The Risks of SSL Inspection

By Will Dormann

Published: 2015-03-13 · Archived: 2026-04-06 00:13:11 UTC

Recently, SuperFish and PrivDog have received some attention because of the risks that they both introduced to customers because of implementation flaws. Looking closer into these types of applications with my trusty CERT Tapioca VM at hand, I've come to realize a few things.

In this blog post, I will explain

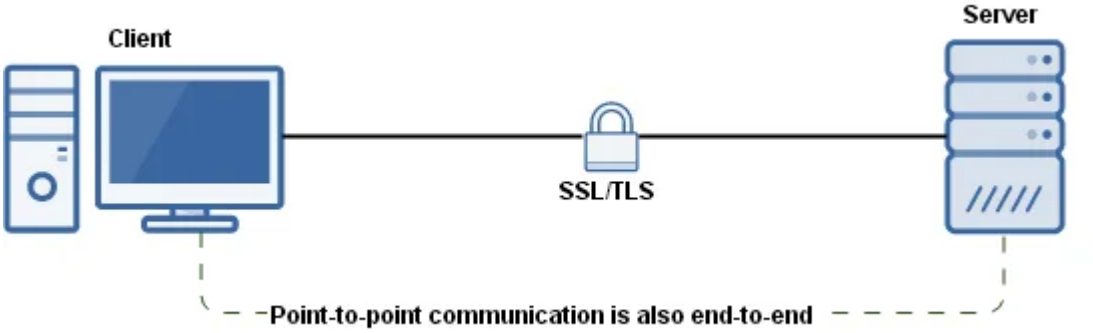
1. The capabilities of SSL and TLS are not well understood by many.
2. SSL inspection is much more widespread than I suspected.
3. Many applications that perform SSL inspection have flaws that put users at increased risk.
4. Even if SSL inspection were performed at least as well as the browsers do, the risk introduced to users is not zero.

Background

SSL and TLS are used for two primary purposes:

1. Authentication of the server that a client is communicating with.
2. Encrypting data sent between the client and the server.

Some folks may assume that SSL or TLS attempt to achieve the above goals on an end-to-end basis. This is an invalid assumption. SSL and TLS can practically achieve secure authentication and encryption only on a point-to-point basis, not an end-to-end basis. What is the difference, you may ask. Let's first look at the case where the point-to-point communication is also end-to-end:



The client has the ability to verify secure communications **only** with the next point that it is communicating with. In the above case, the client establishes an SSL or TLS connection directly with the target system. The client software verifies that the system that it is connected to has been verified by one of the potentially hundreds of root

CA certificates that may be present on the system. Whether or not the system that is being communicated with is what the user expects is another story.

When a user loads a website in his/her browser that should be protected with HTTPS (and subsequently SSL or TLS), that browser is actually only confirming that the system it is communicating with is providing a certificate that is issued by one of the root CAs that the browser trusts. You are implicitly trusting that each root CA has done its due diligence to verify the identity of the server that you are connecting to. Sometimes root CAs get tricked. You are also trusting that each root CA has taken the appropriate steps to protect its own systems. Sometimes root CAs get compromised. See Moxie Marlinspike's blog post, SSL And The Future Of Authenticity, for more details about these problems.

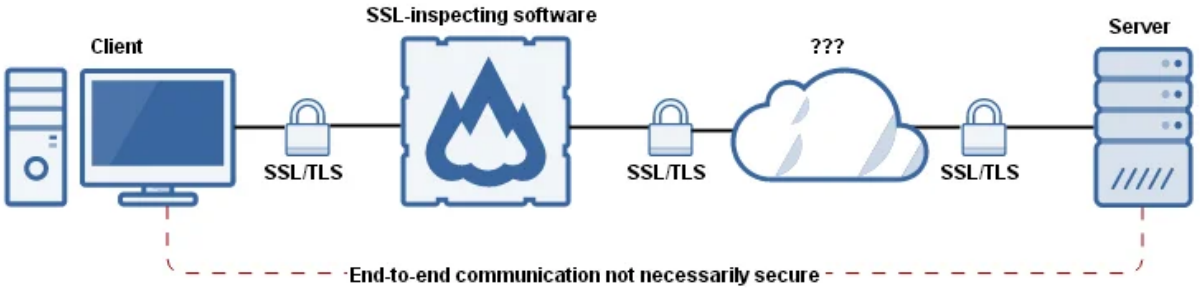
Consider the case of the recent SuperFish and PrivDog vulnerabilities. Those applications install a new trusted root CA certificate. Note the specific use of the term "trusted" here. We're not talking about anything along the lines of the web of trust used by the PGP and GPG applications, where the user explicitly indicates a level of trust with individual keys. A "trusted root CA certificate" is simply one that can be used by client software without generating warnings. It has nothing to do with the "trust" that is the confidence a user has in something.

Luckily for the end user, both SuperFish and PrivDog install trusted root CA certificates that have clear indications that they are from SuperFish and Privdog, respectively. However, there is absolutely nothing preventing an application from installing a certificate with a more deceptive name, such as "VeriSign."

Now, consider the case where a user with PrivDog or Superfish installed is visiting a site over HTTPS because he/she is providing sensitive information, such as a username and password. If his/her browser does not complain about the certificate, **at best** that means that whatever system he/she is talking to was issued by one of the trusted root CAs, including SuperFish or Privdog.

If somebody wants to verify that a certificate is issued by a "real" certificate authority, for example by checking the issuer's thumbprint, the steps required to do so can vary from browser to browser. Some browsers, such as Microsoft Internet Explorer, don't even allow viewing a certificate until after you have accepted it, which can make it tricky for even a security professional to determine if he/she is connected to the machine that he/she thinks.

Host-level software like Superfish or Privdog is just the beginning, though. As it turns out, there are plenty of network-level applications, devices, and appliances that do SSL inspection. Secure web gateways, firewalls, data loss prevention (DLP) products, and other applications all seem to have jumped on the SSL inspection bandwagon. Sure, there may be reasons why a network administrator may want to look into traffic that should be protected by SSL or TLS, but what people may not realize are the security impacts of deploying software that does not do SSL inspection at least as well as the browsers do. Consider the following diagram:



In the above scenario, the client can verify **only** that it is communicating with the SSL-inspecting software. The client is unaware of what technique the SSL-inspecting software is using for validating SSL certificates. And perhaps more importantly, whether there are additional points between the SSL-inspecting software and the target system is impossible for the client to determine. Is there an attacker between the SSL-inspecting software and the target server? The client has no way of knowing. Because of this lack of transparency, the client must assume that the SSL inspecting software is doing everything perfectly. Unfortunately, SSL-inspecting software does not do everything perfectly.

Common SSL Mistakes

In our analysis of software that performs SSL inspection, we have observed SSL inspection software make a variety of mistakes:

1) Incomplete validation of upstream certificate validity
Some SSL-inspecting software fails to validate the certificates of systems that it connects to. In some cases, the software may attempt to perform some validation of the certificate, but the validation may be insufficient.

Risks: Clients cannot know if they are connected to a legitimate site or not.

2) Not conveying validation of upstream certificate to the client
In some cases, the SSL inspection software does perform validation of upstream certificates, but it does not relay the results of the validation to the client.

Risks: Clients cannot know if they are connected to a legitimate site or not.

3) Overloading of certificate Canonical Name (CN) field
Some SSL inspecting software attempts to relay the validity of the upstream certificate to the client by way of the CN field of the certificate. For example, Komodia SSL Digestor changes the CN to begin with "verify_fail" if the server-provided certificate is invalid for any reason. There are a number of issues with this technique. The most obvious mistake being that the actual error conveyed to the user usually has nothing to do with why it failed.

Risks: Users of client systems may not know why certificate validation failed, or even if it failed. An attacker may be able to specify a Subject Alternative Name field to specify any domain that the certificate specifies it is valid for, which results in a browser that does not display a certificate warning.

4) Use of the application layer to convey certificate validity
To relay the validity of the certificate to the client, some SSL inspectors provide web content (e.g. HTML) to the

client, describing what is wrong with the certificate. The normal mechanisms through which client software ascertains and displays certificate validity may still indicate that the certificate is fine.

Risks: Not everything that accesses data using HTTPS is a human using a web browser. For example, the client may be an application that communicates with a server using JSON data. This flaw also causes inconsistent use of SSL validity indicators (e.g., "Where do I look for the padlock again?").

5) Use of a User-Agent HTTP header to determine when to validate a certificate
Some software will selectively decide when to validate upstream certificates based on the User-Agent HTTP header provided by the client. This technique is likely used in conjunction with the technique described above where certificate validity is conveyed in the application layer.

Risks: Not every web client may receive certificate validation. Various web browser versions and non-browser software may be exempt from validation.

6) Communication before warning
Upon detecting a certificate error, some SSL inspection applications will send the client's request to the server prior to presenting a warning to the user.

Risks: An attacker still may be able to view or modify sensitive data.

7) Same root CA certificate
Some SSL inspection applications use and install the same trusted root CA certificate for each installation of the application.

Risks: An attacker may be able to extract the private key from the software and sign all visited sites with the universally-trusted root CA certificate.

Consider the case where an SSL inspection application performs flawlessly. That is, it has none of the above seven flaws, or flaws similar to them. It validates the identity of the upstream server based on its own trusted root CA certificate store. However, the client has no visibility into what root CAs the SSL inspection software trusts, in particular if it's a separate device on the network. The use of SSL inspection software reduces or completely prevents clients from successfully validating the identity of the servers that they are communicating with.

Given the architecture of SSL and TLS, users have a difficult enough time making a security decision based on the information provided to them. Once the concept of SSL inspection is thrown into the mix, it only makes the decision more difficult.

Potentially Affected Software

By performing a web search for "ssl inspection," we were able to construct a list of applications that **may** be affected by a number of the above-outlined vulnerabilities. However, due to time and resource constraints, we have not been able to acquire or test a number of these applications, so we are soliciting feedback from the community to ascertain their status. Just because a product is listed, does not necessarily mean that it contains any of the seven vulnerabilities above. The list of applications possibly affected includes the following:

1. A10 vThunder

2. Arbor Networks Pravail
3. Baracuda Web Filter
4. BASCOM School Web Filter
5. Bloxx Web Filter
6. Blue Coat SSL Visibility Appliance
7. Check Point Data Loss Prevention (DLP), Anti Virus, Anti-Bot, Application Control, URL Filtering, Threat Emulation and IPS.
8. Cisco ScanCenter
9. Citrix NetScaler AppFirewall
10. Clearswift SECURE Web Gateway
11. ContentKeeper
12. Cymphonix Internet Management Suite
13. Dell SonicWALL
14. EdgeWave iPrism Web Security
15. ESET Smart Security
16. F5 BIG-IP
17. Fortinet FortiGate
18. Fidelis Security XPS
19. Finjan Vital Security (pdf)
20. GFI WebMonitor
21. GigaMon GigaSmart
22. IBM Security Network Protection
23. iboss Web Security
24. Imperva Incapsula
25. iSHERIFF Cloud Security
26. Juniper IDP devices
27. Kaspersky Anti-Virus
28. Komodia SSL Decoder
29. M86 Secure Web Gateway (pdf)
30. McAfee Web Gateway and Firewall Enterprise (pdf)
31. Microsoft Forefront TMG
32. NetNanny
33. NextGig Netronome
34. Optenet WebFilter (pdf)
35. Palo Alto PAN-OS
36. Panda Cloud Internet Protection
37. PrivDog
38. Radware AppXcel
39. SafeNet eSafe Web Security Gateway
40. Sangfor IAM (pdf)
41. Smoothwall Secure Web Gateway
42. Sophos Cyberoam

43. Sourcefire SSL Appliance
44. Squid
45. Symantec Web Gateway
46. Thomason Technologies Next Gen IPS
47. Trend Micro Deep Security (pdf)
48. Trustwave WebMarshal, Secure Web Gateway
49. Untangle NG Firewall
50. Venafi TrustAuthority
51. VSS Monitoring vInspector (pdf)
52. WatchGuard HTTPS Proxy
53. Wavecrest CyBlock
54. WebSense Content Gateway
55. WebTitan
56. Qbik WinGate
57. WolfSSL SSL Inspection
58. Zscaler
59. ZyXel Firewall

Again, just because a product is listed, does not mean that it is necessarily vulnerable.

Conclusion

SSL and TLS do not provide the level of end-to-end security that users may expect. Even in absence of SSL inspection, there are problems with how well browsers are conveying SSL information to users. The fact that "SSL inspection" is a phrase that exists, should be a blazing red flag that what you think SSL is doing for you is fundamentally broken. Compounding the problem are the mistakes that SSL inspection software authors are making.

System administrators may wish to reassess whether they want to deploy SSL inspection capabilities in their environment. CERT Tapioca can be used to verify that the SSL inspection solution being used is doing its due diligence to minimize the increased risk to the users. At the very least, system administrators could contact the vendors of SSL inspection software to have them confirm the proper configuration options and behaviors.

If you are aware of products that do SSL inspection but are not listed above, or if you have details about which products contain which vulnerabilities, please send us feedback.

Update (March 22, 2017)

US-CERT has published Technical Alert TA17-075A about this topic. One aspect of this situation that has changed since the first publication of this blog entry in 2015 is the existence of the badssl.com website. This site makes it easier to test the effects of HTTPS interception, and it also makes it possible to test HTTPS behaviors much more thoroughly than Tapioca allows. To test your own HTTPS inspection environment:

1. Visit www.badssl.com using a browser that has direct (non-intercepted) internet connectivity.

2. Visit each of the red test sites listed to confirm your browser's behavior. In many cases, a modern browser will refuse to display the red-linked sites.
3. Complete the same tests on the same browser version, but in a network configuration where HTTPS traffic is being intercepted.
4. Compare how the browser behaves in a configuration where HTTPS interception is performed against the behavior where the browser has a non-intercepted internet connection. If an intercepted browser allows connectivity to any of the tests that are blocked in non-intercepted mode, this is an indication of a weakness introduced by HTTPS interception.
