Sample Web App/Service Pen Test Report

ExampleCorp

December 10, 2019

Prepared for
John Zenith

Prepared by
Alice Symvoulos
Bob Consigliere

Executive Summary

Synopsis

During the summer of 2019, ExampleCorp engaged NCC Group to conduct a time-boxed web application security assessment of the AcmeOrders application. ExampleCorp provided NCC Group consultants with architecture diagrams, source code for the application, and a test environment. The assessment uncovered both common and application-specific flaws. NCC Group provided per-Finding mitigation guidance as well as systemic recommendations for improving the security of AcmeOrders.

Scope

NCC Group's evaluation included the AcmeOrders application, which provides a dashboard that aggregates information related to orders generated by ExampleCorp's sales agents who negotiate sales transactions with ExampleCorp clients. Additionally, the application provides account management functionality, mechanisms for submitting customer feedback, and a set of customer-facing APIs that allow ExampleCorp's customers to programmatically submit sales orders directly, using pre-negotiated pricing. To support this feature, ExampleCorp has modified AcmeOrders to support a database-per-tenant model.

Testing was performed on a User Acceptance Testing ("UAT") version of the application hosted on https://acmetest.[examplecorp].com/.

NCC Group's assessment methodology is described in Methodology on page 4, and included, at a high level:

- Web application penetration testing
- Web services penetration testing
- Limited source code review (to assist penetration testing)
- Limited security architecture review (to assist penetration testing)
- Limited AWS security review

NCC Group also provided optional Project Management and Technical Oversight services.

Key Findings

The assessment uncovered both common and application-specific flaws. Some of the more notable findings included:

- **SQL injection vulnerability.** During testing, NCC Group identified an unsafe SQL query that introduced an exploitable SQL injection vulnerabilities into the AcmeOrders application.
- **Multiple unique ways to compromise user accounts.** The assessment uncovered several unique ways for both authenticated and unauthenticated users to compromise user accounts using, password change, password reset, and OAuth token refresh functionality.
- **Multiple access control issues in the application.** The test identified multiple instances in the user-facing and internal APIs where data access controls were missing that would allow a lower privileged user to access sensitive customer information.
- **A lack of cross-site request forgery protections.** The AcmeOrders application does not implement controls that mitigate cross-site request forgery vulnerabilities.
- **Serious AWS misconfigurations.** Some portions of the AcmeOrders application are hosted on AWS. NCC Group identified misconfigurations of the underlying AWS infrastructure that resulted in privilege escalation and data disclosure vulnerabilities.
- **An arbitrary file read vulnerability in the `OrderRecordController`.** The `OrderRecordController` provides a route that an attacker can leverage to read files from the AcmeOrders server's file system.

Key Recommendation

Based on, and in addition to, NCC Group's per-finding mitigation guidance provided in Table of Findings on page 7, NCC Group suggests the following systemic recommendation for improving the security of AcmeOrders:

- **Design and implement strong access controls** A strong access control model verifies the identity of users (authentication) and the privileges associated with each user (authorization). Issues with both of these items were found in AcmeOrders. For authentication, NCC Group recommends implementing Single Sign-On (SSO) for AcmeOrders. Use a well-tested off-the-shelf library to implement this functionality, which is less likely to contain security vulnerabilities than a custom implementation. After authenticating users via SSO, grant the user a unique token and verify the token with each request. Perform security testing to ensure that security tokens are strongly validated to prevent issues like finding NCC-SampleWAPT-002 on page 10.

Dashboard

Target Metadata

Name	AcmeOrders
Type	Web Application
Platforms	Windows .NET Application
Environment	Dedicated test environment

Engagement Data

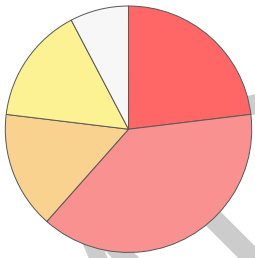
Type	Web application security assessment
Method	Web application/services pen test, source code-assisted, limited security architecture review, AWS security review.
Consultants	2
Level of effort	20 person-days

Targets

AcmeOrders Portal	https://acmeorders.[examplecorp].com

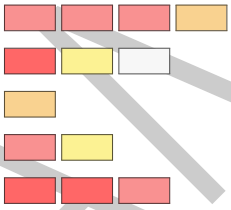
Finding Breakdown

Critical Risk issues	3
High Risk issues	5
Medium Risk issues	2
Low Risk issues	2
Informational issues	1
Total issues	**13**



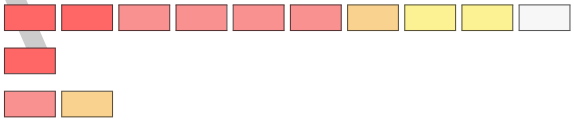
Category Breakdown

Access Controls	4
Authentication	3
Cryptography	1
Data Exposure	2
Data Validation	3



Component Breakdown

Web Application	10
API	1
AWS	2



Key

Critical High Medium Low Informational

Methodology

Web application assessments identify security vulnerabilities that may be exploited to compromise either the server-side application or ordinary users of the application.

NCC Group primarily relies on manual security testing techniques performed by highly experienced experts, supported by both internally-developed and open/commercial tools (such as Burp Suite Pro) as appropriate. Examples of NCC Group's internally-developed tools can be found at https://www.nccgroup.trust/us/our-research/?research=Public+tools.

All work was subject to NCC Group's internal peer-review processes and quality assurance before being released to ExampleCorp.

For this project, NCC Group performed a targeted web application security assessment on the in-scope targets, tailored to the project timeframe, focused on the following:

- **Custom Logic and Application Abuse Testing:** using manual techniques and both public and internal tools, NCC Group implemented selected tests based on the following attack patterns:
  - Obtain access to private data belonging to other users or subscribers.
  - Make unauthorized changes to system or customer data.
  - Bypass business logic rules around account changes.
  - Bypass authentication and authorization mechanisms.
  - Elevate privileges to site administrator or other higher-privileged users.
  - Hijack accounts belonging to other users.
  - Abuse username and password recovery methods.
- **Common Web Application Vulnerability Testing:** NCC Group also evaluated the application for standard and advanced application web security issues, including:
  - Cross-site scripting (XSS)
  - Cross-site request forgery (CSRF)
  - Vulnerabilities in rich content (such as Flash, ActiveX, and Silverlight)
  - Command injection (SQL, LDAP, and OS command injection)
  - Server side includes (SSI) injection
  - Common weaknesses in session management (authentication and authorization) including:
    - Improper token invalidation
    - Predictable session tokens
    - Insufficient protection against session fixation
  - Cookie injection attacks
  - XML injection attacks
  - Attacks against the HTTP application server, including:
    - Insecure configuration
    - Processing of unsafe HTTP verbs
    - Response splitting
    - Directory traversal
    - Forced browsing
  - Information leaks
  - Improper authorization/permissions/object references using common techniques like dot-dot-slash
  - Transport layer security weaknesses, including:
    - Weak cipher suite configuration
    - Insufficient protection of sensitive information in transit

For this project, ExampleCorp additionally contracted for the following assessment services:

- **Source Code Security Review:** Source code was provided and included within the scope of the Statement of Work (SOW), and thus NCC Group performed partial source code security review to support dynamic testing, focused on the application's security protections, including:

  - CSRF protection mechanisms
  - Data validation mechanisms
  - Output encoding mechanisms
  - Database calls and queries (usage and implementation)
  - Session creation (session generation and session token randomness)
  - Authentication and access control mechanisms and frameworks
  - Cryptographic protections
- **Architecture Security Review:** Access to ExampleCorp's development personnel (for interviews) and documentation was provided and included within the scope of the SOW, and thus NCC Group performed partial architecture security review including:
  - Interview ExampleCorp's personnel to understand the system's threat model and security requirements or objectives
  - Review engineering and product documentation, such as data flow diagrams, product requirements, security documentation (such as threat models, if any), and deployment scenarios.
  - Review security assertions desired and any gaps between them and those provided by the system.
  - Identify possible weaknesses in the existing design as a result of misunderstood assumptions, missing security guarantees in the underlying networks, applications, or operating systems, or insufficient specification.
  - Identify future weaknesses that could occur due to changing assumptions or deployment scenarios, and document features/functionality that must be maintained for the system to mitigate intended risks over time
  - Identify and describe additional features or functionality that could increase the security of the application/system, such as compensating controls that can provide defense-in-depth.
- **AWS Security Review:** AWS security configuration was optionally included within the scope of the SOW, and thus NCC Group performed partial AWS security review including:
  - Evaluated the configuration, security groups, IAM settings, and associated controls of the following:
    - Amazon EC2 instances
    - Virtual Private Cloud (VPC) instances and components
    - Elastic Load Balancing (ELB) services
    - S3 buckets
    - DynamoDB tables
  - Evaluated the overall use and leverage of IAM and permissions within the environment
  - Evaluated authentication and access control mechanisms, including use of bastion host(s)
  - Identification of sensitive materials (such as AWS secret keys and tokens) and protection mechanisms
  - Reviewed logging and auditing mechanisms and configuration

For this project, ExampleCorp additionally contracted for the following supporting services:

- **Project Management:** a dedicated NCC Group Project Manager (PM) with experience managing security projects acted as the primary interface with ExampleCorp project personnel, and was responsible for the following:
  - Provide oversight and coordination of entire project from start to finish. Serve as the single point of contact for project scheduling, logistics, and non-technical issues
  - Collate, communicate, and track all project-related tasks and issues. Perform necessary follow-up to ensure tasks and issues are resolved in a timely manner
  - Ensure all required review materials are ready at the proper time. Notify the appropriate parties of missing or overdue materials impacting NCC Group's ability to deliver the work in scope
  - Ensure any schedule or scope adjustments are accurate and made with the appropriate agreement of all parties
  - Facilitate access to internal and/or privileged client systems and locations
  - As needed, coordinate the acquisition, provisioning, shipping and/or tracking of any required hardware or devices
  - Coordinate and schedule all project meetings per standard NCC Group process
  - Quickly escalate and resolve project issues as they arise
  - Ensure any agreed-to procedures or plans are followed, both by NCC Group and ExampleCorp
  - Resolve any third-party project dependencies

- **Technical Oversight:** an NCC Group Senior Manager oversaw planning, execution, quality assurance, and post-project follow-up. The NCC Group Senior Manager also was accountable for overall performance under the associated contract for this project. Activities included:
  - Strategic input to project kick-off, status, and final read-out meetings
  - Final review and approval of all deliverables, including status reports and final report
  - Overall accountability for project performance, including management of escalations such as project schedule changes and technical feasibility issues
  - Delivery of specific portions of the project as necessary to enhance or amplify in-scope activities and deliverables
  - Strategic input to technical direction, including helping prioritize focus areas based on target risk model, providing subject matter expertise related to in-scope target technologies and assessment techniques, and guidance on integrating the outcomes of this project into ExampleCorp's overall cybersecurity program

Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. For an explanation of NCC Group's risk rating and finding categorization, see Appendix A on page 30.

Title	Status	ID	Risk
Pre-authentication SQL Injection in DoLogin	Reported	001	Critical
Users Can Become Other Users via Manipulated JWT	Reported	002	Critical
Account Takeover via SMS Password Reset	New	014	Critical
Users Can Change Arbitrary User's Password via UpdatePassword	New	003	High
Arbitrary File Read in `ExampleFilesObject`	Reported	004	High
Design Pattern Creates Risk of Server-Side Request Forgery (SSRF)	Reported	007	High
Application Returns Incorrect Content-Type for JSON Responses	New	011	High
S3 Buckets Misconfigured	New	013	High
Application Uses Weak Hash to Store Passwords	Reported	006	Medium
IAM PassRole Permission Allows Potential Authorization Bypass	New	012	Medium
Verbose Error Messages Disclose Sensitive Information	Reported	008	Low
Potential Backdoor User	Reported	010	Low
Multiple Complex Authentication Methods	Reported	009	Informational

Finding Details

Finding	**Pre-authentication SQL Injection in DoLogin**
Risk	Critical Impact: High, Exploitability: High
Identifier	NCC-SampleWAPT-001
Status	Reported
Category	Data Validation
Component	Web Application
Location	The `DoLogin` function in `AcmeOrders-git/v2/ExampleCorp.Orders.OrderUtils/Users/UserRepo.cs`.
Impact	An attacker can compromise all data in the database, and can often compromise the underlying operating system.
Description	SQL Injection is a class of attacks related to web application input and output validation. The flaw is found in situations where an end user can submit data to the application residing on the server, which then uses this data to dynamically generate and execute SQL queries to a database (most commonly, via string concatenation). If the application fails to adequately limit the input a user can present to the application, it is possible for an attacker to take advantage of this to execute arbitrary SQL statements. This will generally result in complete compromise of the data in the database, and can often result in compromise of the underlying operating system. Generally, the AcmeOrders backend API was architected to use stored procedures with parameter binding. This is a safe method of query construction that is not vulnerable to SQL injection. However, in this instance, the application is generating a dynamic SQL query by interpolating String values into the SQL query. At line 835 of `DoLogin`, the `userName` parameter is interpolated into a SQL statement, which is then executed. This is a string value that is entirely controlled by the user. By using specially chosen characters, an attacker can modify the query that is ultimately performed against the database in order to read or modify critical application data, bypass application logic, escalate privileges within the database, or use the database to access the filesystem or operating system functionality. **Note:** Similar instances of this issue are present in the `ModifyCreds` and `StoreCreds` functions.
Reproduction Steps	1. Send the following `POST HTTP` request to https://acmeorders.[examplecorpurl].com/Home/ValidateMe:

```
POST Home/ValidateMeHTTP/1.1
Host: acmeorders.[examplecorpurl].com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:68.0) Gecko/2010
➜  0101 Firefox/68.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://acmeorders.[examplecorpurl].com/Home/Start
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 91
```

```
DNT: 1
Connection: close
Cookie: ASP.NET_SessionId=<sessionId>

username=invalid@nccgroup.com' OR 1=1--&password=foo
```

2. Note that the user is now authenticated as the default "Orders" role.

Recommendation

SQL injection is best prevented by the use of parameterized queries (also called prepared statements), which define a SQL query string and data elements separately. This prevents the user from entering data that could escape the original SQL query to modify its functionality. As an additional benefit, execution of parameterized queries is often faster than that of dynamic SQL queries. Prepared statements should be used for all non-static queries, even those that do not directly take user input; dynamic data must not be interpolated into queries using string concatenation.

The following code shows a safely constructed query:

```
java.sql.PreparedStatement safeStatement = connection.preparedStatement(
    "SELECT * FROM products WHERE prodName = ? AND serialNumber = ?" );

safeStatement.setString(1, productName);
safeStatement.setInt(2, serialNumber);

safeStatement.executeQuery();
```

For more advice on fixing SQL injection, we recommend consulting the OWASP SQL Injection Prevention Cheat Sheet .

Finding	**Users Can Become Other Users via Manipulated JWT**
Risk	Critical Impact: High, Exploitability: High
Identifier	NCC-SampleWAPT-002
Status	Reported
Category	Data Validation
Component	Web Application
Location	https://acmeorders.[examplecorp].com/api/token
Impact	An attacker can create a valid JWT for any other user by sending a JWT containing a modified username to the `https://acmeorders.[examplecorp].com/api/token` endpoint.
Description	The `https://acmeorders.[examplecorp].com/api/token` endpoint provides multiple code paths whereby users can exchange their access token for a new access token based on the `perm_type` parameter sent in the request. The `validate`, `validate_token`, and `reload_token` grant types call the `ValidateToken`, `ValidateTokenObj`, and `ReloadToken` functions respectively. Each of these functions call the `LoadToken` function on the JWT access token provided by the user. The `LoadToken` function does not validate the signature of the `JwtToken` object returned by the function. As such, an attacker can set arbitrary values within the JWT that are then trusted by the application. Using this ability, the attacker can change the username of the JWT to receive a new, signed JWT from the application that would allow the attacker to act as the user.

Reproduction Steps	1. Send the following `POST` request body to `https://acmeorders.[examplecorp].com/api/token`:
	`client_id=AO&client_secret=AO%314159&access_token=eyJhbGciOiJIUzI1NiIsInR5cCI`
	`➜ 6IkpXVCJ9Cg.eyJzdWIiOiJ0ZXN0dXNlcjNAbmNjZ3JvdXAuY29tIiwidGVuYW50IjoiQWNtZS`
	`➜ IsInR1bmFudE1kIjoiMSIsInNhbGVzRm9ybWF0IjoiMSIsImFsbFR1bXBsYXRlcyI6IjEiLCJh`
	`➜ bGxBZ2VudHMiOiIxIiwiIjowOTAwODg4NSwiZXhwIjoxNTY5OTk5OTk5LCJpc3MiOiJBY211Q2`
	`➜ 9ycCIs.&grant_type=validate_token.`
	2. **Note:** Observe that the `access_token` parameter has an empty signature field.
	3. Send the above request and observe that a valid `access_token` and `refresh_token` are returned for `testuser3@nccgroup.com`.
	4. Modify the contents of the above `access_token` to include any user within the `sub` field of the `access_token`.
	5. Resend the request and observe that a new `access_token` and `refresh_token` are returned for the new user.

Recommendation	Use the `LoadAndVerifyToken` function, instead of the `LoadToken` to validate the access token using the certificate that the access token was signed with. Additionally, validate the `alg` and any other pre-validation field match the expected algorithm or value of the received access token. Finally, the OAuth Working Group has developed a draft document outlining the JSON Web Token Best Current Practices[1] that ExampleCorp can use to better ensure their design conforms with the industry accepted best practices.

[1] https://tools.ietf.org/html/draft-ietf-oauth-jwt-bcp-06

Finding	**Account Takeover via SMS Password Reset**
Risk	Critical Impact: High, Exploitability: High
Identifier	NCC-SampleWAPT-014
Status	New
Category	Authentication
Component	API
Location	https://acmeorders.[examplecorp].com/api/reset-pw-sms
Impact	Attackers can takeover a user account by sending a forged password reset request. Once an attacker takes over an account, they can post new content, delete content, modify account settings, manage orders, and more. Private information, such as private order information, phone numbers, location, and gender can be obtained by an attacker.
Description	The endpoint at https://acmeorders.[examplecorp].com/api/reset-pw-sms does not verify if the `_uid` parameter is tied with the phone number and four-digit password reset code when a `POST` request is sent to it. An attacker can request a SMS password reset of their own account and have a four-digit code sent to their phone. After getting the code, they can submit the code to the `reset-pw-sms` endpoint and replace the `_uid` value with any `_uid` of their choice. This will allow the attacker to change the password for another users account. The `_uid` value of a specific user can be found by visiting the target user's profile and using a browser proxy to view the request sent to `/profile/username/[_uid]` (where `[_uid]` looks like a long string of random letters and numbers) to get the user's `_uid`.
Reproduction Steps	1. Create a test account on https://acmeorders.[examplecorp].com. 2. Add a phone number to the test account by going to the Profile page of the account. 3. Save the phone number and logout. 4. At the Login page, go to "Request password reset." 5. Enter the phone number from step 2 to receive a four-digit reset code by SMS. 6. After receiving the code, use a web proxy such as BurpSuite to intercept web requests from a browser. 7. Enter the code on https://acmeorders.[examplecorp].com and intercept the request to https://acmeorders.[examplecorp].com/api/reset-pw-sms, which contains the four-digit code. 8. Change the `_uid` parameter to the `_uid` of the victim account. The `_uid` value can be found by visiting a target user's profile and viewing, using a web proxy, a request to "/profile/username/[_uid]" (where `[_uid]` looks like a long string of random letters and numbers) to get a user's `_uid`. 9. Send the request and change the password for the target/victim account.
Recommendation	Ensure that the `reset-pw-sms` endpoint is properly handling requests, including: • Perform basic validity checking, including checking any posted `_uid` parameter, phone number, and four-digit code are currently valid. • Generate random (low predictability), unique codes to authorize password reset. • Make sure four-digit codes expire after some reasonable length of time, and do not re-use codes. • Verify that the phone number in the request is associated with the proper `_uid` and that

the supplied code was sent to that phone number.

Finding	**Users Can Change Arbitrary User's Password via UpdatePassword**
Risk	High Impact: High, Exploitability: High
Identifier	NCC-SampleWAPT-003
Status	New
Category	Access Controls
Component	Web Application
Location	https://acmeorders.[examplecorpurl].com/api/User/UpdatePassword
Impact	Any AcmeOrders user can change the password of any other user.
Description	The `User/UpdatePassword` API allows users to specify an arbitrary `username` and `forceNewPassword` parameter for the request. These values are passed to the `UpdatePassword` function in the `UserRepo` class. The `UpdatePassword` function interpolates the values into a database query and stores them into the database. This allows an attacker to update or deny access to any user's account by directly invoking this endpoint.
Reproduction Steps	1. Authenticate to the AcmeOrders application and retrieve the access token returned to the user. 2. Send the following request to https://acmeorders.[examplecorpurl].com, filling in the "victim username" parameter with the name of a user you have credentials for and intend to reset the password of:

```
POST /api/v2/UpdatePassword HTTP/1.1
Host: acmeorders.[examploecorpurl].com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:68.0) Gecko/2010
➜  0101 Firefox/68.0
Accept: application/json
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Authorization: Bearer <ACCESS TOKEN HERE>
Connection: close
Referer: https://acmeorders.[examplecoropurl].com/Dash
Cookie: LB.acmeorders.[examplecorpurl].com_Pool=<value>

{"username":"<victim username>", "forceNewPassword":"NCCgrp@1234"}
```

	3. Observe that it is now no longer possible to authenticate with the victim username and credentials. 4. Additionally, by viewing the database directly, observe that the password for the victim user is now `NCCgrp@1234`.
Recommendation	Do not allow the user to submit a `username` parameter as part of a password change request; instead, retrieve the user's `username` from their JWT OAuth token. Additionally, create separate endpoints for performing a password reset for a user with a forgotten password and for users updating their password using their `prevPassword`. This will allow the endpoint to validate the user's existing password to prevent attackers with access to a vulnerability, for example server-side request forgery, which allows the attacker to invoke this endpoint.

Finding	**Arbitrary File Read in** `ExampleFilesObject`
Risk	High Impact: High, Exploitability: High
Identifier	NCC-SampleWAPT-004
Status	Reported
Category	Access Controls
Component	Web Application
Location	The 'ExampleFilesObject`controller method located at` AcmeOrders-git/v7/ExampleCorp.Orders.Busin Files/ExampleFile.cs:452'
Impact	An attacker can read files from the server, including application source code and sensitive operating system files. This could enable the attacker to steal infrastructure credentials, perform denial-of-service attacks, dump user information, or find additional vulnerabilities in the application.

Description	When accessing the filesystem, it is extremely important that an application performs security checking to ensure valid access. If a user-controlled filename is provided to the system, it opens a risk of file access that is not limited to the intended files.
	
	The application does not perform validation when returning files to the user from the `ExampleFilesObject` controller. As a result, an attacker can read arbitrary files on the application server, including application source code and configuration files. Inserting path characters (`/` or `../`) into the vulnerable parameter allows the attacker to traverse outside of the default directory to access sensitive files stored in system or application directories.
	
	For most methods in this controller, files are retrieved from a separate backend service. Those backend services may be vulnerable to file access issues as well, but were not in scope for the engagement and so were not tested. However, at least one method in the `ExampleFilesObject` controller reads files from the local filesystem, as shown here:
	
	`[HttpGet]`
	`public ActionResult DownloadSavedFile(string fileName)`
	`{`
	`  var ms = default(byte[]);`
	`  var tmpPath = Path.GetTempPath();`
	`  var fileBytes = System.IO.File.ReadAllBytes(tmpPath + fileName);`
	`  return File(fileBytes, MediaTypeNames.Application.Zip, fileName);`
	`...`
	
	The `fileName` parameter is provided by the user.

Reproduction Steps	Submit the following request to the application, substituting valid cookies:
	
	`GET /v2/ExampleFiles/SaveFile?fileIds=1,2 HTTP/1.1`
	`Host: apppreview.[examplecorpurl].com`
	`Connection: close`
	`Accept: application/json, text/javascript, */*; q=0.01`
	`X-Requested-With: XMLHttpRequest`
	`User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (`
	`➜  KHTML, like Gecko) Chrome/77.0.3865.0 Safari/537.36`

```
Sec-Fetch-Mode: cors
Content-Type: application/json; charset=utf-8
Sec-Fetch-Site: same-origin
Referer: https://apppreview.[examplecorpurl].com/ExampleFinder/Index
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: BIGipServer~[www.examplecorpurl].com_Pool=5551212; ASP.NET_SessionId=blo
➜  bxvy5778RGqf2; shortDateFormat=M/DD; longDateFormat=MM/DD/YYYY; _ga=5551212;
➜  _gid=5551212; lzt=Zulu; .ASPXAUTH=A45666888BF7899etc
```

After successfully submitting that request, submit the following request with valid cookies:

```
GET /v2/ExampleFiles/DownloadSavedFile?filename=..\..\..\..\..\..\..\..\WINDOWS\
➜  System32\drivers\etc\hosts HTTP/1.1
Host: apppreview.[examplecorpurl].com
Connection: close
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (
➜  KHTML, like Gecko) Chrome/77.0.3865.0 Safari/537.36
Sec-Fetch-Mode: cors
Content-Type: application/json; charset=utf-8
Sec-Fetch-Site: same-origin
Referer: https://apppreview.[examplecorpurl].com/ExampleFinder/Index
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: BIGipServer~[www.examplecorpurl].com_Pool=5551212; ASP.NET_SessionId=exa
➜  mbleBAFy5778RGqf2; shortDateFormat=M/DD; longDateFormat=MM/DD/YYYY; _ga=5551
➜  212; _gid=5551212; lzt=Zulu; .ASPXAUTH=example788943rrttu
```

Observe that the server responds with the contents of the `WINDOWS\System32\drivers\etc\hosts` file:

```
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: application/zip
Server: Microsoft-IIS/8.5
X-AspNetMvc-Version: 5.2
Content-Disposition: attachment; filename="..\\..\\..\\..\\..\\..\\..\\..\\WINDO
➜  WS\\System32\\drivers\\etc\\hosts"

# localhost name resolution is handled within DNS itself.
#       127.0.0.1       localhost
#       ::1             localhost
# ClientFileServer Host Entries
172.48.4.11  EXAMPLECORP01  EXAMPLECORP01.{EXAMPLEDNSNAME}.COM
...
```

Recommendation

If the multiple file zip download logic is not currently used in the application, then this can simply be removed from the application. From the web UI, it did not appear to be possible to exercise this logic, indicating that it is not currently used.

Otherwise, apply file access controls to prevent an attacker from performing attacks such as path traversal. Access controls can encompass a number of different strategies:

- Whitelisting of allowed files for writing and reading
- Input sanitization to strip filenames of dangerous characters
- Ignoring user-controlled filenames and storing files using a randomly generated filename in a safe directory

These strategies must be applied in addition to normal application protections, such as authorization checking for file access, preventing access to restricted files, and storing uploaded files outside of the application's root.

Finding	**Design Pattern Creates Risk of Server-Side Request Forgery (SSRF)**
Risk	High Impact: High, Exploitability: Undetermined
Identifier	NCC-SampleWAPT-007
Status	Reported
Category	Data Exposure
Component	Web Application
Impact	The design patterns used in the AcmeOrders web application's source code create a significant risk of SSRF vulnerabilities. Using an SSRF vulnerability, an attacker could port scan and fetch web content from the application's internal network and from services listening locally on the application server. It could also be possible to exfiltrate sensitive information or exploit insecure services that are only exposed internally.
Description	In a Server-Side Request Forgery (SSRF) attack, the attacker abuses functionality on the server to read or modify resources internal to the application's network infrastructure. The attacker can supply or modify a URL that the code running on the server will fetch or submit data to.[2] AcmeOrder's architecture and source code design patterns create a significant risk of SSRF vulnerabilities. Specifically, the AcmeOrders web application creates HTTP requests to other services on ExampleCorp's network using user-supplied parameters. If the user-supplied data can be used to change the intended destination of the request, SSRF attacks could result. The AcmeOrders web application performs a backend HTTP request for almost every non-static user request. These requests primarily go to the internal AcmeOrders API. For example, when retrieving details on a specified order, the following method of `ExampleController.cs` is called:

```
[HttpGet]
public ActionResult ExampleOverview(int exampleId)
{
    var client = new ApiClientHandler(Session["accessToken"].ToString());
    var apiUrl = $"{ConfigurationManager.AppSettings["API_URLBase"]}/Examples/{e
➜   xampleId}";

    var examples = client.GetAsyncWithAutoTokenRefresh<Example>(this, apiUrl);
...
```

This method concatenates the user-supplied `exampleId` parameter into a URL `apiUrl`, which is then requested via the method `GetAsyncWithAutoTokenRefresh`. This is a pattern that has a significant risk for SSRF vulnerabilities. In this case, because `exampleId` is parsed as an integer, there is little risk. But in other methods, if a string parameter is provided, an attacker could potentially control the URL of the request to change the destination host or to change the specific route on the host.

There are other locations where this pattern occurs using user-supplied string parameters, such as the `registration` parameter in the following method of `ExampleReportController.cs`:

[2] Wallarm's SSRF bible (PDF) provides a thorough introduction to SSRF vulnerabilities and exploitation.

```
public ActionResult ExportData(DateTime beginDate, DateTime endDate, string regi
➜  stration,
    string examplenumber, string customerCode, string exampledepStn,
    [DataSourceRequest] DataSourceRequest request)
{
    if (Session["accessToken"] == null)
        return Redirect("~/Home/Authorization");
    var client = new ApiClientHandler(Session["accessToken"].ToString());
    var getAllUrlString = $"{ConfigurationManager.AppSettings["API_URLBase"]}/"
➜  +
        $"ExampleReport?beginDate={beginDate}&endDate={endDate}&" +
        $"registration={registration}&examplenumber={examplenumber}&" +
        $"customerCode={customerCode}&exampledepStn={exampledepStn}";
```

NCC Group did not attempt to identify all locations where this pattern occurs, as the pattern is systemic across the AcmeOrders web application. There are a number of possible ways to exploit the issue:

- Changing the destination host from the expected host (e.g. the internal AcmeOrders API) to another backend system
- Changing the requested route (e.g. `/ExampleReport`) to a different route on the same host, such as `/User/ChangePassword`
- Adding or modifying request parameters that are not otherwise user-controlled

The actual exploitability of the issue has not been explored, as an exploit would depend on a number of complex factors: the context of the vulnerability itself, the code logic that handles URL parsing,[3] and the logic and routing of the targeted system.

Recommendation

First, all user-supplied parameters should undergo URL encoding before being included in URL strings. This appears to be the primary issue within the AcmeOrders codebase. By forcing URL encoding on user-supplied parameters, those parameters will not be able to break the application's URL parsing logic or add additional parameters. .NET provides the `HttpUtility.UrlEncode` method for this purpose. Additional information can be found on Wikipedia.

There are a number of additional strategies that could help to mitigate this design flaw. Network traffic controls could be used to whitelist routes from the AcmeOrders web application to the internal API and prevent outbound HTTP requests to other servers; however, the web application talks to other hosts as well, and all whitelisted hosts would still be vulnerable to attacks.

Another strategy would be to perform parameter validation on all user-supplied parameters in the user-facing web application. The validation of ID types (must be an integer) provides an example of this. If there are known formats for other fields (in particular, all string fields that are supplied by users), those could be validated to ensure they match a safe character set. For most purposes in AcmeOrders, NCC Group believes that whitelisting the character set `[a-zA-Z0-9_\-]` (letters, digits, underscores, and hyphens) would prevent exploitation of most vulnerabilities without overly restricting input. However, this character set may not suffice for all purposes (such as email addresses), and so it would not necessarily be possible to apply it universally.

[3] See Orange Tsai's talk *A New Era of SSRF: Exploiting URL Parser in Trending Programming Languages* (PDF)

Finding	**Application Returns Incorrect Content-Type for JSON Responses**
Risk	High Impact: High, Exploitability: High
Identifier	NCC-SampleWAPT-011
Status	New
Category	Data Validation
Component	Web Application
Location	The JSON responses for XMLHttpRequest requests performed by AcmeOrders, such as: • https://[acmeorder_uri]/dir1/get_orders • https://[acmeorder_uri]/dir1/get_invoices • https://[acmeorder_uri]/dir1/foo.json • etc. • etc.
Impact	Browsers may behave unexpectedly when presented with an incorrect `Content-Type`. Since AcmeOrders is returning JSON content with a `Content-Type` of `text/html`, the application is vulnerable to cross-site scripting attacks.
Description	Web browsers rely on the HTTP `Content-Type` header to determine how to handle responses from web applications, whether that content is HTML, JavaScript, an image, or something else. If an application returns an invalid or incorrect `Content-Type`, browsers will still attempt to parse the response, but often behave unexpectedly or insecurely. The AcmeOrders application returns JSON responses containing user-controller information, but with an HTML `Content-Type`:

```
HTTP/1.1 200 OK
Server: TornadoServer/6.0.3
Content-Type: text/html

{"draw":4,"recordsTotal":94050,"recordsFiltered":1,"data":[["ncc-group-
➜  test@[examplecorptest].com","A test group","Member"]]}
```

Upon receiving this response, a web browser will skip past all preliminary characters and begin parsing once it reaches an HTML tag. A malicious user could change their group description in order to cause cross-site scripting:

```
HTTP/1.1 200 OK
Server: TornadoServer/6.0.3
Content-Type: text/html

{"draw":4,"recordsTotal":94050,"recordsFiltered":1,"data":[["ncc-group-
➜  test@[examplecorptest].com","A test group<script>alert(1)</script>","Member"
➜  ]]}
```

Applications must always set the correct `Content-Type` for responses. Any invalid or incorrect `Content-Type` could potentially be exploited.

Recommendation	For every response containing a message body, the web application should include a single `Content-Type` header that correctly and unambiguously states the MIME type of the content in the response body. AcmeOrders should ensure that all JSON responses have a `Content-`

`Type` of `application/json`. Tornado's `RequestHandler.write()` will do this automatically if the supplied argument is a dictionary (`dict`); however, AcmeOrders instead passes `write()` a string (`str`) by calling `json.dumps()`.

Applications that serve uploaded files should additionally set the following headers to prevent `Content-Type` confusion on user-uploaded content:

- `X-Content-Type-Options: nosniff`: instructs browsers not to guess at the `Content-Type`
- `Content-Disposition: attachment`: instructs the browser to download the file rather than render it

Finding	**S3 Buckets Misconfigured**
Risk	High Impact: High, Exploitability: High
Identifier	NCC-SampleWAPT-013
Status	New
Category	Access Controls
Component	AWS

Location

NOTE: this finding results from an optional service focused on cloud security review; while not typically a part of a standard web application security assessment, we commonly identify cloud-related issues like this for applications hosted on platforms such as AWS, Azure, and Google Cloud Platform.

The following S3 buckets in ExampleCorp's AWS environment.

- `examplecorp-sitemaps`
- `examplecorp-demo`
- `examplecorp-orders`
- `examplecorp-invoices`
- `examplecorp-test`
- `examplecorp-chipper`
- `examplecorp-builds`

Impact

For components of AcmeOrders that utilize S3 buckets for storage, an attacker may do the following:

- Enumerate the contents of the buckets whose ACLs grant read (list) permission to the "authenticated users" group or anonymous users.
- Upload new files and use ExampleCorp's bucket to host and distribute malware.
- Gain information about the access controls in place for the target buckets and mount further attacks using this information.
- Access objects in buckets that are not granted the read (list) permission for anonymous users.

Description

NCC Group identified the following misconfigurations in ExampleCorp's S3 buckets.

- The `examplecorp-sitemaps` bucket is world-readable (listable) for anonymous users.
- The `examplecorp-demo` bucket is world-writable for anonymous users.

Additionally, it was discovered that the following buckets have a policy to allow any AWS principle to perform get actions on the objects that reside within the buckets. This effectively makes all objects in the bucket readable.

- `examplecorp-orders`
- `examplecorp-invoices`
- `examplecorp-test`
- `examplecorp-chipper`
- `examplecorp-builds`

```
{
  "Action": [
    "s3:GetObject"
  ],
  "Effect": "Allow",
  "Principal": "*",
  "Resource": "arn:aws:s3:::[bucket-name]/*",
```

```
  "Sid": "AddPerm"
}
```

Although NCC Group was not able to examine each exposed object, due to the time restricted nature of the engagement, several buckets seem to contain sensitive information. For example, objects in the `examplecorp-invoices` bucket include ERB (Embedded RuBy) templates, PDFs, and images. Objects in the `examplecorp-builds` bucket include client-side source code used by the web application.

It should be noted that at the time of the engagement no malicious items were found in the `examplecorp-demo` bucket.

Reproduction Steps

1. Examine the bucket ACLs for the target buckets. The only permission grants for the given bucket are for the user `zack`.

```
aws s3api get-bucket-acl --bucket examplecorp-orders
```

```
{
    "Owner": {
        "DisplayName": "zack",
        "ID": "820a4f[IDstring..................]64"
    },
    "Grants": [
        {
            "Grantee": {
                "DisplayName": "zack",
                "ID": "820a4f[IDstring..................]64",
                "Type": "CanonicalUser"
            },
            "Permission": "FULL_CONTROL"
        }
    ]
}
```

2. Navigate to: https://[examplecorp-ordersurl].s3.amazonaws.com/dir1/documents/dir2/dir3/order_template.html.erb
3. Note that you have just accessed the ERB template used by the AcmeOrders web application.

Recommendation

Ensure that the IAM Policies, S3 Bucket Policies, and S3 Access Control Lists (ACLs) limit the access of the S3 bucket to those who have a defined business need. If an S3 bucket is no longer needed, it should be deleted.

Finding	**Application Uses Weak Hash to Store Passwords**
Risk	Medium Impact: Medium, Exploitability: Medium
Identifier	NCC-SampleWAPT-006
Status	Reported
Category	Cryptography
Component	Web Application
Location	The authentication logic defined in `exampleorder-git/v7/Example.Path.BusinessLogic/Users/ExampleUser.cs:235-258`.
Impact	An attacker who is able to compromise the application's storage, such as through a database flaw or information disclosure, can more easily recover user passwords.
Description	The application stores user passwords for accounts using two password hashing schemes, with varying weaknesses. NCC Group traced the flow of authentication logic through the application and found the following method: `ExampleBranch::AuthenticateUser()` – logic to authenticate user against local database, and to choose between two password hashes • `OldExampleEncryption::CheckPassword()` – legacy password check, uses `bcrypt` with a static salt • `SecurityExampleLibrary::DiffExamplePasswords()` – current password check, uses SHA512 (single round) with a unique salt `bcrypt` is a safe, widely-supported password hash. However, the underlying implementation uses a static salt (line 721 of `OldExampleEncryption.cs`). A salt is a random per-password value that slows down cracking attempts by requiring attackers to perform a new brute-force attack on each password. For example, with unique salts, two users with the password "123456" would have different password hashes. With a static salt, an attacker with access to hashed passwords would not need to attempt to crack each password individually. Instead, a brute-force attack would enable them to attack all password hashes concurrently without any extra work. In comparison, the `DiffExamplePasswords` does use a unique salt for each password. However, SHA512 is not an effective hash for password hashing. Because this hash can be performed very quickly, hashed passwords are vulnerable to brute-force cracking. An attacker with access to the hashed passwords is likely to be able to recover significant numbers of plaintext passwords using a tool such as hashcat. An ideal password hash forces an attack to use significant resources to crack dumped password hashes. Additionally, the presence of two separate hashing schemes (one named "old") suggests that a migration was never finalized. When switching to a new hashing scheme, it is best to create a defined process and a timeline for the complete switchover to be performed so that legacy logic may be removed in an explicit timeframe. This way, vulnerabilities in outdated code do not affect the application after the code is no longer in use.
Recommendation	NCC Group recommends using bcrypt, a widely-supported hash that handles salting automatically. This hashing algorithm is designed to resist brute-forcing attempts. The only aspect of bcrypt that requires manual configuration is the "cost factor", a value which deter-

mines how many iterations to perform[4] (thus determining the amount of processing time necessary to hash a single password). NCC Group recommends that most applications use a cost factor of 12. On modern processors, this cost factor results in hashing taking approximately half a second for each password.[5]

The existing bcrypt implementation has a number of issues and should not be used going forward. Instead, use a library with a safe API such as https://github.com/BcryptNet/bcrypt.net. This library handles salting internally and a salt should not be supplied to the library calls.

To upgrade to a modern password hash, there are a number of possible strategies:

- Discard all existing hashes and force users to set new passwords. This strategy may work well for applications with a small number of users who can be reached directly, such as internal company applications.
- Upgrade users at their next login by comparing the existing hash, saving the new hash, and finally discarding the existing hash. This strategy is effective for active users, but can have issues with users who do not log in for long periods of time: their old, weak password hashes will remain in the database until they log in.
- Immediately upgrading all hashes by using the new hashing algorithm on the existing hashed passwords. When users log in, first hash their password with the legacy algorithm, then hash again with the modern algorithm. This strategy is safe in most cases (even for weak algorithms like MD5), but requires the application to maintain the backwards-compatibility logic indefinitely.

[4]See https://security.stackexchange.com/a/3993 for a more in-depth explanation of work factors for password hashing.
[5]See https://security.stackexchange.com/a/83382 for a benchmark of bcrypt on a recent Intel processor.

Finding	**IAM PassRole Permission Allows Potential Authorization Bypass**
Risk	Medium Impact: High, Exploitability: Low
Identifier	NCC-SampleWAPT-012
Status	New
Category	Access Controls
Component	AWS

Location

NOTE: this finding results from an optional service focused on cloud security review; while not typically a part of a standard web application security assessment, we commonly identify cloud-related issues like this for applications hosted on platforms such as AWS, Azure, and Google Cloud Platform.

The following IAM policies in ExampleCorp's AWS environment:

- `AWS-CodePipeline-Service`
- `CloudFormationLambdaExecuteRoll`
- `AWSLambdaFullAccess`
- `AutoScalingServiceRolePolicy`
- `AWSDataPipelineRole`
- `TeamCityServer`
- `AWSEC2SpotServiceRolePolicy`
- `LambdaFunctionDeploy`

Impact

An IAM user in the groups above may be able to achieve privilege escalation to access resources or services without authorization.

Description

Some AcmeOrders application instances are hosted on AWS. When configuring or creating an instance of an AWS service, users generally pass on an IAM role to that service. When the service needs to perform a privileged AWS task, it can then use the permissions associated with its IAM role (and the automatically-provisioned AWS credentials) to perform the action. The AWS permission `iam:PassRole`[6] is what defines which roles an IAM user (or a service itself) can pass on. The groups listed in the location section above have the following permissions statement (edited for brevity).

```
"Action":
[ ... "iam:PassRole", ... ],
"Effect": "Allow",
"Resource": [
  "*"
]
```

This statement allows a user to pass *any* role to a new service, even one the user does not have. This ability is somewhat restricted; for example, the requested role may only be valid for certain services (defined as the "trusted entities" via the "sts:AssumeRole" permission on the role). Additionally, to pass a role to an EC2 instance, there must be an instance profile for the given role.

The effect of this statement is that a user in the target groups may be able to pass a role that has privileges the user themselves does not. For example, a user, such as `michael`, who

[6] https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_passrole.html

would normally have read-only access to S3 buckets but is also a member of `AmazonKinesisFirehoseFullAccess`, `AmazonRedshiftFullAccess`, `AmazonS3FullAccess`, or `AWSLambda_FullAccess` policies could pass themselves a role that would allow them full access to all AWS resources. As a result, the expected permissions model may be violated, allowing users to access many services or resources outside of their expected abilities.

Recommendation

When granting the `iam:PassRole` permission, the "Resource" should always be limited to only the intended role for the given user. Amazon's AWS documentation provides a tutorial with a number of examples at https://aws.amazon.com/blogs/security/granting-permission-to-launch-ec2-instances-with-iam-roles-passrole-permission/.

Finding	**Verbose Error Messages Disclose Sensitive Information**
Risk	Low Impact: Low, Exploitability: Medium
Identifier	NCC-SampleWAPT-008
Status	Reported
Category	Data Exposure
Component	Web Application
Location	The customErrors setting defined in `Web.Config:116`
Impact	The application discloses application stack traces when an unhandled error occurs. An attacker may be able to use this information to aid further attacks or as part of a social engineering attack.
Description	When an error condition occurs in AcmeOrders, the response includes a full application stack trace. A user can trigger an error condition by sending invalid input in a number of locations. Specifically, the error page included: • Full file paths of source code files • Class names, method names and arguments, and line within source code file
Reproduction Steps	Trigger a 500 error and observe that a detailed stack trace is included in the HTML body and also within an HTML comment. For example, navigate to https://exampleorder.[examplecorpurl].com/ExampleAlert/GetEmailAlertDetail?emailalertid=100 and observe the following stack trace:

```
[HttpRequestException: Response status code does not indicate success: 500 (Inte
➜  rnal Server Error).]

[AggregateException: One or more errors occurred.]
   System.Threading.Tasks.Task`1.GetResultCore(Boolean waitCompletionNotificatio
➜  n) +589907
   ExamplePath.v2.Utilities.Api`1.GetRequest(String url, String token) in C:\Cod
➜  e\ExamplePath\v2\Utilities\Api.cs:22
   ExamplePath.v2.Utilities.Api`1.GetApiData(HttpContext currentContext, String
➜  url, String recache) in C:\Code\ExamplePath\Utilities\Api.cs:123
   ExamplePath.v2.Example.GetEmailAlertDetail(JsonDynamicWrapper json) in...
[9 more similar references]
```

Recommendation	The application should trap all errors and present a generic error message to the user. Error details should be logged in the back end to a location not accessible from the external network. In .NET MVC, the `customErrors` configuration attribute can bet set to `On` to display a generic error message when an unhandled exception occurs. Microsoft provides documentation at https://msdn.microsoft.com/en-us/data/h0hfz6fc(v=vs.110).

Finding	**Potential Backdoor User**
Risk	Low Impact: Medium, Exploitability: Low
Identifier	NCC-SampleWAPT-010
Status	Reported
Category	Authentication
Component	Web Application
Location	The authentication logic defined in `AcmeOrders-git/v7/Example.Path.BusinessLogic/Users/ExampleUser.cs:512-515`.
Impact	An attacker with access to the application's source code or other internal knowledge of the AcmeOrders application may be able to use the `ExampleAPIUser` account to authenticate to the application without proper credentials. Additionally, a malicious user may be able to use this account to access the application after their application access has been removed.
Description	The authentication application logic contains two conditional statements checking for a `ExampleAPIUser` account using a hardcoded password. It appears that this conditional bypasses the normal application authentication flow. Backdoor users that circumvent application logic make it difficult to restrict access to the application after a user's access is removed. Additionally, if an attacker is able to recover the source code for the application without a valid set of credentials the attacker can potentially authenticate to the application using these backdoor credentials. Below is a portion of the conditional that checks for the `ExampleAPIUser`:

```
512 if (userLogin.UserName == "ExampleAPIUser" && userLogin.Password == "hardcoded_p
    ➜  assword_value")
513 {
514         user = new User { Id = -1, DisplayName = "ExampleAPIUser" };
515 }
```

Reproduction Steps	Navigate to https://url.[examplecorpurl].com/Login/and provide 'ExampleAPIUser' and 'hard coded_password_value' as credentials. Authentication succeeds and the user context is set as shown in the following screenshot: {example screenshot}
Recommendation	Remove this conditional statement from the application authentication logic. If AcmeOrders requires service accounts for performing application functionality, ExampleCorp should create a limited service account role, register the accounts with the application, and store their credentials in a secure secret storage mechanism such as AWS Secrets Manager,[7] Hashicorp Vault,[8] or CyberArk.[9]

[7] https://aws.amazon.com/secrets-manager/
[8] https://www.vaultproject.io
[9] https://www.cyberark.com/products/privileged-account-security-solution/enterprise-password-vault/

Finding	**Multiple Complex Authentication Methods**
Risk	Informational Impact: Undetermined, Exploitability: Undetermined
Identifier	NCC-SampleWAPT-009
Status	Reported
Category	Authentication
Component	Web Application
Impact	The risk for authentication vulnerabilities is greatly increased due to the complexity introduced by the varied authentication sources, authentication tokens, and credential storage.
Description	The AcmeOrders application uses a combination of Active Directory, application credentials, .NET session management, and JWT-based OAuth tokens for authentication. Additionally, user credentials are currently stored in two locations: the AcmeORders database ExampleUsers table and the Exampleblob database ExampleUsers table. This varied collection of authentication sources, representation, and storage has caused the AcmeOrders authentication logic to be spread across various parts of the code base. In most locations where authentication logic is implemented, the logic is complex and includes a substantial amount of control flow. In particular, the main AcmeOrders authentication entry point, `ExampleUser.AuthenticateExampleUser`, contains about 220 lines of code including a series of deeply nested control flow statements. This complexity and decentralization of the authentication logic contributed to the many authentication related vulnerabilities identified by NCC Group during testing.
Recommendation	Transition from the combination of Active Directory and application-based authentication sources to a single standardized authentication solution for both ExampleCorp employees and AcmeOrders application users. Additionally, ExampleCorp should standardize the authentication token between all ExampleCorp services. ExampleCorp should centralize all authentication logic into a single portion of the codebase, ideally in the internal APIs to allow easy reuse of the authentication logic. Optimally, ExampleCorp should migrate all authentication to an external single-sign on (SSO) provider to limit the necessity of managing user credentials. Migrating to an SSO provider would allow ExampleCorp to simplify the AcmeOrders authentication logic, improve customer credential management, easily implement security measures like multi-factor authentication, and reduce the difficulty of integrating with a customer's sign on solution if requested.

Appendix A: Finding Field Definitions

The following sections describe the risk rating and category assigned to issues NCC Group identified.

Risk Scale

NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. The risk rating is NCC Group's recommended prioritization for addressing findings. Every organization has a different risk sensitivity, so to some extent these recommendations are more relative than absolute guidelines.

Overall Risk

Overall risk reflects NCC Group's estimation of the risk that a finding poses to the target system or systems. It takes into account the impact of the finding, the difficulty of exploitation, and any other relevant factors.

Critical Implies an immediate, easily accessible threat of total compromise.

High Implies an immediate threat of system compromise, or an easily accessible threat of large-scale breach.

Medium A difficult to exploit threat of large-scale breach, or easy compromise of a small portion of the application.

Low Implies a relatively minor threat to the application.

Informational No immediate threat to the application. May provide suggestions for application improvement, functional issues with the application, or conditions that could later lead to an exploitable finding.

Impact

Impact reflects the effects that successful exploitation has upon the target system or systems. It takes into account potential losses of confidentiality, integrity and availability, as well as potential reputational losses.

High Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level.

Medium Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information.

Low Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security.

Exploitability

Exploitability reflects the ease with which attackers may exploit a finding. It takes into account the level of access required, availability of exploitation information, requirements relating to social engineering, race conditions, brute forcing, etc, and other impediments to exploitation.

High Attackers can unilaterally exploit the finding without special permissions or significant roadblocks.

Medium Attackers would need to leverage a third party, gain non-public information, exploit a race condition, already have privileged access, or otherwise overcome moderate hurdles in order to exploit the finding.

Low Exploitation requires implausible social engineering, a difficult race condition, guessing difficult-to-guess data, or is otherwise unlikely.

Category

NCC Group categorizes findings based on the security area to which those findings belong. This can help organizations identify gaps in secure development, deployment, patching, etc.

Access Controls	Related to authorization of users, and assessment of rights.
Auditing and Logging	Related to auditing of actions, or logging of problems.
Authentication	Related to the identification of users.
Configuration	Related to security configurations of servers, devices, or software.
Cryptography	Related to mathematical protections for data.
Data Exposure	Related to unintended exposure of sensitive information.
Data Validation	Related to improper reliance on the structure or values of data.
Denial of Service	Related to causing system failure.
Error Reporting	Related to the reporting of error conditions in a secure fashion.
Patching	Related to keeping software up to date.
Session Management	Related to the identification of authenticated users.
Timing	Related to race conditions, locking, or order of operations.

Appendix B: Cross-Site Request Forgery Background

[NOTE: this is an optional custom section that is sometimes included in reports to amplify a specific finding or set of findings, or to delve more deeply into complex issues. The intent is to provide further information and background to help readers develop a deeper understanding, ideally leading to stronger overall security posture as a result. One practical example is shown here, providing additional background on CSRF so that engineers can more readily identify and fix other instances in applications.]

Cross-Site Request Forgery (CSRF) is a type of attack that occurs when a user loads or interacts with an untrusted web site while logged into a vulnerable application in the same browser. The untrusted web site can cause the user's browser to submit requests (including the user's cookies) to any other site; vulnerable sites cannot differentiate between a legitimate request from the user and malicious requests submitted by other sites. Because an attacker cannot directly steal response data, CSRF attacks primarily target requests that a user would use to perform an action within the application (called "state-changing requests").

The following example demonstrates the type of state-changing action an attacker would target with a forged request:

```
app.post('/CreateUser', function(req, res) {
  if(!req.session.user.isAdmin) {
    res.status(403).send("Admins only!");
    return;
  }

  db.createUser({
    email: req.query.email,
    isAdmin: req.query.admin == 1
  });
});
```

While an attacker may not be able to access the functionality of the application themselves, they can easily target an existing user with an email, advertisement, or vulnerability in an existing webpage. In the above code, the attacker could send a targeted email to an administrator of the application. If the administrator opens a link in the email while logged into the application (even if an application tab or window is not currently open), the resulting site can submit the following request to create a new administrator:

```
POST /CreateUser?email=attacker@example.com&admin=1 HTTP/1.1
Host: vulnerable.example.com
Cookie: SessionID=123456
```

When the request is submitted, the victim's browser will automatically fill in the valid cookies for the current session.

CSRF attacks must be actively defended against, as browsers and the default configuration of many web frameworks do not offer any protection against this type of attack. However, they are generally limited by the Same-Origin Policy, which partially restricts the types of requests that can be sent, and prevents the requesting site from reading the targeted application's response. The following caveats restrict the ability of an attacker to exploit CSRF:

- An attacker must possess detailed knowledge of a targeted route in order to make the proper request on the user's behalf
- The submitted request can only use a `GET`, `HEAD`, or `POST` method
- The request's Content-Type header must be one of `application/x-www-form-urlencoded`, `multipart/form-data`, or `text/plain`[10]
- An attacker cannot create or modify other HTTP headers in the request

Applications can be protected from CSRF attacks by rejecting state-changing requests that do not originate from

[10] Previous browser issues, such as a bug in Chrome's sendBeacon API, have allowed bypasses of this restriction.

the application itself. The primary method of verifying that a request originated from the application rather than an external site is to require all state-changing requests to contain an extra parameter known as a CSRF token. These tokens are random values, generated by the server, which are returned to the browser in the body of a response or in a cookie and automatically submitted in the request body with every state-changing request. Because an attacking site cannot read the application's cookies or responses, they will be unable to submit the correct value, and any forged request will be rejected.

All state-changing actions must require a CSRF token generated by the server to accompany the request, as a body parameter or header value. If possible, the applicable functionality provided by the framework should be used to accomplish this and applied to all state-changing routes. If it is not possible to use built-in CSRF protections, the following strategy is recommended:

1. When a user logs in, generate a 128-bit random value using a cryptographic random number generator. Set the value in a cookie returned to the user.
2. When displaying an HTML form to the user, copy the token from the cookie into a hidden parameter in the form.
3. When using JavaScript to build requests (e.g. using `XMLHttpRequest`), copy the cookie's value into an extra body parameter (e.g. `csrf_token`).
4. For all state changing requests, the server should verify that the extra parameter is present and that its value matches the value of the cookie.

After implementing protections, review the application's routes for the following issues:

- All state-changing routes must have CSRF protection applied
- No `GET` or `HEAD` routes allow state-changing functionality
- Sensitive actions (e.g. password change) should require the user to re-enter their password

[Additional customer or application-specific guidance may be included here.]

Appendix C: Project Contacts

The team from NCC Group has the following primary members:

- Alice Symvoulos — Senior Consultant
  alice.symvoulos@nccgroup.com
- Bob Consigliere — Consultant
  bob.consigliere@nccgroup.com

The team from ExampleCorp has the following primary member:

- John Zenith — ExampleCorp
  jzenith@examplecorp.com