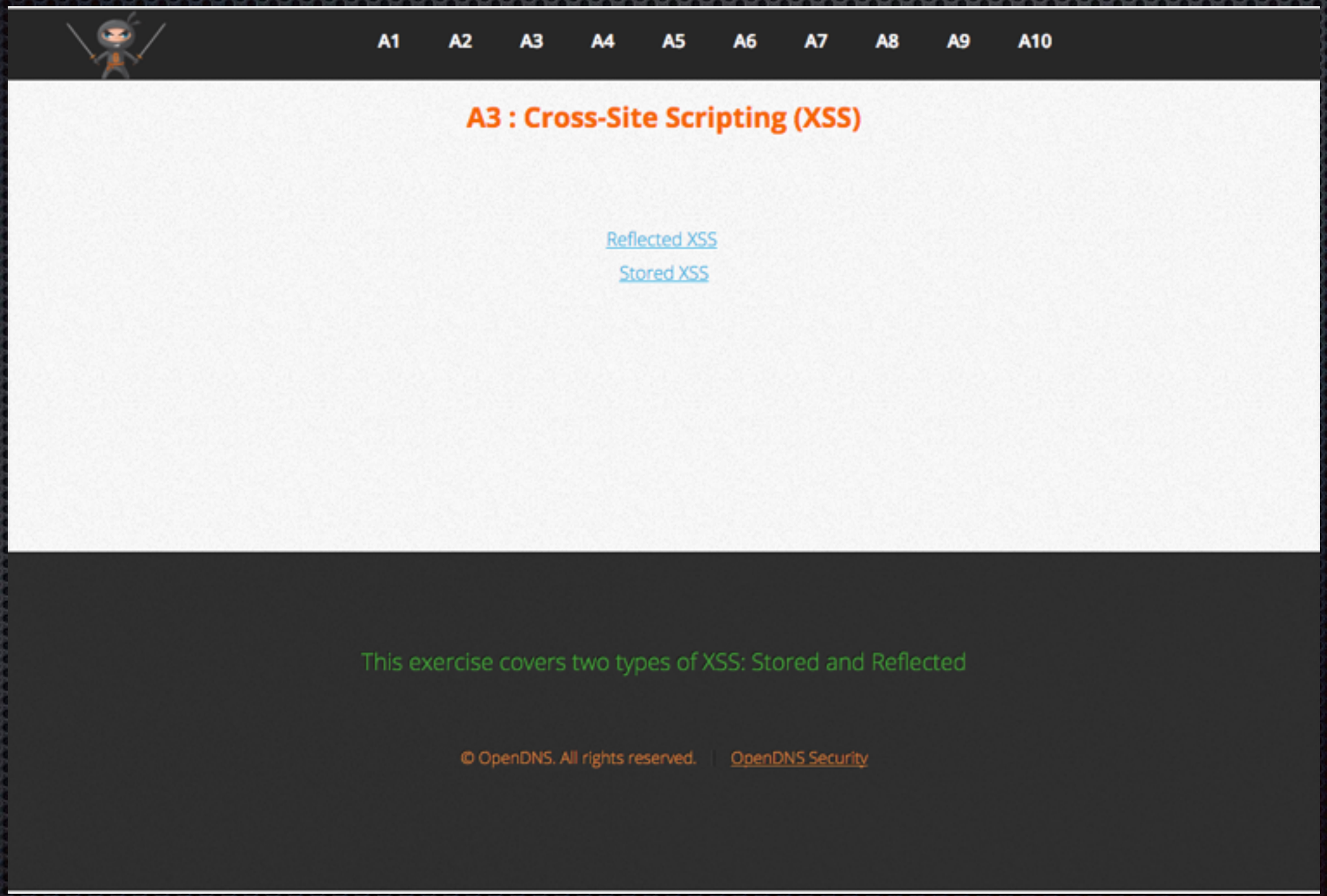
A3. Cross-Site Scripting

Types: Dom-based Example



A3. Cross-Site Scripting

ATTACK!



A3. Cross-Site Scripting

Prevention

- Implicitly deny
- Validate all input, regardless of where it comes from (inbound handling)
- Context-based output encoding (outbound handling)

A3. Cross-Site Scripting

Prevention - Context-based output encoding

- HTML Element Content
  DATA
- HTML Attribute Values
  <input name="DATA" type="text" value="DATA" />
- Javascript Data
  <script>someFunc('DATA');</script>
- CSS Property Values
  .foo a:hover{ color: DATA }
- URI Attribute VAlues
  <a href="/search/?q=DATA">

- (&, <, >, ") —> &entity;
  (', /) —> &#DDD; or &#xHH;
  (UTF-8 encode)
- All non-alphanumeric —> &#xHH;
- All non-alphanumeric —> \xHH
- All non-alphanumeric —> \HH
- All non-alphanumeric —> %HH

A3. Cross-Site Scripting

Prevention

- Implicitly deny
- Validate all input, regardless of where it comes from
- Context-based output encoding
- Content-Security-Policy headers

A3. Cross-Site Scripting

Prevention - Content-Security-Policy headers

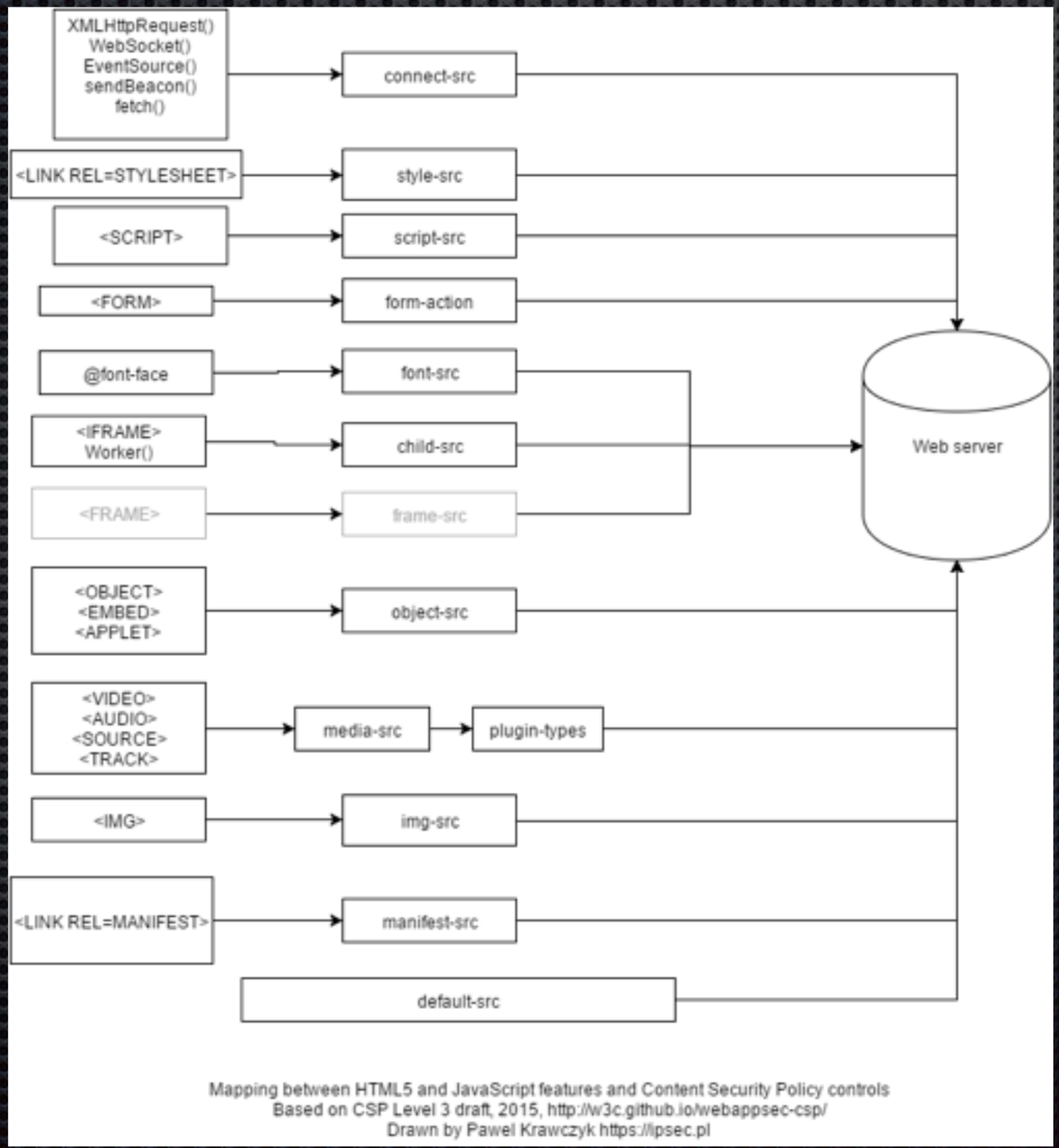
Set of HTTP Response headers (or can be implemented via meta element) that defines where resources can be loaded from, preventing users' browser from loading data from any other locations.

```
Header Set Content-Security-Policy: script-src 'self' https://apis.google.com
```

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self'">
```

A3. Cross-Site Scripting

Prevention - Content-Security-Policy headers



Mapping between HTML5 and JavaScript features and Content Security Policy controls
Based on CSP Level 3 draft, 2015, http://w3c.github.io/webappsec-csp/
Drawn by Pawel Krawczyk https://lipsec.pl

A3. Cross-Site Scripting

Prevention - Content-Security-Policy headers

Content Security Policy 1.0 - CR

Global 85.46% + 5.37% = 90.84%

Mitigate cross-site scripting attacks by whitelisting allowed sources of script, style, and other resources.

Current aligned | Usage relative | Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			49					4.4	
8	13	47	51			9.2		4.4.4	
[1] 11	14	48	52	9.1	39	9.3	all	51	51
		49	53	10	40				
		50	54	TP	41				
		51	55						

Notes | Known issues (3) | Resources (5) | Feedback

The standard HTTP header is `Content-Security-Policy` which is used unless otherwise noted.

[1] Supported through the `X-Content-Security-Policy` header

[2] Supported through the `X-Webkit-CSP` header

http://caniuse.com/#feat=contentsecuritypolicy

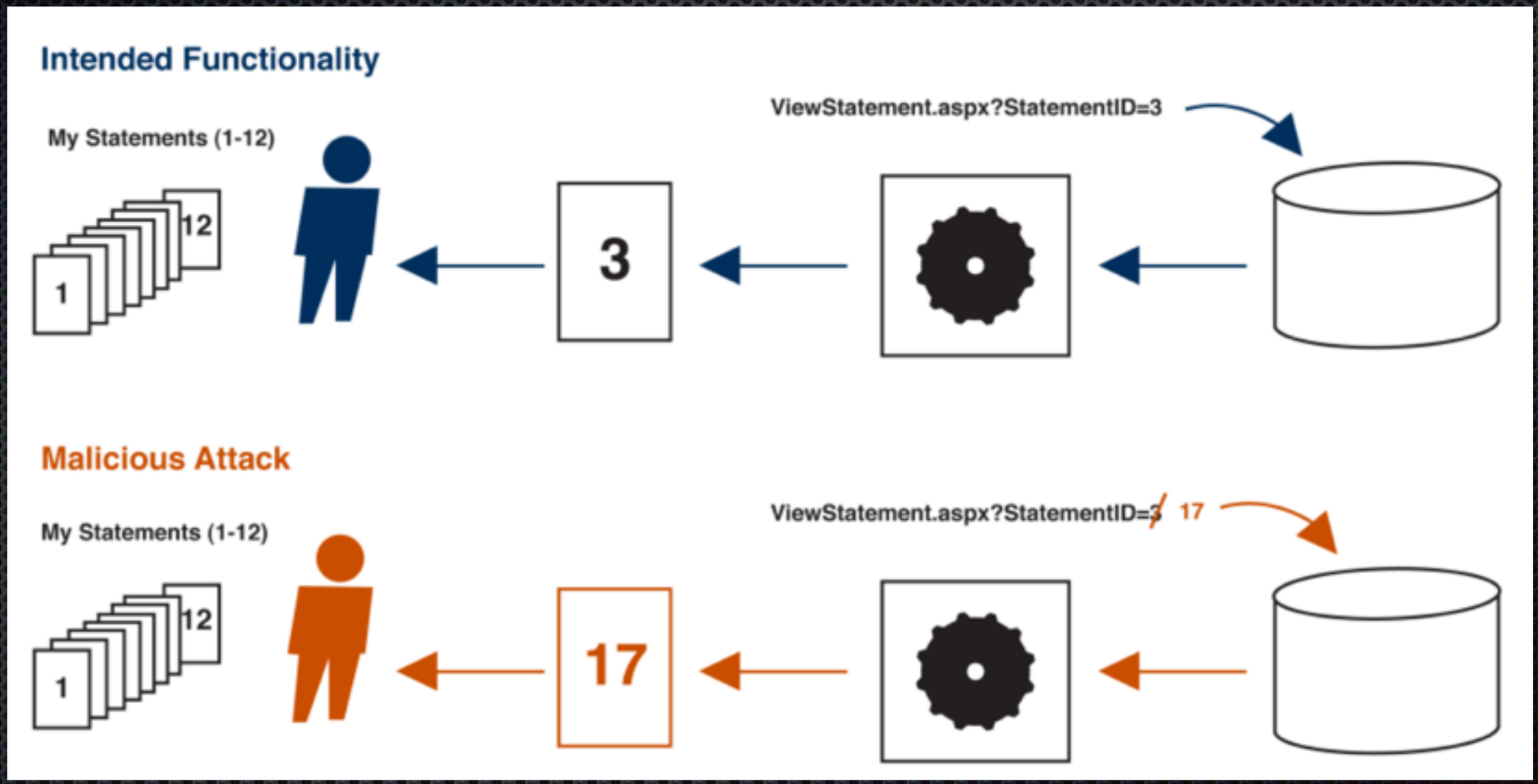
A4. Insecure Direct Object References

What is it?

- Occurs when a reference to an internal implementation object (file, directory, database record/key) is exposed as a URL or form parameter
- Without subsequent authorization checks, an attacker can manipulate these parameters to access unauthorized data

A4. Insecure Direct Object References

Examples



A4. Insecure Direct Object References

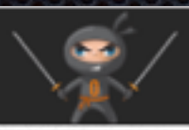
Examples

http://foo.bar.edu/file.php?file=report.txt

http://foo.bar.edu/file.php?file=../../app/config/database

A4. Insecure Direct Object References

ATTACK!



A1 A2 A3 A4 A5 A6 A7 A8 A9 A10

A4 : Insecure Direct Object References

Non-confidential document

Link to a non-confidential document is provided here. Can you find a confidential document on the server?

Hint 1

Hint 2

Solution

 | OpenDNS Security

A4. Insecure Direct Object References

Prevention

- Do not expose internal keys or identifiers for objects
- Use object references that are difficult to guess
- Perform server-side authorization checks for object access

A5. Security Misconfiguration

What is it?

- Security issues anywhere in the stack
  - Framework
  - Application Server
  - Web Server
  - Database Server
  - Operating System
  - 3rd party contols/modules

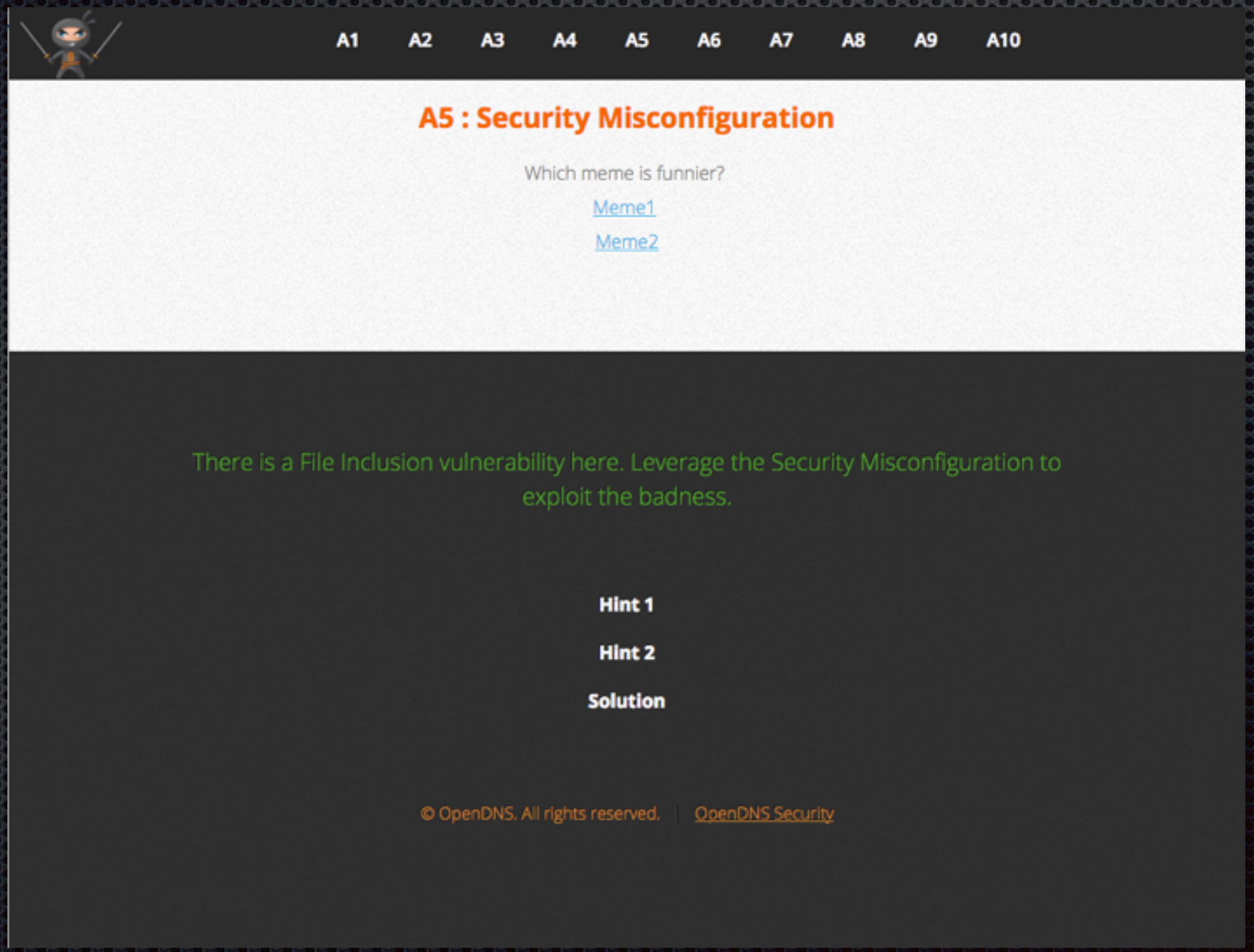
A5. Security Misconfiguration

Examples

- Application dumps stack trace or reveals other debugging information
- Default passwords not changed
- Out-of-date software
- Directory listings enabled

A5. Security Misconfiguration

ATTACK!



A5. Security Misconfiguration

Prevention

- Security hardening throughout the **entire** application stack
- Change default passwords
- Disable/remove debugging message and overly informative error messages in production
- Remove unnecessary/unused features/modules/components

It can’t be exploited if it isn’t there

A6. Sensitive Data Exposure

What is it?

- Storing and transmitting sensitive data insecurely, due to:
  - Failure to identify all sensitive data
  - Failure to identify all locations where sensitive data is stored (database, files, logs, backups, etc)
  - Failure to identify all locations sensitive data is sent (web, databases, internal communications, external business partners)

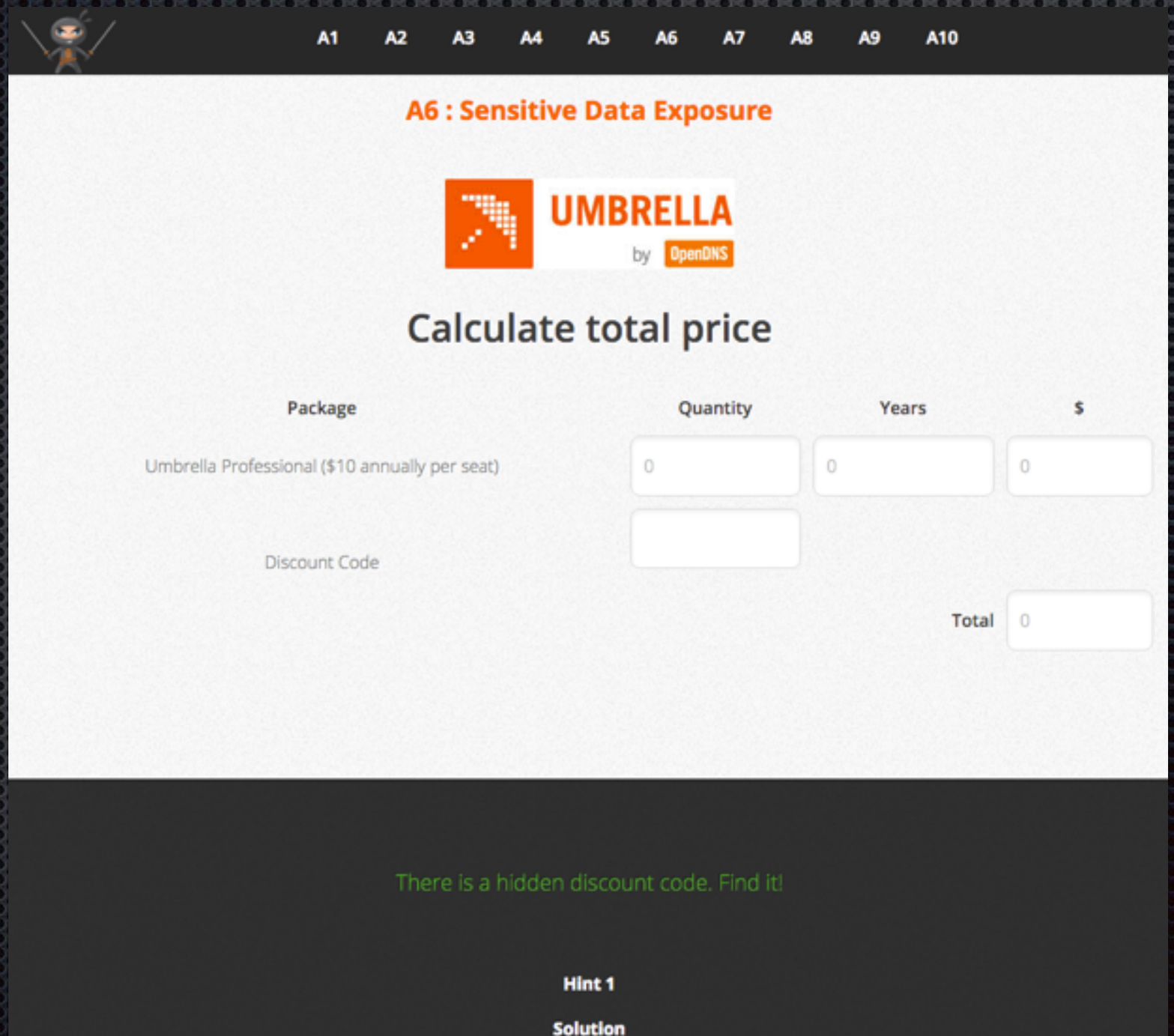
A6. Sensitive Data Exposure

Examples

- Debugging information in comments of ouput sent to the browser
- Application that stores sensitive data (correctly) encoutners an error and logs that data insecurely

A6. Sensitive Data Exposure

ATTACK!



A6. Sensitive Data Exposure

Prevention

Someone can’t steal it if you don’t have it

A6. Sensitive Data Exposure

Prevention

- Don’t store sensitive data if you don’t **need** it
- Catalog all sensitive data and where it is used
- Disable caching of pages that contains sensitive data
- Encrypt all sensitive data at rest and in transit
- Use strong standard algorithms and strong keys

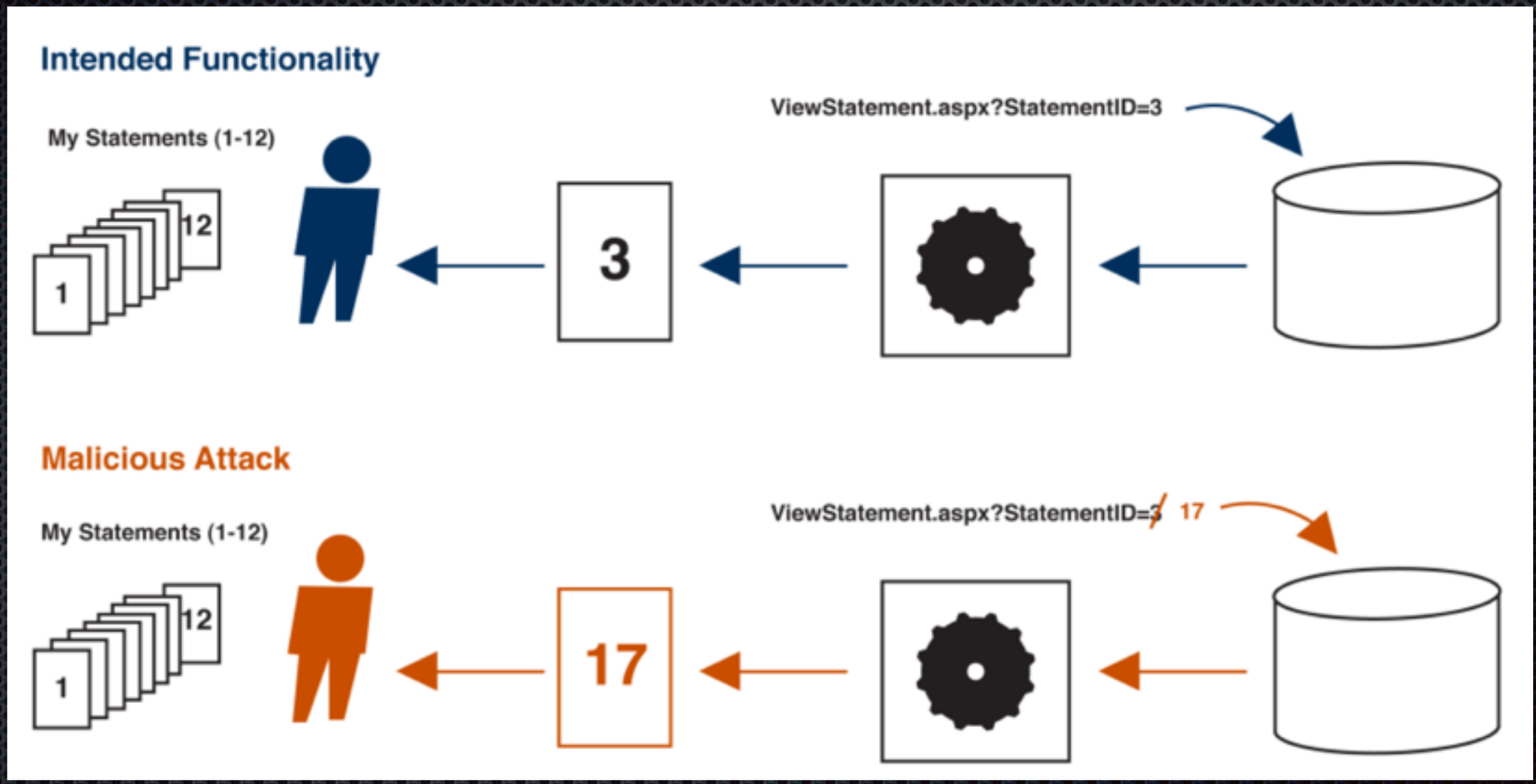
A7. Missing Function Level Access Controls

What is it?

- Enforcing proper authorization for requests

A7. Missing Function Level Access Controls

Examples



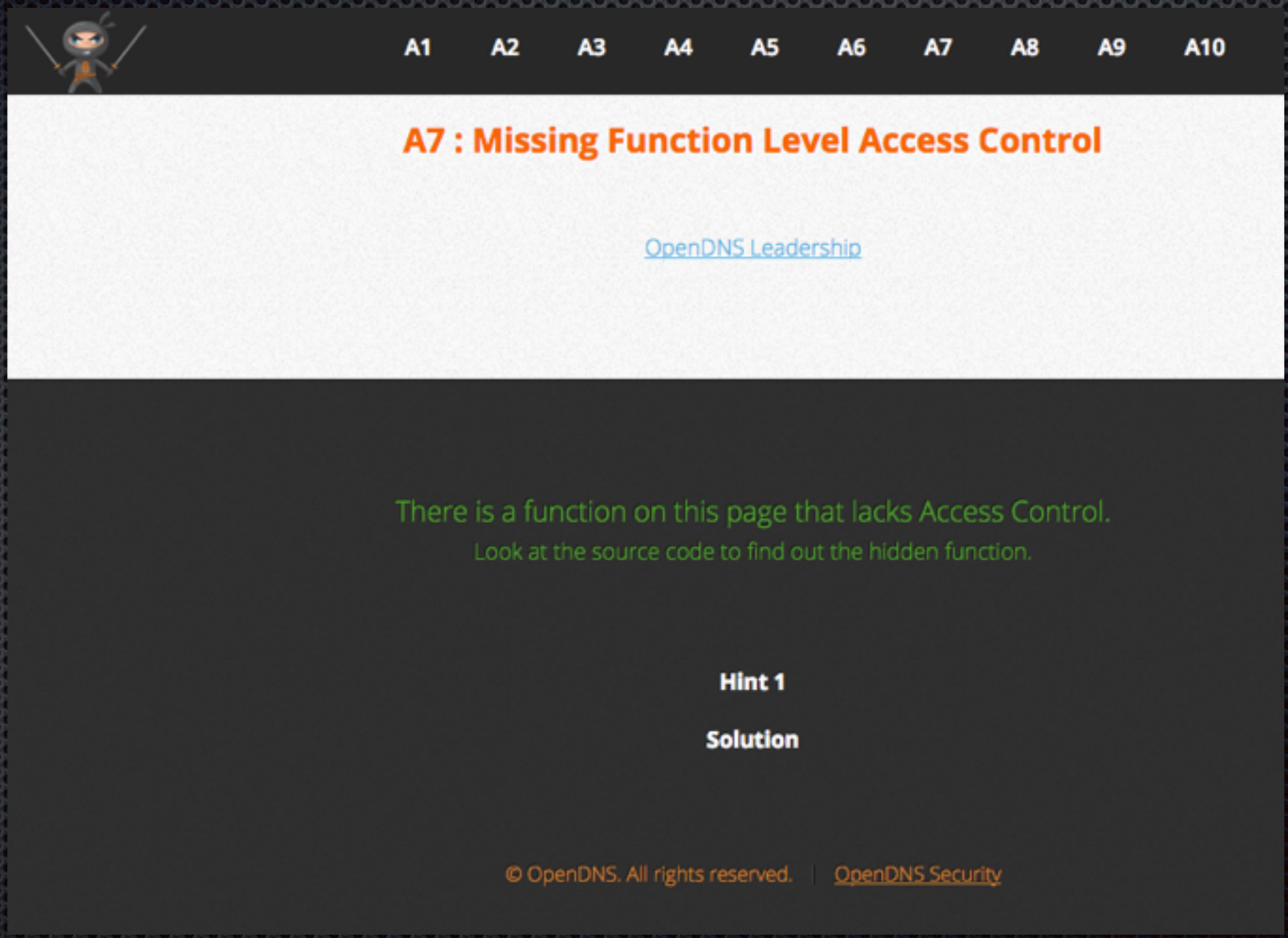
A7. Missing Function Level Access Controls

Examples

- Attacker notices the URL structure indicates his role
  https://example.com/user/getAccount
- Attacker modifies it to another role
  https://example.com/admin/getAccount
- The application fails to do a server-side check to verify the request is coming from a user with the proper authorization and the attacker now has access to data on any account

A7. Missing Function Level Access Controls

ATTACK!



A7. Missing Function Level Access Controls

Prevention

- Deny by default
- Perform server-side checks for **every** function/action that requires authorization
- Server-side checks should not rely soley on information provided by the client

A8. Cross-Site Request Forgery

What is it?

- Victim's browser is tricked into issuing commands to a vulnerable web application, to perform an unwanted action on a trusted site for which the user is currently authenticated.
- Web browsers automatically include credentials (cookies, sessions, basic authentication headers) with each request, even for requests initiated by a form, script or image on another site
- Server-side code processes requests as if they came from the authenticated user
- Often combined with Cross-Site Scripting

A8. Cross-Site Request Forgery

Examples

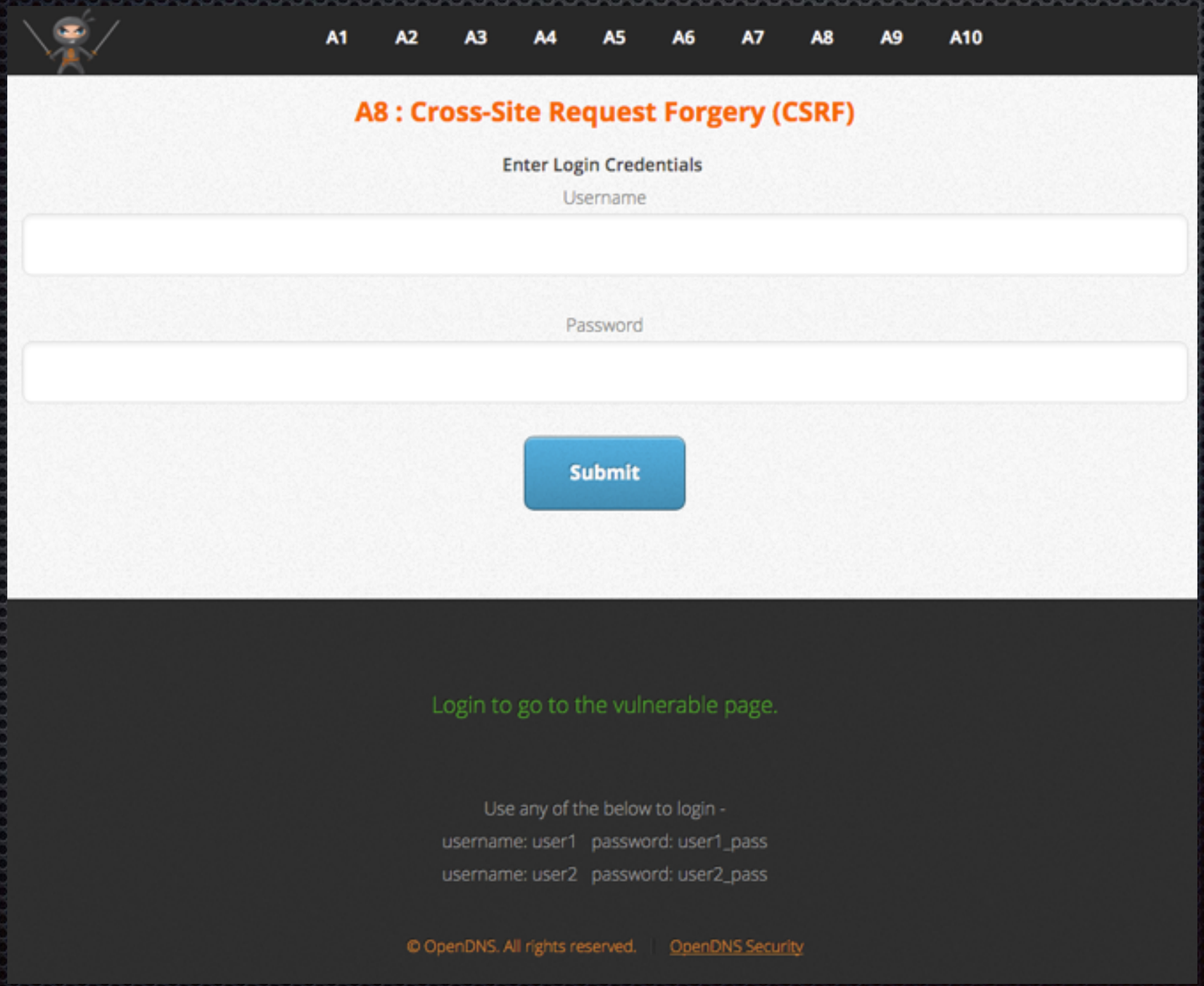
- Application is built to allow a state change via URL parameters

http://bankx.com/app?action=transfer&amount=3500&destinationAccount=4673243243

- Attacker constructs a request that will transfer funds to to their account and embeds the requests on sites he controls. He then tricks the victim into visiting the site

A8. Cross-Site Request Forgery

ATTACK!



A8. Cross-Site Request Forgery

Prevention

- Eliminate all XSS vulnerabilities in your application
- Never allow changes to data via a GET request
- Check Origin and Referer headers as a first step
- Include a per-request, cryptographically strong or random token to all forms, where the verification token is not provided by the browser
- Require reauthentication for sensitive transactions

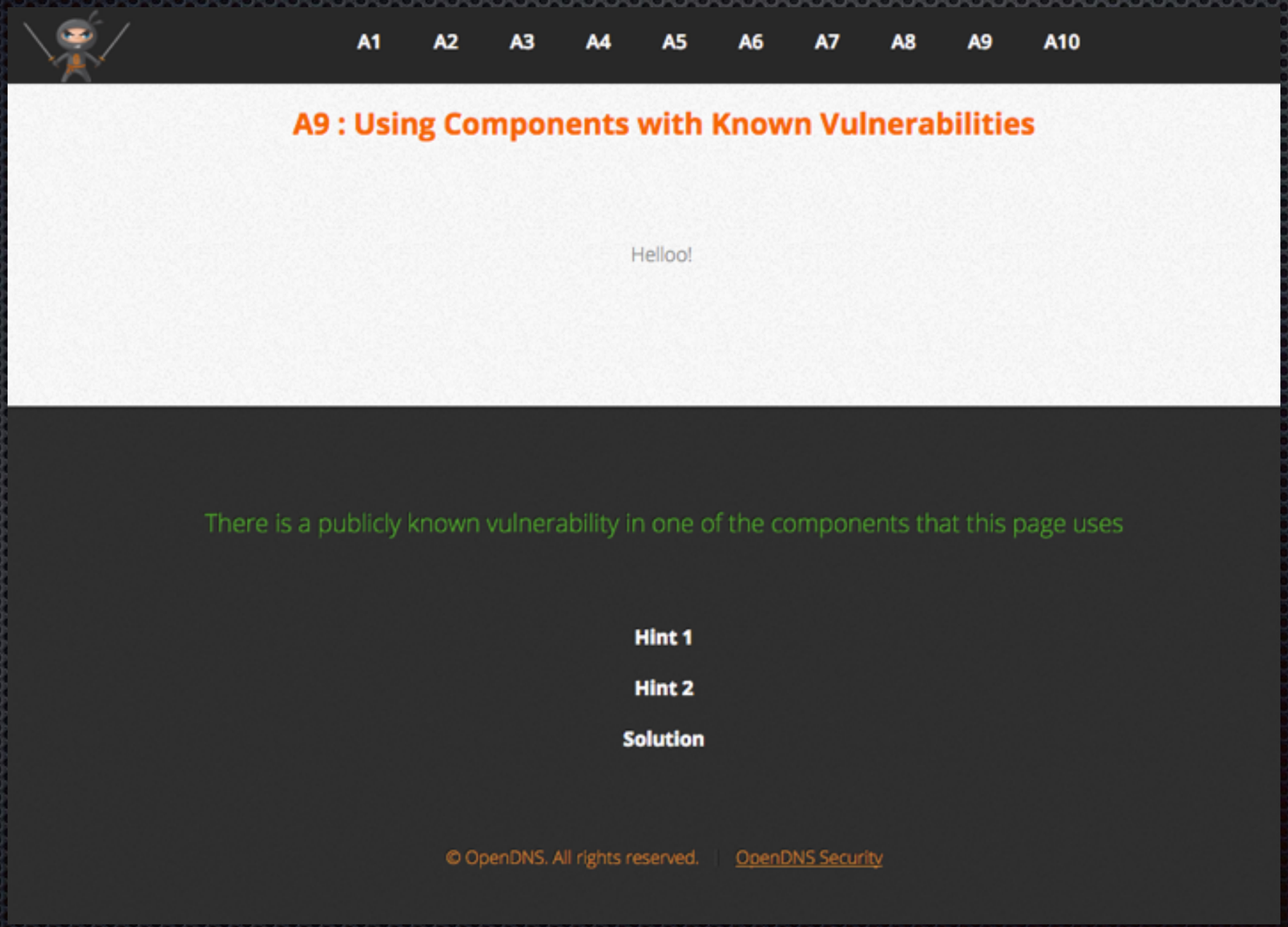
A9. Using Components with Known Vulnerabilities

What is it?

- Using third-party controls/modules/frameworks with known vulnerabilities anywhere in the stack can be discovered and exploited by automated tools
- Third-party libraries/frameworks in an application typically run with the same privileges as the application
- Those vulnerabilities can undermine other defenses

A9. Using Components with Known Vulnerabilities

ATTACK!



A9. Using Components with Known Vulnerabilities

Prevention

- Identify all third-party components in your applications, and potentially throughout the entire stack
- Uninstall/disable all third-party components that are not being used
- Monitor the security of these components in public databases (CVE, NVD, etc.), security mailing lists
- Keep third-party components up-to-date **religiously**
- Develop a security policy on third-party component use

A10. Unvalidated Redirects and Forwards

What is it?

- Application takes input from an external source that is then used to formulate a redirection or forward location without validation

- Attacker links to an unvalidated redirect and tricks victims into clicking it. Victims are more likely to click on it, since the link is to a valid site

  https://www.irs.gov/taxrefund/claim.jsp?year=2015&dest=ev.il/sDfX4

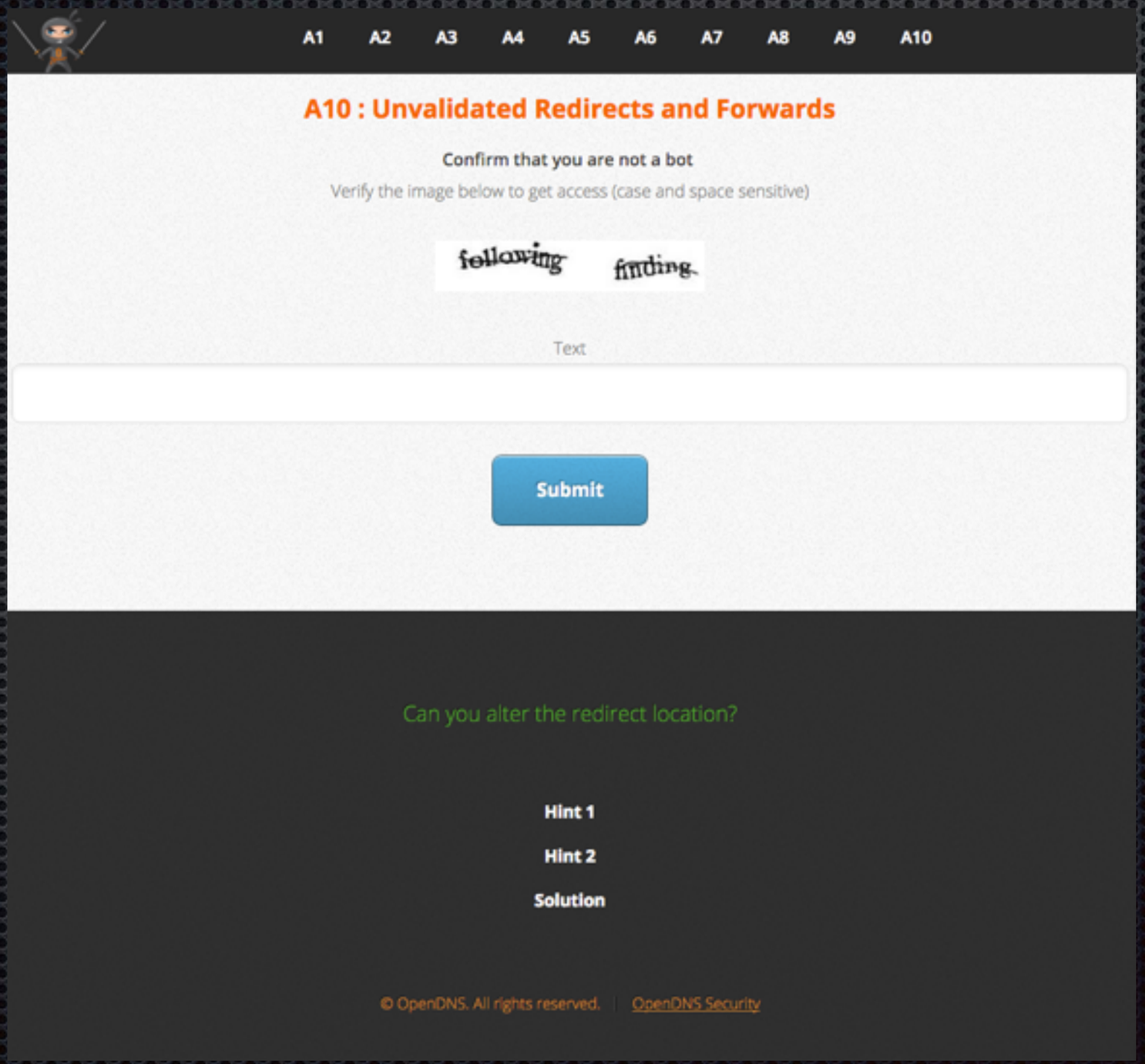
A10. Unvalidated Redirects and Forwards

What is it?

- Application uses forwards to route requests between different parts of the site. Pages use a parameter to indicate where the user should be sent if a transaction is successful.
- Attacker crafts a URL that will pass the application's access control check (or is missing control checks) and then forwards the attacker to administrative functionality for which the attacker isn't authorized.

A10. Unvalidated Redirects and Forwards

ATTACK!



A10. Unvalidated Redirects and Forwards

Prevention

- Don’t use them unless you absolutely have to
- Avoid using user input to detemine destination URL
- Whitelist allowed pages or external sites
- Ensure URL is valid and authorized for the user

Developer Take-aways

Security is a *process*, not an *event*.

Developer Take-aways

- Security is a *process*, not an *event*
- Build security into the software development life-cycle

Be paranoid; be skeptical

Developer Take-aways

- Security is a *process*, not an *event*
- Build security into the software development life-cycle
- Be paranoid; be skeptical
- Treat all data as tainted
  - Validate data first (whitelist), then filter (blacklist)
  - Validate/clean as early in the process as possible
  - Encode data appropriately before sending to the client
  - Never ***rely*** on client-side checks

Developer Take-aways

Continued

- Defense-in-depth
- Principle of Least Privilege
  - Limit access to **everything**
- Enforce strong passwords
- Don’t harcode secrets in source code
- Keep third-party components up-to-date
- Remove overly informative error message and debug information from production

OWASP 2017 RC Changes

- 2013-A4 Insecure Direct Object References and 2013-A7 Missing Function Level Access Control were combined into 2017-A4 Broken Access Control
- Removed 2013-A10 Unvalidated Redirects and Forwards
- Two new “risks” have been introduced
  - Insufficient Attack Detection and Prevention
  - Underprotected APIs

OWASP 2017 Release Candidate

1. Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)
4. Broken Access Controls
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Insufficient Attack Protection
8. Cross-Site Request Forgery (CSRF)
9. Using Components with Known Vulnerabilities
10. Underprotected APIs

Questions?

- Contact
  - gilzow@missouri.edu
  - @gilzow on twitter
- Example Files: https://github.com/opendns/Security_Ninjas_AppSec_Training
- Slidedeck: https://github.com/gilzow/security-ninja/