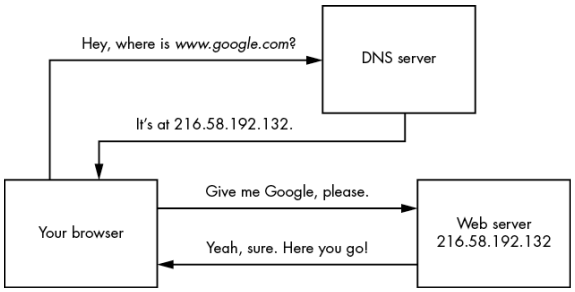
Bug Bounty Bootcamp

The Guide to Finding and Reporting Web Vulnerabilities

by Vickie Li

updated to print 3

Page	Error	Correction	Print corrected
xvi	Mechanisms **Cooking** Sharing	Mechanisms **Cookie** Sharing	Print 2
35	Figure update		Print 2
42	`{ "alg" : "none", "typ" : "JWT" } { "user" : "admin" }`	`{ "alg" : "none", "typ" : "JWT" } { "user_name" : "admin" }`	Print 2

135	`inurl:redirecturi site:example.com`
	`inurl:redirect_uri site:example.com`
	`inurl:redirecturl site:example.com`
	`inurl:redirect_uri site:example.com`
		`inurl:redirecturi site:example.com`
		`inurl:redirect_uri site:example.com`
		`inurl:redirecturl site:example.com`
		`inurl:redirect_url site:example.com`
			Print 2

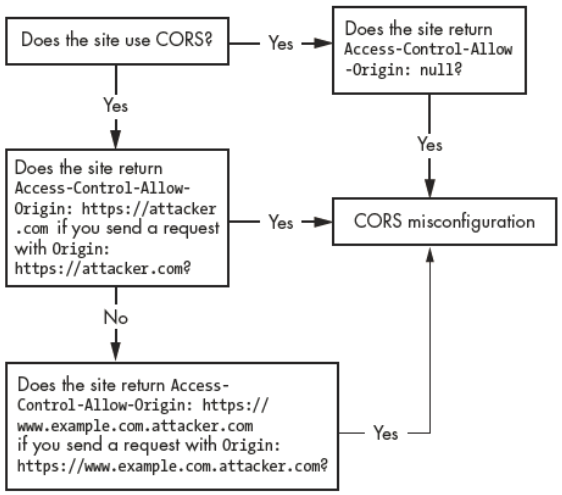
166	`def validate_token():`
	`❶if (request.csrf_token == session.csrf_token):`
	`  pass`
	`  else:`
	`  ❷throw_error("CSRF token incorrect. Request rejected.")`
	`[...]`
	
	`def process_state_changing_action():`
	`  if request.csrf_token:`
	`  validate_token()`
	`❸execute_action()`
	
	This fragment of Python code first checks whether the CSRF token exists ❶. If it exists, the code will proceed to validate the token. If the token is valid, the code will continue. If the token is invalid, the code will stop the execution and produce an error ❷.
		`def validate_token():`
		`  if (request.csrf_token == session.csrf_token):`
		`  pass`
		`  else:`
		`  ❶throw_error("CSRF token incorrect. Request rejected.")`
		`[...]`
		
		`def process_state_changing_action():`
		`❷if request.csrf_token:`
		`  validate_token()`
		`❸execute_action()`
		
		This fragment of Python code first checks whether the CSRF token exists ❷. If it exists, the code will proceed to validate the token. If the token is valid, the code will continue. If the token is invalid, the code will stop the execution and produce an error ❶.
			Print 2

203	URL update	You can find it at *https://github.com/digininja/DVWA/*	Print 2
273	URL update	• CTF Wiki, *https://ctf-wiki.org/pwn/sandbox/python/python-sandbox-escape/*	Print 2

297	```Access-Control-Allow-Origin: b.example.com```
	
	The application can also return the `Access-Control-Allow-Origin` header with a wildcard character (`*`) to indicate that the resource on that page can be accessed by any **domain**:
	
	```Access-Control-Allow-Origin: *```
	
	On the other hand, if the origin of the requesting page isn't allowed to access the resource, the user's browser will block the requesting page from reading the data.
	
	CORS is a great way to implement cross-origin communication. However, CORS is safe only when the list of allowed origins is properly defined. If CORS is misconfigured, attackers can exploit the misconfiguration and access the protected resources.
	
	The most basic misconfiguration of CORS involves allowing the `null` origin. If the server sets `Access-Control-Allow-Origin` to `null`, the browser will allow any site with a `null` origin header to access the resource. This isn't safe because any origin can create a request with a `null` origin. For instance, cross-**site** requests generated from a document using the `data:` URL scheme will have a `null` origin.
		```Access-Control-Allow-Origin: https://b.example.com```
		
		The application can also return the `Access-Control-Allow-Origin` header with a wildcard character (`*`) to indicate that the resource on that page can be accessed by any **origin**:
		
		```Access-Control-Allow-Origin: *```
		
		On the other hand, if the origin of the requesting page isn't allowed to access the resource, the user's browser will block the requesting page from reading the data.
		
		CORS is a great way to implement cross-origin communication. However, CORS is safe only when the list of allowed origins is properly defined. If CORS is misconfigured, attackers can exploit the misconfiguration and access the protected resources.
		
		The most basic misconfiguration of CORS involves allowing the `null` origin. If the server sets `Access-Control-Allow-Origin` to `null`, the browser will allow any site with a `null` origin header to access the resource. This isn't safe because any origin can create a request with a `null` origin. For instance, cross-**origin** requests generated from a document using the `data:` URL scheme will have a `null` origin.
			Print 3

298	An interesting configuration that isn't **exploitable** is setting the allowed origins to the wildcard (`*`). **This isn't exploitable because CORS doesn't allow credentials, including cookies, authentication headers, or client-side certificates, to be sent with requests to these pages. Since credentials cannot be sent in requests to these pages,** no private information can be accessed:	An interesting configuration that isn't **susceptible to information leak** is setting the allowed origins to the wildcard (`*`). **If a client sends a request with credentials to a page with a wildcard `Access-Control-Allow-Originc` header, the browser will raise an error and won't allow the client to read the response, so** no private information can be accessed:	Print 3

304	If not, send a request to the site with the origin header `attacker.com`, and see if the `Access-Control-Allow-Origin` in the response is set to `attacker.com`. (You can add an Origin header by intercepting the request and editing it in a proxy.)
	
	```Origin: attacker.com```
	
	Finally, test whether the site properly validates the origin URL by submitting an Origin header that contains an allowed site, such as *www.example.com.attacker.com*. See if the `Access-Control-Allow-Origin` header returns the origin of the attacker's domain.
	
	```Origin: www.example.com.attacker.com```
		If not, send a request to the site with the origin header **`https://`**`attacker.com`, and see if the `Access-Control-Allow-Origin` in the response is set to **`https://`**`attacker.com`. (You can add an Origin header by intercepting the request and editing it in a proxy.)
		
		```Origin: https://attacker.com```
		
		Finally, test whether the site properly validates the origin URL by submitting an Origin header that contains an allowed site, such as ***https://****www.example.com.attacker.com*. See if the `Access-Control-Allow-Origin` header returns the origin of the attacker's domain.
		
		```Origin: https://www.example.com.attacker.com```
			Print 3

304	Figure update	 *Figure 19-2: Is the site vulnerable to a CORS misconfiguration vulnerability?*	Print 3
308	**Cooking** Sharing	**Cookie** Sharing	Print 2