۱۰ آسیب پذیری API

OWASP API Security Top 10 2023

WWW.SECURECODING.IR

۱۰ آسیب پذیری API بر اساس OWASP API Security Top 10 2023

API1:2023 - Broken Object Level Authorization

این آسیب پذیری به سبب عدم بررسی مجوز دسترسی به منابع و اشیا رخ می دهد و مهاجم به واسطه آن می تواند به منابع و داده های گروه های کاربری و سیستم دسترسی پیدا نماید. بررسی‌های مجوز دسترسی به سطح اشیا باید در هر تابعی که با استفاده از یک شناسه از کاربر به یک منبع داده دسترسی می‌یابد، در نظر گرفته شود.

مثال:

درخواست GET برای دریافت جزئیات یک محصول با شناسه محصول:

```
GET /api/products/{product_id}
```

کد آسیب پذیر(.NET):

```
// Non-compliant code
public class UserController : ApiController
{
    [HttpGet]
    public User GetUser(int userId)
    {
        User user = UserRepository.GetUserById(userId);
        return user;
    }

    [HttpPut]
    public IHttpActionResult UpdateUser(User user)
    {
        UserRepository.UpdateUser(user);
        return Ok();
    }
}
```

پیشگیری (.NET):

```
// Compliant code
public class UserController : ApiController
{
    [HttpGet]
    [Authorize(Roles = "Admin")]
    public User GetUser(int userId)
    {
        User user = UserRepository.GetUserById(userId);
        return user;
```

```
    }

    [HttpPut]
    [Authorize(Roles = "Admin")]
    public IHttpActionResult UpdateUser(User user)
    {
        UserRepository.UpdateUser(user);
        return Ok();
    }
}
```

کد آسیب پذیر (جاوا):

```
// Non-compliant code
@RestController
public class UserController {

    @GetMapping("/users/{userId}")
    public User getUser(@PathVariable int userId) {
        User user = UserRepository.getUserById(userId);
        return user;
    }

    @PutMapping("/users/{userId}")
    public ResponseEntity<?> updateUser(@PathVariable int userId,
@RequestBody User user) {
        UserRepository.updateUser(user);
        return ResponseEntity.ok().build();
    }
}
```

پیشگیری (جاوا):

```
// Compliant code
public class UserController : ApiController
{
    [HttpGet]
    [Authorize(Roles = "Admin")]
    public User GetUser(int userId)
    {
        User user = UserRepository.GetUserById(userId);
        return user;
    }

    [HttpPut]
    [Authorize(Roles = "Admin")]
    public IHttpActionResult UpdateUser(User user)
    {
        UserRepository.UpdateUser(user);
        return Ok();
    }
}
```

پیشنهادات کلی جلوگیری:

در هر تابع که با استفاده از شناسه کاربر به منبع داده دسترسی می‌یابد، بررسی‌های لازم را برای مجوز دسترسی به سطح اشیا در نظر بگیرید. اطمینان حاصل کنید که کاربر مجاز به دسترسی به این منبع است.

از تأیید صحت شناسه‌ها و مجوزها در هر درخواست استفاده کنید. مطمئن شوید که کاربری که اقدام به دسترسی به یک شیء خاص می‌کند، مجاز به انجام این عمل است.

از نحوه ارسال شناسه کاربر در درخواست‌ها محافظت کنید. از استفاده از روش‌های امن برای انتقال و ذخیره شناسه استفاده کنید، مانند استفاده از توکن‌های اعتبارسنجی.

API2:2023 - Broken Authentication

در این آسیب پذیری به سبب مکانیزم های ناکافی امنیتی برای احراز هویت کاربر برای دسترسی منابع، امکان اختلال و دسترسی به اطلاعات محافظت شده توسط مهاجم وجود دارد.

مثال:

درخواست POST برای ورود کاربر با استفاده از اطلاعات ورود:

```
POST /api/login

Body:

{

 "username": "exampleuser",

 "password": "secretpassword"

}
```

کد آسیب پذیر(.NET):

```
// Non-compliant code
[ApiController]
[Route("api/[controller]")]
public class UserController : ControllerBase
{
    [HttpPost]
    public IActionResult Login(string username, string password)
    {
        if (AuthenticateUser(username, password))
        {
            // Generate and return authentication token
            var token = GenerateAuthToken(username);
            return Ok(token);
        }
        else
        {
            return Unauthorized();
        }
    }

    [HttpGet]
    public IActionResult GetUserData(int userId)
    {
        // Retrieve user data from the database
        var userData = Database.GetUserById(userId);

        // Return user data
        return Ok(userData);
    }

    // Other methods...
}
```

پیشگیری (.NET):

```
// Compliant code
[ApiController]
[Route("api/[controller]")]
public class UserController : ControllerBase
{
    private readonly IUserService _userService;
    private readonly IAuthenticationService _authenticationService;

    public UserController(IUserService userService,
IAuthenticationService authenticationService)
    {
        _userService = userService;
        _authenticationService = authenticationService;
    }

    [HttpPost]
    public IActionResult Login(LoginModel loginModel)
    {
        if (_authenticationService.AuthenticateUser(loginModel.Username,
loginModel.Password))
        {
            // Generate and return authentication token
            var token =
_authenticationService.GenerateAuthToken(loginModel.Username);
            return Ok(token);
        }
        else
        {
            return Unauthorized();
        }
    }

    [HttpGet]
    [Authorize]
    public IActionResult GetUserData(int userId)
    {
        // Retrieve the authenticated user's identity
        var identity = HttpContext.User.Identity as ClaimsIdentity;
```

```
        if (identity != null)
        {
            // Get the user ID from the authentication token
            var userIdFromToken = identity.FindFirst("UserId")?.Value;

            if (!string.IsNullOrEmpty(userIdFromToken) &&
userIdFromToken == userId.ToString())
            {
                // Retrieve user data from the database
                var userData = _userService.GetUserData(userId);
                return Ok(userData);
            }
        }

        return Unauthorized();
    }

    // Other methods...
}
```

کد آسیب پذیر (جاوا):

```
@RestController
@RequestMapping("/api/users")
public class UserController {

    @Autowired
    private UserService userService;

    @PostMapping("/login")
    public ResponseEntity<String> login(@RequestParam String username,
@RequestParam String password) {
        if (userService.authenticateUser(username, password)) {
            // Generate and return authentication token
```

```
            String token = generateAuthToken(username);
            return ResponseEntity.ok(token);
        } else {
            return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
        }
    }

    @GetMapping("/{userId}")
    public ResponseEntity<User> getUserData(@PathVariable int userId) {
        // Retrieve user data from the database
        User user = userService.getUserById(userId);

        // Return user data
        return ResponseEntity.ok(user);
    }

    // Other methods...
}
```

پیشگیری (جاوا):

```
@RestController
@RequestMapping("/api/users")
public class UserController {

    @Autowired
    private UserService userService;

    @Autowired
    private AuthenticationService authenticationService;

    @PostMapping("/login")
    public ResponseEntity<String> login(@RequestParam String username,
@RequestParam String password) {
```

```
        if (authenticationService.authenticateUser(username, password))
{
            // Generate and return authentication token
            String token =
authenticationService.generateAuthToken(username);
            return ResponseEntity.ok(token);
        } else {
            return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
        }
    }

    @GetMapping("/{userId}")
    public ResponseEntity<User> getUserData(@PathVariable int userId,
@RequestHeader("Authorization") String authToken) {
        if (authenticationService.validateAuthToken(authToken)) {
            // Retrieve user data from the database
            User user = userService.getUserById(userId);

            // Return user data
            return ResponseEntity.ok(user);
        } else {
            return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
        }
    }

    // Other methods...
}
```

پیشنهادات کلی جلوگیری:

از مکانیزم‌های احراز هویت قوی و استاندارد استفاده کنید مانند JWT (JSON Web Tokens) یا OAuth.

از روش‌های رمزنگاری قوی برای ذخیره و انتقال اطلاعات حساس مانند رمزنگاری اتصال (TLS/SSL) استفاده کنید.

اطلاعات احراز هویت را به درستی بررسی کنید و تأیید کنید که هر درخواست احراز هویت شده از یک کاربر معتبر است.

به دقت بررسی کنید که اطلاعات احراز هویت (مانند رمز عبور) در انتقال یا ذخیره‌سازی داده‌ها در سرور، به صورت امن و رمزنگاری شده باشند.

استفاده از محدودیت تعداد تلاش‌های ناموفق برای ورود و بسته شدن موقت حساب کاربری پس از تعداد مشخصی از تلاش‌های ناموفق.

API3:2023 - Broken Object Property Level Authorization

در این آسیب پذیری مهاجم به واسطه عدم بررسی مدل های داده در درخواست و پاسخ امکان استخراج و یا عملیات CRUD را بر روی متد های مربوطه دارد این موضوع به سبب عدم صحت اعتبارسنجی مجوز دسترسی به ویژگی‌های اشیا و در نتیجه افشا اطلاعات یا اختلال در درخواست ها می گردد.

مثال:

درخواست PUT برای به روزرسانی یک ویژگی از یک مورد:

```
PUT /api/items/{item_id}

Body:

{

 "name": "Updated Item Name",

 "price": 10.99,

 "is_available": true

}
```

کد آسیب پذیر(.NET):

```
[Route("api/items")]
public class ItemController : ControllerBase
{
    private readonly IItemService _itemService;

    public ItemController(IItemService itemService)
    {
        _itemService = itemService;
    }

    [HttpGet("{itemId}")]
    public IActionResult GetItem(int itemId)
    {
        // Retrieve the item from the database
        Item item = _itemService.GetItem(itemId);

        // Return the item without any authorization check
        return Ok(item);
    }

    [HttpPut("{itemId}")]
    public IActionResult UpdateItem(int itemId, [FromBody] Item
updatedItem)
    {
        // Retrieve the existing item from the database
        Item existingItem = _itemService.GetItem(itemId);

        // Update only the allowed properties
        existingItem.Name = updatedItem.Name;
        existingItem.Price = updatedItem.Price;
        existingItem.IsAvailable = updatedItem.IsAvailable;

        // Save the changes to the database
        _itemService.UpdateItem(existingItem);

        // Return a success response
        return Ok();
```

```
    }

    // Other methods...
}
```

پیشگیری (.NET):

```
[Route("api/items")]
public class ItemController : ControllerBase
{
    private readonly IItemService _itemService;

    public ItemController(IItemService itemService)
    {
        _itemService = itemService;
    }

    [HttpGet("{itemId}")]
    public IActionResult GetItem(int itemId)
    {
        // Retrieve the item from the database
        Item item = _itemService.GetItem(itemId);

        // Check if the user is authorized to access the item
        if (!IsUserAuthorized(item))
        {
            return Forbid();
        }

        // Return the item
        return Ok(item);
    }
```

```
    [HttpPut("{itemId}")]
    public IActionResult UpdateItem(int itemId, [FromBody] Item
updatedItem)
    {
        // Retrieve the existing item from the database
        Item existingItem = _itemService.GetItem(itemId);

        // Check if the user is authorized to update the item properties
        if (!IsUserAuthorized
```

کد آسیب پذیر (جاوا):

```
@RestController
@RequestMapping("/api/items")
public class ItemController {

    private final ItemService itemService;

    public ItemController(ItemService itemService) {
        this.itemService = itemService;
    }

    @GetMapping("/{itemId}")
    public Item getItem(@PathVariable int itemId) {
        // Retrieve the item from the database
        Item item = itemService.getItem(itemId);

        // Return the item without any authorization check
        return item;
```

```
    }

    @PutMapping("/{itemId}")
    public void updateItem(@PathVariable int itemId, @RequestBody Item
updatedItem) {
        // Retrieve the existing item from the database
        Item existingItem = itemService.getItem(itemId);

        // Update only the allowed properties
        existingItem.setName(updatedItem.getName());
        existingItem.setPrice(updatedItem.getPrice());
        existingItem.setAvailable(updatedItem.isAvailable());

        // Save the changes to the database
        itemService.updateItem(existingItem);
    }

    // Other methods...
}
```

پیشگیری (جاوا):

```
@RestController
@RequestMapping("/api/items")
public class ItemController {

    private final ItemService itemService;
```

```
    public ItemController(ItemService itemService) {
        this.itemService = itemService;
    }

    @GetMapping("/{itemId}")
    public ResponseEntity<Item> getItem(@PathVariable int itemId) {
        // Retrieve the item from the database
        Item item = itemService.getItem(itemId);

        // Check if the user is authorized to access the item
        if (!isUserAuthorized(item)) {
            return ResponseEntity.status(HttpStatus.FORBIDDEN).build();
        }

        // Return the item
        return ResponseEntity.ok(item);
    }

    @PutMapping("/{itemId}")
    public ResponseEntity<Void> updateItem(@PathVariable int itemId,
@RequestBody Item updatedItem) {
        // Retrieve the existing item from the database
        Item existingItem = itemService.getItem(itemId);

        // Check if the user is authorized to update the item properties
        if (!isUserAuthorized(existingItem)) {
            return ResponseEntity.status(HttpStatus.FORBIDDEN).build();
        }

        // Only update the allowed properties
        existingItem.setName(updatedItem.getName());
        existingItem.setPrice(updatedItem.getPrice());
        existingItem.setAvailable(updatedItem.isAvailable());

        // Save the changes to the database
        itemService.updateItem(existingItem);

        // Return a success response
        return ResponseEntity.ok().build();
    }

    private boolean isUserAuthorized(Item item) {
        // Implement your authorization logic here
        // Check if the user has the necessary permissions to access the
```

```
item
        // Return true if authorized, false otherwise
    }

    // Other methods...
}
```

پیشنهادات کلی جلوگیری:

در هنگام ایجاد یا به روزرسانی اشیا، اطمینان حاصل کنید که مجوز دسترسی به ویژگی‌ها در سطح صحیح تنظیم شده است.

اعتبارسنجی داده‌های ورودی کاربران و فقط قبول دادن آن‌ها در صورتی که دسترسی مجاز به ویژگی‌های مربوطه را دارند.

استفاده از مکانیزم‌های قوی و امن برای تعیین و مدیریت مجوزها و نقش‌ها در سیستم، مانند RBAC (Role-Based Access Control).

محدود کردن دسترسی کاربران به ویژگی‌های اشیا بر اساس نیازهای کسب و کار و اصول حداقل اصول (Principle of Least Privilege).

اجرای آزمون های امنیتی منظم بر روی API ها و سیستم اطمینان حاصل کنید که تمامی مجوزها و اعتبارسنجی‌های مورد نیاز به درستی پیاده سازی شده‌اند.

API4:2023 - Unrestricted Resource Consumption

مهاجم به واسطه این آسیب پذیری می تواند به سبب عدم محدودیت در درخواست دسترسی به منابع امکان اختلال در وضیعت سرویس دهی API و همچنین منجرب به خطاهای عدم منابع کافی برای پردازش گردد.

مثال:

درخواست POST برای ارسال پیامک به شماره موبایل مشخص:

```
POST /api/sms/send

Body:

{

 "phone_number": "1234567890",

 "message": "Hello, this is a test message."

}
```

کد آسیب پذیر(.NET):

```
[ApiController]
[Route("api/resource")]
public class ResourceController : ControllerBase
{
    private readonly ResourceService _resourceService;

    public ResourceController(ResourceService resourceService)
    {
        _resourceService = resourceService;
    }

    [HttpPost]
    public IActionResult ProcessResource(ResourceRequest request)
    {
        // Process the resource request
        string result = _resourceService.Process(request);

        // Return the result
        return Ok(result);
    }

    // Other methods...
}
```

پیشگیری (.NET):

```
[ApiController]
[Route("api/resource")]
public class ResourceController : ControllerBase
{
    private readonly ResourceService _resourceService;

    public ResourceController(ResourceService resourceService)
    {
        _resourceService = resourceService;
    }

    [HttpPost]
    public IActionResult ProcessResource(ResourceRequest request)
    {
        // Validate the resource request
        if (!IsValidRequest(request))
        {
            return BadRequest();
        }

        // Process the resource request with resource consumption limits
        bool success = _resourceService.ProcessWithLimits(request);

        // Check if the resource consumption was successful
        if (!success)
        {
            return StatusCode((int)HttpStatusCode.TooManyRequests);
        }

        // Return the result
        return Ok("Resource processed successfully");
    }

    private bool IsValidRequest(ResourceRequest request)
    {
        // Implement your validation logic here
        // Check if the request is valid
        // Return true if valid, false otherwise
```

```
    }

    // Other methods...
}
```

کد آسیب پذیر (جاوا):

```
@RestController
@RequestMapping("/api")
public class ResourceController {

    private final ResourceService resourceService;

    public ResourceController(ResourceService resourceService) {
        this.resourceService = resourceService;
    }

    @PostMapping("/resource")
    public ResponseEntity<String> processResource(@RequestBody
ResourceRequest request) {
        // Process the resource request
        String result = resourceService.process(request);

        // Return the result
        return ResponseEntity.ok(result);
    }

    // Other methods...
}
```

پیشگیری (جاوا):

```
@RestController
@RequestMapping("/api")
public class ResourceController {

    private final ResourceService resourceService;

    public ResourceController(ResourceService resourceService) {
        this.resourceService = resourceService;
    }

    @PostMapping("/resource")
    public ResponseEntity<String> processResource(@RequestBody
ResourceRequest request) {
        // Validate the resource request
        if (!isValidRequest(request)) {
            return ResponseEntity.badRequest().build();
        }

        // Process the resource request with resource consumption limits
        boolean success = resourceService.processWithLimits(request);

        // Check if the resource consumption was successful
        if (!success) {
            return
ResponseEntity.status(HttpStatus.TOO_MANY_REQUESTS).build();
        }

        // Return the result
        return ResponseEntity.ok("Resource processed successfully");
    }

    private boolean isValidRequest(ResourceRequest request) {
        // Implement your validation logic here
        // Check if the request is valid
        // Return true if valid, false otherwise
```

```
    }

    // Other methods...
}
```

پیشنهادات کلی جلوگیری:

محدودیت منابع مصرفی هر درخواست API، مانند محدودیت پهنای باند، تعداد درخواست‌ها در یک بازه زمانی معین و حداکثر تعداد پیامک‌ها یا تماس‌های تلفنی.

بررسی و اعتبارسنجی درخواست‌های API بر اساس سقف مجاز برای مصرف منابع و اعمال محدودیت‌های لازم.

استفاده از مکانیزم‌های محدودیت ترافیک و کنترل پهنای باند مانند محدودیت‌های پیشرفته شبکه (Advanced Network Limiting)، به منظور کنترل منابع مصرفی توسط هر کاربر یا سرویس.

نظارت و ثبت لاگ‌های مصرف منابع برای تشخیص الگوهای مشکوک و اقدام به اعتبارسنجی دقیق‌تر در صورت لزوم.

اجرای آزمون‌های تحمل بار (Load Testing) و ارزیابی عملکرد منابع سیستم به منظور تشخیص و پیشگیری از مشکلات مصرف منابع نامناسب.

API5:2023 - Broken Function Level Authorization

به واسطه عدم احراز سیاست‌های کنترل دسترسی با سلسله مراتب دسترسی مهاجم امکان فراخوانی و اجرا درخواست های غیرمجاز از Endpoint مجاز برای دسترسی به منابع دیگر کاربران و/یا عملکردهای مدیریتی دسترسی را دارد.

مثال:

درخواست DELETE برای حذف یک نظر با شناسه نظر:

DELETE /api/comments/{comment_id}

کد آسیب پذیر(.NET):

```
[ApiController]
[Route("api/data")]
public class DataController : ControllerBase
{
    private readonly DataService _dataService;

    public DataController(DataService dataService)
    {
        _dataService = dataService;
    }

    [HttpGet]
    public IActionResult GetData()
    {
        // Get data from the service
        var data = _dataService.GetData();

        // Return the data
```

```
        return Ok(data);
    }

    [HttpPost]
    public IActionResult UpdateData(DataModel data)
    {
        // Update the data using the service
        _dataService.UpdateData(data);

        // Return success response
        return Ok("Data updated successfully");
    }

    // Other methods...
}
```

پیشگیری (.NET):

```
[ApiController]
[Route("api/data")]
[Authorize]
public class DataController : ControllerBase
{
    private readonly DataService _dataService;

    public DataController(DataService dataService)
    {
        _dataService = dataService;
    }
```

```
    [HttpGet]
    [Authorize(Roles = "ReadAccess")]
    public IActionResult GetData()
    {
        // Get the user's identity
        var identity = HttpContext.User.Identity as ClaimsIdentity;

        // Get the user's role
        var role = identity.FindFirst(ClaimTypes.Role)?.Value;

        // Check if the user has the required role for reading data
        if (role != "ReadAccess")
        {
            return Forbid(); // Return 403 Forbidden if the user is not
authorized
        }

        // Get data from the service
        var data = _dataService.GetData();

        // Return the data
        return Ok(data);
    }

    [HttpPost]
    [Authorize(Roles = "WriteAccess")]
    public IActionResult UpdateData(DataModel data)
    {
        // Get the user's identity
        var identity = HttpContext.User.Identity as ClaimsIdentity;

        // Get the user's role
        var role = identity.FindFirst(ClaimTypes.Role)?.Value;

        // Check if the user has the required role for updating data
        if (role != "WriteAccess")
        {
            return Forbid(); // Return 403 Forbidden if the user is not
authorized
        }

        // Update the data using the service
        _dataService.UpdateData(data);
```

```
        // Return success response
        return Ok("Data updated successfully");
    }

    // Other methods...
}
```

کد آسیب پذیر (جاوا):

```
@RestController
@RequestMapping("/api/data")
public class DataController {
    private final DataService dataService;

    public DataController(DataService dataService) {
        this.dataService = dataService;
    }

    @GetMapping
    public ResponseEntity<List<Data>> getData() {
        // Get data from the service
        List<Data> data = dataService.getData();

        // Return the data
        return ResponseEntity.ok(data);
    }

    @PostMapping
    public ResponseEntity<String> updateData(@RequestBody Data data) {
        // Update the data using the service
```

```
        dataService.updateData(data);

        // Return success response
        return ResponseEntity.ok("Data updated successfully");
    }

    // Other methods...
}
```

پیشگیری (جاوا):

```
@RestController
@RequestMapping("/api/data")
public class DataController {
    private final DataService dataService;

    public DataController(DataService dataService) {
        this.dataService = dataService;
    }

    @GetMapping
    @PreAuthorize("hasRole('ROLE_READ')")
```

```
    public ResponseEntity<List<Data>> getData() {
        // Get data from the service
        List<Data> data = dataService.getData();

        // Return the data
        return ResponseEntity.ok(data);
    }

    @PostMapping
    @PreAuthorize("hasRole('ROLE_WRITE')")
    public ResponseEntity<String> updateData(@RequestBody Data data) {
        // Update the data using the service
        dataService.updateData(data);

        // Return success response
        return ResponseEntity.ok("Data updated successfully");
    }

    // Other methods...
}
```

پیشنهادات کلی جلوگیری:

اعتبارسنجی کامل در هر عملکرد API بر اساس سطوح دسترسی و نقش کاربران.

استفاده از سیستم‌های مجوز دسترسی چندسطحی و اعمال سطوح دسترسی به منابع مختلف.

جداسازی صحیح بین عملکردهای مدیریتی و عادی و اعمال سیاست‌های دسترسی مناسب برای هر کدام.

بررسی مجوزها در هر عملکرد و اعتبارسنجی دسترسی کاربر در زمان اجرا.

استفاده از فریمورک‌ها و کتابخانه‌های مدیریت دسترسی کاربران و اجرای سیاست‌های دسترسی پیچیده‌تر مانند RBAC (Role-Based Access Control) یا ABAC (Attribute-Based Access Control).

API6:2023 - Unrestricted Access to Sensitive Business Flows

به واسطه این آسیب پذیری مهاجم امکان بهره برداری از عملکرد های مجاز برنامه را برای هدف های غیر مجاز به واسطه قابلیت های برنامه دارد.

مثال:

درخواست POST برای خرید بلیط هواپیما با ارائه جزئیات مسافر:

```
POST /api/tickets/buy

Body:

{

 "passenger_name": "John Doe",

 "flight_number": "AB123",

 "departure_date": "2023-07-01"

}
```

کد آسیب پذیر(.NET):

```
[Route("api/orders")]
public class OrderController : ApiController
{
    private readonly IOrderService _orderService;

    public OrderController(IOrderService orderService)
    {
        _orderService = orderService;
    }

    [HttpPost]
    public IHttpActionResult CreateOrder(OrderRequest request)
    {
        // Create a new order without proper validation
        Order order = _orderService.CreateOrder(request);

        // Return the created order
        return Ok(order);
    }

    [HttpGet]
    [Route("{orderId}")]
    public IHttpActionResult GetOrder(string orderId)
    {
        // Get the order by ID without proper authorization
        Order order = _orderService.GetOrder(orderId);

        // Return the order
        return Ok(order);
    }

    // Other methods...
}
```

پیشگیری (.NET):

```
[Route("api/orders")]
public class OrderController : ApiController
{
    private readonly IOrderService _orderService;

    public OrderController(IOrderService orderService)
    {
        _orderService = orderService;
    }

    [HttpPost]
    [Authorize(Roles = "Admin")]
    public IHttpActionResult CreateOrder(OrderRequest request)
    {
        // Validate the request and create a new order with proper
authorization
        Order order = _orderService.CreateOrder(request);

        // Return the created order
        return Ok(order);
    }

    [HttpGet]
    [Route("{orderId}")]
    [Authorize(Roles = "User")]
    public IHttpActionResult GetOrder(string orderId)
    {
        // Authorize the user's access to the order
        // Only users with the "User" role can access the order
        Order order = _orderService.GetOrder(orderId);

        // Return the order
        return Ok(order);
```

```
    }

    // Other methods...
}
```

کد آسیب پذیر (جاوا):

```
@RestController
@RequestMapping("/api/orders")
public class OrderController {
    private final OrderService orderService;

    public OrderController(OrderService orderService) {
        this.orderService = orderService;
    }

    @PostMapping
    public ResponseEntity<Order> createOrder(@RequestBody OrderRequest
request) {
        // Create a new order without proper validation
        Order order = orderService.createOrder(request);

        // Return the created order
        return ResponseEntity.ok(order);
    }

    @GetMapping("/{orderId}")
    public ResponseEntity<Order> getOrder(@PathVariable String orderId)
{
        // Get the order by ID without proper authorization
```

```
        Order order = orderService.getOrder(orderId);

        // Return the order
        return ResponseEntity.ok(order);
    }

    // Other methods...
}
```

پیشگیری (جاوا):

```
@RestController
@RequestMapping("/api/orders")
public class OrderController {
    private final OrderService orderService;

    public OrderController(OrderService orderService) {
        this.orderService = orderService;
    }

    @PostMapping
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public ResponseEntity<Order> createOrder(@RequestBody OrderRequest
request) {
        // Validate the request and create a new order with proper
authorization
        Order order = orderService.createOrder(request);

        // Return the created order
        return ResponseEntity.ok(order);
```

```
    }

    @GetMapping("/{orderId}")
    @PreAuthorize("hasRole('ROLE_USER') or hasPermission(#orderId,
'READ')")
    public ResponseEntity<Order> getOrder(@PathVariable String orderId)
{
        // Authorize the user's access to the order
        // Only users with ROLE_USER or permission to read the order can
access it
        Order order = orderService.getOrder(orderId);

        // Return the order
        return ResponseEntity.ok(order);
    }

    // Other methods...
}
```

پیشنهادات کلی جلوگیری:

اجرای مکانیزم‌های احراز هویت و اعتبارسنجی کاربران قبل از دسترسی به جریان کسب و کار حساس.

بررسی و اعتبارسنجی داده‌ها و ورودی‌های کاربران به دقت، از جمله بررسی اعتبار تاریخ و فرمت‌های ورودی.

اعمال محدودیت‌ها و قوانین منطقی برای دسترسی به جریان کسب و کار حساس.

استفاده از سیستم‌های لاگینگ و مانیتورینگ برای آشکارسازی و پیگیری فعالیت‌های مشکوک یا نامناسب در جریان‌های کسب و کار.

ارائه و استفاده از واسطه‌ها (API Gateways) که قابلیت کنترل و مدیریت دسترسی به جریان‌های کسب و کار را فراهم می‌کنند.

API7:2023 - Server Side Request Forgery

مهاجم به واسطه این آسیب پذیری امکان جعل درخواست در سمت سرور و ارسال درخواست های تقلبی را به مقصد مجاز را دارد.

مثال:

درخواست GET برای دریافت تصویر از یک URL مشخص:

GET /api/image?url=http://malicious-website.com/malware.jpg

کد آسیب پذیر(.NET):

```
[Route("api/images")]
public class ImageController : ApiController
{
    [HttpGet]
    public IHttpActionResult GetImage(string url)
    {
        // Fetch the image from the specified URL without proper
validation
        using (WebClient client = new WebClient())
        {
            byte[] imageData = client.DownloadData(url);
            return File(imageData, "image/jpeg");
        }
    }

    // Other methods...
}
```

پیشگیری (.NET):

```
[Route("api/images")]
public class ImageController : ApiController
{
    [HttpGet]
    public IHttpActionResult GetImage(string url)
    {
        // Validate and sanitize the URL before fetching the image
        if (!IsValidUrl(url))
        {
            return BadRequest("Invalid URL");
        }

        using (WebClient client = new WebClient())
        {
            byte[] imageData = client.DownloadData(url);
            return File(imageData, "image/jpeg");
        }
    }

    private bool IsValidUrl(string url)
    {
        // Implement URL validation logic here (e.g., whitelist trusted
domains)
        // Return true if the URL is valid, otherwise false
        // Example validation logic:
        return url.StartsWith("http://trusted-domain.com");
    }

    // Other methods...
}
```

کد آسیب پذیر (جاوا):

```
@RestController
@RequestMapping("/api/images")
public class ImageController {

    @GetMapping
    public ResponseEntity<byte[]> getImage(@RequestParam("url") String
url) throws IOException {
        // Fetch the image from the specified URL without proper
validation
        URL imageUrl = new URL(url);
        byte[] imageData = IOUtils.toByteArray(imageUrl);
        return
ResponseEntity.ok().contentType(MediaType.IMAGE_JPEG).body(imageData);
    }

    // Other methods...
}
```

پیشگیری (جاوا):

```
@RestController
@RequestMapping("/api/images")
```

```
public class ImageController {

    @GetMapping
    public ResponseEntity<byte[]> getImage(@RequestParam("url") String
url) throws IOException {
        // Validate and sanitize the URL before fetching the image
        if (!isValidUrl(url)) {
            return ResponseEntity.badRequest().build();
        }

        URL imageUrl = new URL(url);
        byte[] imageData = IOUtils.toByteArray(imageUrl);
        return
ResponseEntity.ok().contentType(MediaType.IMAGE_JPEG).body(imageData);
    }

    private boolean isValidUrl(String url) {
        // Implement URL validation logic here (e.g., whitelist trusted
domains)
        // Return true if the URL is valid, otherwise false
        // Example validation logic:
        return url.startsWith("http://trusted-domain.com");
    }

    // Other methods...
}
```

پیشنهادات کلی جلوگیری:

قبل از ارسال درخواست به یک URL مشخص، بررسی و اعتبارسنجی دقیق URI و منبع مقصد.

محدود کردن قابلیت دریافت اطلاعات از منابع خارجی و محدود کردن لیست دسترسی‌های مجاز به URL های راه دور.

استفاده از Whitelist برای نشان دادن تنها آدرس های معتبر و اجازه دسترسی به آنها.

اعتبارسنجی و فیلتر کردن ورودی کاربران و پارامترهای مرتبط با URL مورد استفاده قبل از استفاده از آنها در درخواست.

استفاده از محدودیت‌های شبکه، مانند فایروال، برای محدود کردن دسترسی به منابع خارجی.

آموزش تیم توسعه برای ارزیابی و اعتبارسنجی صحیح URI قبل از استفاده از آن در درخواست ها.

API8:2023 - Security Misconfiguration

به واسطه پیکربندی های نادرست و یا عدم مدیریت صحیح تنظیمات مربوط به پیکربندی، امکان بهره برداری مهاجم از تنظیمات پیش فرض و یا نادرست وجود دارد.

مثال:

درخواست GET برای دریافت تنظیمات سیستم:

GET /api/configurations

کد آسیب پذیر(.NET):

```
using System.Web.Http;
```

```
namespace MyAPI.Controllers
{
    public class UserController : ApiController
    {
        // GET api/user/{id}
        public IHttpActionResult GetUser(int id)
        {
            // Fetch user data from the database without proper access
control
            var user = Database.GetUser(id);
            return Ok(user);
        }

        // Other methods...
    }
}
```

پیشگیری (.NET):

```
using System.Web.Http;
using Microsoft.AspNetCore.Authorization;

namespace MyAPI.Controllers
{
    [Authorize] // Apply authorization to the controller
    public class UserController : ApiController
    {
        // GET api/user/{id}
```

```
        [Authorize(Roles = "Admin")] // Restrict access to authorized
users with the "Admin" role
        public IHttpActionResult GetUser(int id)
        {
            // Fetch user data from the database only if the user has
the "Admin" role
            var user = Database.GetUser(id);
            return Ok(user);
        }

        // Other methods...
    }
}
```

کد آسیب پذیر (جاوا):

```
@RestController
public class UserController {

    @Autowired
    private UserRepository userRepository;

    // GET /user/{id}
    @RequestMapping(value = "/user/{id}", method = RequestMethod.GET)
    public User getUser(@PathVariable int id) {
        // Fetch user data from the database without proper access
control
        User user = userRepository.findById(id);
        return user;
    }

    // Other methods...
```

```
}
```

پیشگیری (جاوا):

```
@RestController
public class UserController {

    @Autowired
    private UserRepository userRepository;

    // GET /user/{id}
    @PreAuthorize("hasRole('ADMIN')") // Restrict access to users with
the "ADMIN" role
    @RequestMapping(value = "/user/{id}", method = RequestMethod.GET)
    public User getUser(@PathVariable int id) {
        // Fetch user data from the database only if the user has the
"ADMIN" role
        User user = userRepository.findById(id);
        return user;
    }

    // Other methods...
}
```

پیشنهادات کلی جلوگیری:

قبل از ارسال درخواست به یک URL مشخص، بررسی و اعتبارسنجی دقیق URI و منبع مقصد.

محدود کردن قابلیت دریافت اطلاعات از منابع خارجی و محدود کردن لیست دسترسی‌های مجاز به URL های راه دور.

استفاده از Whitelist برای نشان دادن تنها آدرس های معتبر و اجازه دسترسی به آنها.

اعتبارسنجی و فیلتر کردن ورودی کاربران و پارامترهای مرتبط با URL مورد استفاده قبل از استفاده از آنها در درخواست.

استفاده از محدودیت‌های شبکه، مانند فایروال، برای محدود کردن دسترسی به منابع خارجی.

آموزش تیم توسعه برای ارزیابی و اعتبارسنجی صحیح URI قبل از استفاده از آن در درخواست ها.

API9:2023 - Improper Inventory Management

به واسطه عدم مدیریت نسخه های API ها و همچنین لیست قابلیت ها و عملکرد های نمونه موردی برای تمام توابع، امکان بهره برداری مهاجم از عملکرد های متفاوت در نسخه های گوناگون اپلیکیشن وجود دارد.

مثال:

درخواست GET برای دریافت لیست نسخه‌های موجود API:

```
GET /api/versions
```

کد آسیب پذیر(.NET):

```
[ApiController]
[Route("api/inventory")]
public class InventoryController : ControllerBase
{
    private readonly IInventoryService _inventoryService;

    public InventoryController(IInventoryService inventoryService)
    {
        _inventoryService = inventoryService;
    }

    // GET api/inventory/{productId}
    [HttpGet("{productId}")]
    public IActionResult GetProductInventory(int productId)
    {
        // Fetch inventory data directly from the database
        var inventory =
_inventoryService.GetInventoryByProductId(productId);
        return Ok(inventory);
    }

    // POST api/inventory
    [HttpPost]
    public IActionResult UpdateProductInventory(InventoryModel
inventory)
    {
        // Update inventory directly in the database
        _inventoryService.UpdateInventory(inventory);
        return Ok();
    }

    // Other methods...
}
```

پیشگیری (.NET):

```
[ApiController]
[Route("api/inventory")]
public class InventoryController : ControllerBase
{
    private readonly IInventoryService _inventoryService;

    public InventoryController(IInventoryService inventoryService)
    {
        _inventoryService = inventoryService;
    }

    // GET api/inventory/{productId}
    [HttpGet("{productId}")]
    public IActionResult GetProductInventory(int productId)
    {
        // Fetch inventory data through the inventory service
        var inventory =
_inventoryService.GetProductInventory(productId);

        if (inventory == null)
            return NotFound();

        return Ok(inventory);
    }

    // POST api/inventory
    [HttpPost]
    [Authorize(Roles = "Admin")] // Restrict access to authorized users
with the "Admin" role
    public IActionResult UpdateProductInventory(InventoryModel
inventory)
    {
        // Update inventory through the inventory service
        _inventoryService.UpdateProductInventory(inventory);
        return Ok();
    }

    // Other methods...
}
```

کد آسیب پذیر (جاوا):

```
@RestController
@RequestMapping("/api/inventory")
public class InventoryController {

    private final InventoryService inventoryService;

    public InventoryController(InventoryService inventoryService) {
        this.inventoryService = inventoryService;
    }

    // GET /api/inventory/{productId}
    @GetMapping("/{productId}")
    public ResponseEntity<Inventory> getProductInventory(@PathVariable
int productId) {
        // Fetch inventory data directly from the database
        Inventory inventory =
inventoryService.getInventoryByProductId(productId);
        return ResponseEntity.ok(inventory);
    }

    // POST /api/inventory
    @PostMapping
    public ResponseEntity<?> updateProductInventory(@RequestBody
Inventory inventory) {
        // Update inventory directly in the database
        inventoryService.updateInventory(inventory);
        return ResponseEntity.ok().build();
    }

    // Other methods...
}
```

پیشگیری (جاوا):

```
@RestController
@RequestMapping("/api/inventory")
public class InventoryController {

    private final InventoryService inventoryService;

    public InventoryController(InventoryService inventoryService) {
        this.inventoryService = inventoryService;
    }

    // GET /api/inventory/{productId}
    @GetMapping("/{productId}")
    public ResponseEntity<Inventory> getProductInventory(@PathVariable
int productId) {
        // Fetch inventory data through the inventory service
        Inventory inventory =
inventoryService.getProductInventory(productId);

        if (inventory == null) {
            return ResponseEntity.notFound().build();
        }

        return ResponseEntity.ok(inventory);
    }

    // POST /api/inventory
    @PostMapping
    @PreAuthorize("hasRole('ADMIN')") // Restrict access to authorized
users with the "ADMIN" role
    public ResponseEntity<?> updateProductInventory(@RequestBody
Inventory inventory) {
        // Update inventory through the inventory service
        inventoryService.updateProductInventory(inventory);
        return ResponseEntity.ok().build();
    }

    // Other methods...
}
```

پیشنهادات کلی جلوگیری:

مستندسازی کامل و دقیق برای API، شامل نسخه های موجود و نسخه های قبلی.

ایجاد یک سیستم مدیریت نسخه که به‌روز رسانی و مدیریت نسخه‌های API را ساده می‌کند.

معرفی یک سیاست ازادسازی نسخه که شامل دوره زمانی و پشتیبانی نسخه‌های قدیمی شده باشد.

استفاده از روش‌های خودکار برای بررسی نسخه API مورد استفاده توسط مشتریان و هشدار دادن در صورت استفاده از نسخه‌های قدیمی.

مانیتورینگ و پایش مداوم برای تشخیص و رفع مشکلاتی مانند نسخه‌های قدیمی شده API و نقاط پایانی اشکال‌زده.

استفاده از روش‌های اتوماسیون برای بررسی و به‌روزرسانی خودکار نسخه‌های API و میزبان‌ها.

تعیین سیاست های به‌روزرسانی برای نسخه های API قدیمی و عدم پشتیبانی از آنها.

API10:2023 - Unsafe Consumption of APIs

به واسطه آسیب پذیری فوق مهاجم قادر است با ارسال و یا دریافت اطلاعات از منابع زنجیره تامین، اطلاعات و یا درخواست های مورد نظر خود را در گروه مشخص اجرا نماید.

مثال:

درخواست GET برای دریافت اطلاعات هواشناسی از سرویس شخص ثالث:

```
GET /api/weather?location=New+York
```

کد آسیب پذیر(.NET):

```
[ApiController]
[Route("api/weather")]
public class WeatherController : ControllerBase
{
    private readonly IWeatherService weatherService;

    public WeatherController(IWeatherService weatherService)
    {
        this.weatherService = weatherService;
    }

    // GET /api/weather
    [HttpGet]
    public IActionResult GetWeather(string location)
    {
        // Make a direct call to the third-party weather API
        WeatherData weatherData =
weatherService.GetWeatherData(location);
        return Ok(weatherData);
    }

    // Other methods...
}
```

```
[ApiController]
[Route("api/weather")]
public class WeatherController : ControllerBase
{
    private readonly IWeatherService weatherService;

    public WeatherController(IWeatherService weatherService)
    {
        this.weatherService = weatherService;
    }

    // GET /api/weather
    [HttpGet]
    public IActionResult GetWeather(string location)
    {
        // Validate the location parameter and restrict access to
trusted sources
        if (!IsValidLocation(location))
        {
            return BadRequest();
        }

        // Make a call to the third-party weather API through the
weather service
        WeatherData weatherData =
weatherService.GetWeatherData(location);

        if (weatherData == null)
        {
            return NotFound();
        }

        return Ok(weatherData);
    }

    private bool IsValidLocation(string location)
    {
        // Implement validation logic to ensure the location is safe and
trusted
        // This could involve white-listing trusted sources or
```

```
validating against a known set of safe locations
        // Return true if the location is valid, false otherwise
        // Example: return Regex.IsMatch(location,
"^[a-zA-Z]+(,[a-zA-Z]+)*$");
        // Implement your validation logic here

        // For simplicity, assuming any location is valid
        return true;
    }

    // Other methods...
}
```

کد آسیب پذیر (جاوا):

```
@RestController
@RequestMapping("/api/weather")
public class WeatherController {

    private final ThirdPartyWeatherService weatherService;

    public WeatherController(ThirdPartyWeatherService weatherService) {
        this.weatherService = weatherService;
    }

    // GET /api/weather
    @GetMapping
    public ResponseEntity<WeatherData> getWeather(@RequestParam String
location) {
        // Make a direct call to the third-party weather API
```

```
        WeatherData weatherData =
weatherService.getWeatherData(location);
        return ResponseEntity.ok(weatherData);
    }

    // Other methods...
}
```

پیشگیری (جاوا):

```
@RestController
@RequestMapping("/api/weather")
public class WeatherController {

    private final ThirdPartyWeatherService weatherService;

    public WeatherController(ThirdPartyWeatherService weatherService) {
        this.weatherService = weatherService;
    }

    // GET /api/weather
    @GetMapping
    public ResponseEntity<WeatherData> getWeather(@RequestParam String
location) {
        // Validate the location parameter and restrict access to
trusted sources
        if (!isValidLocation(location)) {
            return ResponseEntity.badRequest().build();
        }
```

```
        // Make a call to the third-party weather API through the
weather service
        WeatherData weatherData =
weatherService.getWeatherData(location);

        if (weatherData == null) {
            return ResponseEntity.notFound().build();
        }

        return ResponseEntity.ok(weatherData);
    }

    private boolean isValidLocation(String location) {
        // Implement validation logic to ensure the location is safe and
trusted
        // This could involve white-listing trusted sources or
validating against a known set of safe locations
        // Return true if the location is valid, false otherwise
        // Example: return location.matches("^[a-zA-Z]+(,[a-zA-Z]+)*$");
        // Implement your validation logic here

        // For simplicity, assuming any location is valid
        return true;
    }

    // Other methods...
}
```

پیشنهادات کلی جلوگیری:

اعتماد به داده‌های دریافتی از API های خارجی با احتیاط و اعتبارسنجی دقیق آنها.

بررسی و تحقق از امنیت و استانداردهای امنیتی سرویس شخص ثالث قبل از اتصال به آن.

استفاده از رمزنگاری برای ارتباط با سرویس‌های خارجی و جلوگیری از ارسال اطلاعات حساس به صورت عادی.

محدود کردن دسترسی و سطوح اجازه داده شده به سرویس‌های شخص ثالث و تنظیم محدودیت‌های مناسب.

پیاده‌سازی مکانیزم‌های حفاظتی مانند نمونه‌سازی و تعمیم برای اطمینان از امنیت و قابلیت اطمینان داده‌های دریافتی از سرویس‌های خارجی.

مانیتورینگ و پایش مداوم برای تشخیص و رفع هرگونه نقص در امنیت سرویس‌های خارجی.

آموزش توسعه‌دهندگان در خصوص اصول امنیتی و استفاده صحیح از API های خارجی.