Solving the TLS 1.0 Problem

By Andrew Marshall

Principal Security Program Manager

Microsoft Corporation

##### Contributors/Thanks to

Mark Cartwright

Bryan Sullivan

Patrick Jungles

Michael Scovetta

Tony Rice

David LeBlanc

Mortimer Cook

Daniel Sommerfeld

Andrei Popov

Michiko Short

Justin Burke

Gov Maharaj

Brad Turner

Sean Stevenson

This document is provided “as-is.” Information and views expressed in this document, including URL and other Internet web site references, may change without notice. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2017 Microsoft Corporation. All rights reserved.

# Executive Summary

This document presents guidance on rapidly identifying and removing Transport Layer Security (TLS) protocol version 1.0 dependencies in software built on top of Microsoft operating systems. It is intended to be used as a starting point for building a migration plan to a TLS 1.2+ network environment. While the solutions discussed here may carry over and help with removing TLS 1.0 usage in non-Microsoft operating systems or crypto libraries, they are not a focus of this document.

TLS 1.0 is a security protocol first defined in 1999 for establishing encryption channels over computer networks. Microsoft has supported this protocol since Windows XP/Server 2003. While no longer the default security protocol in use by modern OSes, TLS 1.0 is still supported for backwards compatibility. Evolving regulatory requirements as well as new security vulnerabilities in TLS 1.0 provide corporations with the incentive to disable TLS 1.0 entirely.

Microsoft recommends customers get ahead of this issue by removing TLS 1.0 dependencies in their environments and disabling TLS 1.0 at the operating system level where possible. Given the length of time TLS 1.0 has been supported by the software industry, it is highly recommended that any TLS 1.0 deprecation plan include the following:

* Code analysis to find/fix hardcoded instances of TLS 1.0 (or instances of older TLS/SSL versions).
* Network endpoint scanning and traffic analysis to identify operating systems using TLS 1.0 or older protocols.
* Full regression testing through your entire application stack with TLS 1.0 disabled.
* Migration of legacy operating systems and development libraries/frameworks to versions capable of negotiating TLS 1.2.
* Compatibility testing across operating systems used by your business to identify any TLS 1.2 support issues.
* Coordination with your own business partners and customers to notify them of your move to deprecate TLS 1.0.
* Understanding which clients may not interoperate by disabling TLS 1.0

The goal of this document is to provide recommendations which can help remove technical blockers to disabling TLS 1.0 while at the same time increasing visibility into the impact of this change to your own customers. Completing such investigations can help reduce the business impact of the next security vulnerability in TLS 1.0. For the purposes of this document, references to the deprecation of TLS 1.0 also include TLS 1.1.

# The Current State of Microsoft’s TLS 1.0 implementation

[Microsoft’s TLS 1.0 implementation](https://support.microsoft.com/en-us/kb/3117336) is free of known security vulnerabilities. Due to the potential for future [protocol downgrade attacks](https://www.openssl.org/~bodo/ssl-poodle.pdf) and other TLS 1.0 vulnerabilities not specific to Microsoft’s implementation, it is recommended that dependencies on all security protocols older than TLS 1.2 be removed where possible (TLS 1.1/1.0/ SSLv3/SSLv2).

In planning for this migration to TLS 1.2+, developers and system administrators should be aware of the potential for protocol version hardcoding[[1]](#footnote-1) in applications developed by their employees and partners. Protocol version hardcoding was commonplace in the past for testing and supportability purposes as many different browsers and operating systems had varying levels of TLS support.

# Ensuring support for TLS 1.2 across deployed operating systems

Many operating systems have outdated TLS version defaults or support ceilings that need to be accounted for. Usage of Windows 8/Server 2012 or later means that TLS 1.2 will be the default security protocol version:

#### Figure 1: Security Protocol Support by OS Version

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Windows OS | SSLv2 | SSLv3 | TLS 1.0 | TLS 1.1 | TLS 1.2 |
| Windows Vista | Enabled | Enabled | **Default** | Not Supported | Not Supported |
| Windows Server 2008 | Enabled | Enabled | **Default** | [Disabled](https://cloudblogs.microsoft.com/microsoftsecure/2017/07/20/tls-1-2-support-added-to-windows-server-2008/) | [Disabled](https://cloudblogs.microsoft.com/microsoftsecure/2017/07/20/tls-1-2-support-added-to-windows-server-2008/) |
| Windows 7 (WS2008 R2) | Enabled | Enabled | **Default** | [Disabled](https://support.microsoft.com/en-us/help/3140245/update-to-enable-tls-1-1-and-tls-1-2-as-a-default-secure-protocols-in) | [Disabled](https://support.microsoft.com/en-us/help/3140245/update-to-enable-tls-1-1-and-tls-1-2-as-a-default-secure-protocols-in) |
| Windows 8 (WS2012) | Disabled | Enabled | Enabled | Enabled | **Default** |
| Windows 8.1 (WS2012 R2) | Disabled | Enabled | Enabled | Enabled | **Default** |
| Windows 10 | Disabled | Enabled | Enabled | Enabled | **Default** |
| Windows Server 2016 | Not Supported | Disabled | Enabled | Enabled | **Default** |

A quick way to determine what TLS version will be requested by various clients when connecting to your online services is by referring to the Handshake Simulation at [Qualys SSL Labs](https://www.ssllabs.com/). This simulation covers client OS/browser combinations across manufacturers. See [Appendix A](#_Appendix_A:_Handshake) at the end of this document for a detailed example showing the TLS protocol versions negotiated by various simulated client OS/browser combinations when connecting to [www.microsoft.com](http://www.microsoft.com).

If not already complete, it is highly recommended to conduct an inventory of operating systems used by your enterprise, customers and partners (the latter two via outreach/communication or at least HTTP User-Agent string collection). This inventory can be further supplemented by traffic analysis at your enterprise network edge. In such a situation, traffic analysis will yield the TLS versions successfully negotiated by customers/partners connecting to your services, but the traffic itself will remain encrypted.

# Finding and fixing TLS 1.0 dependencies in code

For products using the Windows OS-provided cryptography libraries and security protocols, the following steps should help identify any hardcoded TLS 1.0 usage in your applications:

1. Identify all instances of [AcquireCredentialsHandle](https://msdn.microsoft.com/en-us/library/windows/desktop/aa374712(v=vs.85).aspx)(). This helps reviewers get closer proximity to code blocks where TLS may be hardcoded.
2. Review any instances of the [SecPkgContext\_SupportedProtocols](https://msdn.microsoft.com/en-us/library/windows/desktop/aa380103(v=vs.85).aspx) and [SecPkgContext\_ConnectionInfo](https://msdn.microsoft.com/en-us/library/windows/desktop/aa379819(v=vs.85).aspx) structures for hardcoded TLS.
3. In native code, set any non-zero assignments of [grbitEnabledProtocols](https://msdn.microsoft.com/en-us/library/windows/desktop/aa379810(v=vs.85).aspx) to zero. This allows the operating system to use its default TLS version.
4. Disable [FIPS Mode](https://blogs.technet.microsoft.com/secguide/2014/04/07/why-were-not-recommending-fips-mode-anymore/) if it is enabled due to the potential for conflict with settings required for explicitly disabling TLS 1.0/1.1 in this document. See [Appendix B](#_Appendix_B:_Disabling) for more information.
5. Update and recompile any applications using WinHTTP hosted on Server 2012 or older.
   1. Applications must add code to support TLS 1.2 via [WinHttpSetOption](https://msdn.microsoft.com/en-us/library/windows/desktop/aa384114(v=vs.85).aspx)
6. To cover all the bases, scan source code and online service configuration files for the patterns below corresponding to enumerated type values commonly used in TLS hardcoding:
   1. SecurityProtocolType
   2. SSLv2, SSLv23, SSLv3, TLS1, TLS 10, TLS11
   3. WINHTTP\_FLAG\_SECURE\_PROTOCOL\_
   4. SP\_PROT\_
   5. NSStreamSocketSecurityLevel
   6. PROTOCOL\_SSL or PROTOCOL\_TLS

The recommended solution in all cases above is to remove the hardcoded protocol version selection and defer to the operating system default. Operating systems which do not support TLS 1.2 as the default should be upgraded to versions which do.

# Testing with TLS 1.2+

Following the fixes recommended in the section above, products should be regression-tested for protocol negotiation errors and compatibility with other operating systems in your enterprise.

* The most common issue in this regression testing will be a TLS negotiation failure due to a client connection attempt from an operating system or browser that does not support TLS 1.2.
  + For example, a Vista client will fail to negotiate TLS with a server configured for TLS 1.2+ as Vista’s maximum supported TLS version is 1.0. That client should be either upgraded or decommissioned in a TLS 1.2+ environment.
* Products using certificate-based Mutual TLS authentication may require additional regression testing as the certificate-selection code associated with TLS 1.0 was less expressive than that for TLS 1.2.
  + If a product negotiates MTLS with a certificate from a non-standard location (outside of the standard named certificate stores in Windows), then that code may need updating to ensure the certificate is acquired correctly.
* Service interdependencies should be reviewed for trouble spots.
  + Any services which interoperate with 3rd-party services should conduct additional interop testing with those 3rd parties.
  + Any non-Windows applications or server operating systems in use require investigation / confirmation that they can support TLS 1.2. Scanning is the easiest way to determine this.

A simple blueprint for testing these changes in an online service consists of the following:

1. Conduct a scan of production environment systems to identify operating systems which do not support TLS 1.2.
2. Scan source code and online service configuration files for hardcoded TLS as described in “[Finding and fixing TLS 1.0 dependencies in code](#_Finding_and_fixing)”
3. Update/recompile applications as required:
   1. Managed apps
      1. Rebuild against the latest .NET Framework version.
      2. Verify any usage of the [SSLProtocols](https://msdn.microsoft.com/en-us/library/system.security.authentication.sslprotocols(v=vs.110).aspx) enumeration is set to SSLProtocols.None in order to use OS default settings.
   2. WinHTTP apps – rebuild with [WinHttpSetOption](https://msdn.microsoft.com/en-us/library/windows/desktop/aa384114(v=vs.85).aspx) to support TLS 1.2
4. Start testing in a pre-production or staging environment with all security protocols older than TLS 1.2 disabled [via registry](https://support.microsoft.com/en-us/help/245030/how-to-restrict-the-use-of-certain-cryptographic-algorithms-and-protocols-in-schannel.dll).
5. Fix any remaining instances of TLS hardcoding as they are encountered in testing. Redeploy the software and perform a new regression test run.

# Notifying partners of your TLS 1.0 deprecation plans

After TLS hardcoding is addressed and operating system/development framework updates are completed, should you opt to deprecate TLS 1.0 it will be necessary to coordinate with customers and partners:

* Early partner/customer outreach is essential to a successful TLS 1.0 deprecation rollout. At a minimum this should consist of blog postings, whitepapers or other web content.
* Partners each need to evaluate their own TLS 1.2 readiness through the operating system/code scanning/regression testing initiatives described in above sections.

# Conclusion

Removing TLS 1.0 dependencies is a complicated issue to drive end to end. Microsoft and industry partners are taking action on this today to ensure our entire product stack is more secure by default, from our OS components and development frameworks up to the applications/services built on top of them. Following the recommendations made in this document will help your enterprise chart the right course and know what challenges to expect. It will also help your own customers become more prepared for the transition.

# Appendix A: Handshake Simulation for various clients connecting to [www.microsoft.com](http://www.microsoft.com), courtesy SSLLabs.com



# Appendix B: Deprecating TLS 1.0/1.1 while retaining FIPS Mode

Follow the steps below if your network requires FIPS Mode but you also want to deprecate TLS 1.0/1.1:

1. Configure TLS versions [via the registry](https://support.microsoft.com/en-us/help/245030/how-to-restrict-the-use-of-certain-cryptographic-algorithms-and-protocols-in-schannel.dll), by setting “Enabled” to zero for the unwanted TLS versions.
2. Disable Curve 25519 (Server 2016 only) via Group Policy.
3. Disable any cipher suites using algorithms that aren’t allowed by the relevant FIPS publication. For Server 2016 (assuming the default settings are in effect) this is means disabling RC4, PSK and NULL ciphers.

1. Hardcoding here means that the TLS version is fixed to a version that is outdated and less secure than newer versions. TLS versions newer than the hardcoded version cannot be used without modifying the program in question. This class of problem cannot be addressed without source code changes and software update deployment. [↑](#footnote-ref-1)