It was discovered in the last penetration test. Please see the recommendation below:

Recommendation

Disable support for weak ciphers on the server. Weak ciphers are generally defined as:   
•Any cipher with key length less than 128 bits  
•Export-class cipher suites  
•NULL ciphers  
•Ciphers that support unauthenticated modes  
•Ciphers assessed at security strenghts below 112 bits  
•All RC4 ciphers  
•All 64-bit block ciphers  
The following ciphers supported by the server are weak and should be disabled:  
•TLS\_RSA\_WITH\_RC4\_128\_MD5 (0x4)  
•TLS\_RSA\_WITH\_RC4\_128\_SHA (0x5)

The following ciphers supported by the server should provide adequate protection and may be left enabled:  
•TLS\_RSA\_WITH\_IDEA\_CBC\_SHA (0x7)  
•TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA (0xa)  
•TLS\_DHE\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA (0x16)  
•TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA (0x2f)  
•TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA (0x33)  
•TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA (0x35)  
•TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA (0x39)  
•TLS\_RSA\_WITH\_CAMELLIA\_128\_CBC\_SHA (0x41)  
•TLS\_DHE\_RSA\_WITH\_CAMELLIA\_128\_CBC\_SHA (0x45)  
•TLS\_RSA\_WITH\_CAMELLIA\_256\_CBC\_SHA (0x84)  
•TLS\_DHE\_RSA\_WITH\_CAMELLIA\_256\_CBC\_SHA (0x88)  
•TLS\_RSA\_WITH\_SEED\_CBC\_SHA (0x96)  
•TLS\_DHE\_RSA\_WITH\_SEED\_CBC\_SHA (0x9a)  
•TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256 (0x3c)  
•TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA256 (0x3d)  
•TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256 (0x67)  
•TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA256 (0x6b)  
•TLS\_RSA\_WITH\_AES\_128\_GCM\_SHA256 (0x9c)  
•TLS\_RSA\_WITH\_AES\_256\_GCM\_SHA384 (0x9d)  
•TLS\_DHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256 (0x9e)  
•TLS\_DHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384 (0x9f)

Tips

Each weak cipher was enumerated by establishing an SSL connection with the target host and specifying the cipher to test in the Client Hello message of the SSL handshake.

Here is the final list of values recommended by the cryptographic services team:  
TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA  
TLS\_DHE\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA  
TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA  
TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA  
TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA  
TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA  
TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256  
TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA256  
TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256  
TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA256

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

John,

Further to our discussion, we need to request a security exception for our Newton connection to Active Directory (AD). I will schedule a call with you and the vendor consultant to discuss this further on Monday.

**Issue:** Single sign-on (SSO) configuration for Newton conflicts with Synergy web services downstream interface configuration. Login with Multi-domain AD is not set up to use application Service ID.

**Exception required:** To use Multi-domain AD login - users key in their user name/password credentials in the application login screen, and that information is directly relayed to the AD server (internally this is managed securely through a robust API).

Exception requested for 3 months from go-live (March 7, 2016).

**Root Cause**

Currently, Web service (WS) clients authenticate to Newton via Web Service Security headers that are embedded in the SOAP requests.  Newton in turn, reads that header and authenticates the user against the configured authentication setup (e.g. Direct, Active Directory, or Multi Active Directory).

When Newton is set to use SSO, authentication is delegated to the Application Server.  JBoss is currently setup up to use SPNEGO for SSO, so it intercepts the web service requests and expects them go through SSO/SPNEGO authentication before they are allowed to pass to Newton for processing.

**Possible Solutions**

There are 3 possible approaches to address this issue:

1. Client Applications Authenticate with SPNEGO

This would require both RBC and Bosch WS clients (Synergy system as well as Rating Migration Tool or any other consumers of webservices from Newton) to authenticate with SPNEGO in order to access services.  This would require **no code changes to Newton**, but **changes to client configuration/code** in order to support SPNEGO.

1. Dual Authentication Model

Newton would need to support SPNEGO for website access and LDAP for services.

This would **require changes to the WAR for descriptors, Spring Security, and possibly code.**  The authentication model would need to be setup to be URL based so that Web Services would not authenticate via the container but all other URLS would use SPNEGO.  This approach has the most impact/changes to the server, but would enable Newton to handle any server requests keeping the WS clients current authentication models (i.e. **no changes to the clients**)

1. JBoss/Newton Mixed Instances

Specific non-SSO nodes of JBoss/Newton would be setup in the environments where there is a need for clients to access Newton without using SSO.  This would require Standalone JBoss instances (separate from the Domain) to be setup using the non-SSO war, and they would be configured to use the same database as the SSO instances of Newton.

These instances would use either AD or MAD authentication and would be accessible to the end clients via a separate Load Balancer URL.  They would be setup to only allow services to come through to the system.

The sizing of these instances (or possibly single instance) would depend on the amount of load we expect from Synergy, and the system load for the Rating Migration.

This would require **no code changes to Newton or the clients**, but would require setting up at least one Standalone JBoss instance in each SSO environment.  The **VM** chosen to host this instance would have to be analyzed to determine the amount of resources that would be required.

1. Newton cannot use SSO for authentication and must use Multi Active Directory instead. The Newton product does not use Service ID to authenticate to AD and would require **code changes by Bosch (estimate 2 months of work effort).**