امنیت در تجارت الکترونیکی

Security of e_ commerce

اشرف جباری

پست الکترونیکی A-Jabary@yahoo.com

چکیده :

نجارت الکترونیکی استفاده ازکامپیوترهای یک یا چند شبکه برای مبادله اطلاعات تجاری بدون استفاده از کاغذ است که بیشتر با خرید و فروش اطلاعات ، کالاها و خدمات از طریق اینترنت در ارتباط است . اما مسایل و مشکلات متعددی نظیر فقدان سرویس زمانی ،و یا بی توجهی بعضی شرکتها امنیت در اینگونه سیستم ها را دچار مخاطره می کنند . در بحث امنیت در تجارت الکترونیکی دو پروتکل استاندارد و معروف به نامهای SSL و SET مطرح می شوند . *طراحان سایتهای سیستمهای تجارت الکترونیکی باید امنیت در کل این سیستمها را سرفصل کار خود قراردهند تا مقصد وهدف کاربران بدینوسیله به خوبی تامین شود*

کلمات کلیدی:

تجارت الکترونیکی ، امنیت ، مبادله الکترونیکی ، SSL، SET ، مشتری ، فروشنده ، تجارت در اینترنت

مقدمه :

تجارت الکترونیکی را می‌توان انجام هرگونه امور تجاری و بازرگانی به صورت online و از طریق شبکهٔ جهانی اینترنت بیان کرد. این تکنیک در سالهای اخیر در بستر اینترنت رشد فزاینده‌ای داشته است. با گذشت زمان عده بیشتری از افراد به این مقوله از علم تجارت علاقه نشان میدهند و میخواهند که به خیل عظیم افرادی بپیوندند که از این طریق به امور تجاری و بازرگانی خود میپردازند. این امور می‌توانند شامل خرید و فروش عمده یا خردهٔ کالاهای فیزیکی و غیرفیزیکی (نظیر اتومبیل و یا نرم‌افزارهای کامپیوتری)، ارایهٔ سرویس‌های مختلف به مشتریان و دیگر موارد تجاری باشد. در این تحقیق سعی شده به بحث در مورد امنیت در تجارت الکترونیکی پرداخته شود . ابتدا در فصل اول تعاریف و مفاهیمی کلی از تجارت الکترونیکی ارائه می شود . فصل دوم مفهوم امنیت در اینگونه سیستم ها مطرح شده ودر فصل سوم در مورد دو پروتکل استاندارد به نامهای SSL, SET بطور اجمالی بحث خواهد شد.

فصل اول : تجارت الکترونیکی

تجارت الکترونیکی عبارت از مبادله اطلاعات تجاری بدون استفاده از کاغذ است که در آن نوآوریها یی مانند مبادله الکترونیکی داده ها ، پست الکترونیکی ،تابلو اعلانات الکترونیک و سایر فن آوریهای مبتنی بر شبکه به کاربرده می شود . تجارت الکترونیک نه تنها عملیاتی را که در انجام معاملات بطور دستی و با استفاده از کاغذ صورت می گیرد به حالت خودکار در می آورد

بلکه سازمانها را یاری می کند که به یک محیط کاملاً الکترونیک قدم بگذارند و شیوه های کاری خود را تغییر دهند . در چند سال اخیر سازمانها ، استفاده از تجارت الکترونیک از طریق اینترنت را شروع کرده اند . اینترنت از لحاظ دیگری نیز موجب پیشرفت و رواج تجارت الکترونیک گردیده است و آن هزینه بالنسبه پایین استقاده از این شبکه در مقایسه با شبکه های خصوصی است . البته باید در نظر داشت که استاندارد های تجارت الکترونیک هنوز در دست تدوین و توسعه است . با این وجود این مبادله الکترونیکی داده ها که فن آوری شناخته شده تری است هنوز بخش برتر تجارت الکترونیک را تشکیل می دهد . مبادله الکترونیکی داده ها عبارت از مبادله اسناد و مدارک تجاری در قالبهای پیش ساخته و قابل پردازش با دستگاهها بین سازمانها از طریق شبکه های ارتباط کامپیوتری است . تجارت الکترونیکی موجب انقلاب دیگری نیز شده است که براساس آن نحوه خرید و فروش کالاها و خدمات توسط موسسات تجاری دچار تحول گردیده است . تجارت الکترونیک روشی است که بوسیله آن اطلاعات ،محصولات و خدمات از طریق شبکه های ارتباط کامپیوتری خرید ویا فروش می شود و کمک می کند تا تجارت سنتی با کاربرد شیوه های فعال و نوین انتقال و پردازش اطلاعات امکانپذیر گردد. زیرا در هر گونه فعالیت بازرگانی ، اطلاعات از اهمیت حیاتی و ویژه ای برخوردار است . در تجارت الکترونیک اطلاعات بصورت الکترونیکی و بطور خودکار از کامپیوتری به کامپیوتر دیگر منتقل می شود . صرفنظر از اینکه چه نوع کالا یا خدماتی مبادله می شود ،گرد آوری و بهره برداری و توزیع اطلاعات در همه رشته های بازرگانی مورد نیاز است .

مبادله الکترونیکی داده ها را می توان برای مخابره الکترونیکی مدارک و اسناد مانند سفارشات خرید ، فاکتور، اعلامیه حمل ، تاییدیه وصول کالا و سایر مکاتبات استاندارد بازرگانی بین طرفین تجاری استفاده نمود . این فن آوری را همچنین می توان برای ارسال اطلاعات مالی و پرداختهای الکترونیکی بکار برد . هنگامیکه داده ها وارد سیستم رایانه ای خریدار شده و بصورت الکترونیکی ارسال میشود عین همان داده ها بدون نیاز به کلید زدن با ورود مجدد ،وارد رایانه فروشنده نیز می گردد . این فرآیند را معمولاً مبادله الکترونیکی داده ها بصورت کاربرد به کاربرد می نامند . مبادله الکترونیکی داده ها را می توان با برنامه های کاربردی کاملا تلفیق نمود . این فرآیند ،جریان مستقیم داده ها بین طرفهای بازرگانی و همچنین کاربرد داده ها جهت مقاصد درون سازمانی در هر یک از طرفهای تجاری را میسر می سازد .

تجارت الکترونیکی به طور عام و مبادله الکترونیکی داده ها بطور خاص به عنوان ابزاری برای ایجاد تغییر در شیوه های عملیاتی سازمانها طراحی و پیش بینی شده اند . در این فرایند تنها

حذف معاملات کاغذی مطرح نیست بلکه همچنین ایجاد و تحول در نحوه انجام معاملات سازمانها یا طرفهای تجاری و نیز پاسخگویی به معاملات مبادله الکترونیکی داده ها نیز موردنظر می باشد و این خود موجب بازسازی فرایندهای درون سازمانی می شود . در حقیقت بالاترین سطح بهره وری و کارآیی زمانی حاصل می شود که این فن آوری پس از بررسیهای کامل و تجزیه و تحلیل . فرایندها بطور مداوم مورد بازنگری و بازسازی قرار می گیرند

فصل دوم : امنیت در تجارت الکترونیکی

با ظهور رایانه در صحنه فعالیت تجاری سازمانها ، علاوه بر شیوه های فیزیکی که سابقاً برای حفاظت از اطلاعات بکار می رفت ، ایمنی اطلاعات با ابزارهای خودکار تامین می شود . به موازات اینکه سیستم های رایانه ای سازمانها از طریق شبکه ها به یکدیگر متصل می شوند ، ایمنی اطلاعات یعنی حفاظت از داده ها در حین مخابره در شبکه های رایانه ای نیز ابعاد جدیدی به خود می گیرد . در بسیاری از نقاط دنیا کاربرد مبادله الکترونیکی داده ها بمنظور تبادل اسناد تجاری پیش ساخته شرکتها ،دائماً در حال گسترش است . اسنادی که بطور الکترونیکی مبادله می شوند شامل اسناد معاملات تجاری (مانند فاکتور ، دستور حمل ، استعلام قیمت) ، اسناد پرداخت (مانند دستور پرداخت ، حواله ، اعلامیه ،بستانکاری ،بدهکاری) ، مدارک حمل و نقل (مانند تائید ذخیره جا ، اعلام وضعیت جا) و اسناد قانونی (مانند اظهار نامه گمرکی) می باشند . در سازمانهایی که به تسهیلات مبادله الکترونیکی داده ها مجهز نیستند کاربرد رایانه در امور تجاری با نرم افزارهای مبادله الکترونیکی داده ها و رابط های ارتباطاتی ادغام شده تا راه حل کارآمدی برای فعالیت های بازرگانی ارائه شود . ایمنی فوق العاده اهمیت دارد و باید مطمئن بود که مبادلات الکترونیکی داده ها که اکنون جایگزین سیستم های کاغذی میشود نیز حداقل از همان سطح ایمنی نسبت به شیوه های سنتی اعمال می شد ، برخوردار گردد .

بسیاری از شرکتها تنها به ارائه نرم افزارهای کاربردی تجارت الکترونیک می اندیشند و امنیت در سیستم هایشان را نادیده می گیرند . اما مشاوران حرفه ای امنیت یا همان Netcraft با مطرح کردن بحث امنیت این نارسائیها را فاش می کنند . از تلاشهایی که توسط Netcraft صورت می گیرد شامل توضیح این مطلب به مشتری است که بسیاری از سیستم هایی که آنها از آن استفاده می کنند دارای نارسائیهای امنیتی زیادی می باشد . این ارزیابی ها اغلب می توانند روی آخرین صفحه پروژه پیدا شود .

گاهی حتی گسترش دهندگان مجرب موفق نمی شوند که سیاست کافی برای امنیت سرویسهای× Web-based بکار گیرند . اگر سیستم عامل سرویس دهنده Web ، صفحات HTML ، فرمهای ورود داده ، بانکهای اطلاعاتی و ... را دسته بندی کند دوره کاربردی مقدم بر Web-site را تا زمان رفع اشکالات ایجاد شده بوسیله طراحان بکار می بریم . نباید بدون توجه به امنیت در کل سیستم امنیت یک قسمت را بررسی کرد . واضح است که هر سیستم کامپیوتری که به اینترنت متصل می شود با هر ویژگی که داشته باشد برای تمام علاقمندان در دسترس است . این در همه جا بودن ویژگی اصلی اینترنت است و این یک فرصت بزرگ تجارت و ترقی را در بازار جهانی فراهم می کند . بنابراین لازم است در صورت فقدان تجربه کافی ، یک شرکت مشاور امنیتی حرفه ای را در طول طراحی سیستم استخدام کنید و در طول کار گسترش سیستم در کنار آنها باشید تا مطمئن شوید که سرمایه شما در سرویسهای امنیتی محافظت می شود .

در چند سال گذشته Netcraft سرویسهای آنالیز امنیت خود را گسترش داده که در سطحی ابتدایی شامل تعیین سرویس دهنده های معیوب مشترک و در سطحی وسیعتر شامل پیدا کردن روزنه های موجود در طراحی و پیاده سازی با پیکره بندی نرم افزارهای کاربردی Web می شود . سرویس اولیه به سازمانها محیطهای تجارت الکترونیک ایمن بر اساس استفاده از پروتکل SSI برای سری کردن انتقال اطلاعات بین مشتری و نرم افزارهای کاربردی پیشنهاد کرده است به شرط اینکه سازمانها تعیین کنند که این امنیت در نقل و انتقال اطلاعات برایشان مهم است . در صورت عدم پیروی از گامهای اولیه برای ایمن سازی سرویستان ، در حقیقت خود را در معرض حمله های متنوع از طرف کاربران جزئی قرار داده اید .فراموش نکنید که بایستی دسترسی به تمام سرویسهارا از بین ببرید و فقط به سرویسهای موردنیاز و با عملکردهای صحیح اجازه دسترسی دهید و برای اطمینان از اینکه سرویسهای در دسترس با یک سطح پذیرفته شده امنیت مطابقت می کنند باید آنها را امتحان کنید .

مشکلی که معمولاً اتفاق می افتد ، خود استانداردهایی است که بصورت صف در آمده اند و به سادگی اجازه می دهند فایلها و کاربردهای نمونه روی یک Server نصب شود . Server ها معمولاً شامل کدی هستند که برای نمایش توانائیهای بیشتر و پیشرفته تر یک سیستم طراحی شده ولی اجازه مزاحمت هم خواهد داد و این امر امنیت سیستم را دچار مخاطره خواهد کرد .

تعداد زیادی از کاربردهای Web-based ها روی سیستم حقیقی توسعه یافته اند که در هنگام آماده شدن پروژه به درون محصولات خواهند رفت اما دو خطر وجود دارد یکی اینکه توسعه دهندگان روی سیستمی کار می کنند که طراحیش به این صورت است که در طول زمان توسعه

ایمن باشد و امکان دارد به آسانی یک سرویس FTP باز ،وسایل و ابزار Front page، Telnet و ... را داشته باشد و حتی ممکن است بوسیله ماشینهای دیگر در شبکه راه اندازی شده باشد . برای آوردن محصول به بازار ، اغلب وجود بعضی سرویسها مورد نظارت قرار می گیرد و آن سرویسها برای همه قابل دسترس خواهد بود . بنابراین وقتی توسعه کامل شد باید از فایلها پشتیبان تهیه شود و بقیه فایلها در طول زمان توسعه استفاده می شوند و داده های ضروری از شاخه Web روی سرویس دهنده منتقل می شوند . سپس تعدادی از.script Haker ها برای شناسایی فایلهای پشتیبان و برای بهبود کدهای مبدا تست می شوند .

نکته مهم اینکه از داده های حساس در سیستم هایon line حتماً باید محافظت کرد . بنابراین Netcraft اغلب سایتها را تست می کند تا ببیند اینکار انجام شده است یا نه ؟ در ضمن باید جلسه خود را مدیریت کنید به این ترتیب که گزارشات ماههای اخیر در مورد تحویل اشتباه یک Web-site به کاربر بررسی شده و در هر مورد دلیل این اشتباه را به نرم افزار glitch یا تابع اشتباه نسبت می دهند . هر چند بسیاری از اشتباهات به علت نقض های اساسی در طراحی کاربرد است . طراحی کاربردهای چند کاربره همیشه مبارزات سختی را برای توسعه دهندگان سیستم های نرم افزاری ترتیب می دهند و به تجربه و نگهداری نیاز دارد.

× web-based: پروتکلی که یک مرورگر web را مستقیماً به دستگاه یا برنامه کاربردیی مرتبط می کند که بر یک شبکه نظارت دارد.

مسئله دیگر اینکه شکسته شدن یک سرویس Web-based تعدادی زیر رویه خودکار را برای جستجو و تست سیستم ها و سرویسهای در معرض آسیب درگیر می کند و غالباً رمز شکسته شدن قفل یک سیستم در جاهای دور از انتظار قرار می گیرد. توضیحات صفحات HTML اطلاعات مفیدی را برای کسانیکه می خواهند امنیت سیستم را به خطر بیاندازند فراهم می کند . دیگر اطلاعات مفید نه تنها شامل ورودی های× Whois مشاهده شده ، فشردن ، رهاکردن و ... بلکه Web-sit مشترک و صفحات Web شخصی را هم در بر می گیرد . بنابراین کوتاهی در فکر کردن به راههایی که می توانند سیستم شما را مورد سوء استفاده قراردهند راه آسانی را برای حمله کنندگان به سیستم فراهم می کند . اشتباهات دیگر کوتاهی در تصحیح داده معتبر یک کاربر نهایی و سپس استفاده از این داده آسیب دیده در کاربرد است .

ظهور تجارت الکترونیک شامل مبادله الکترونیکی داده ها ،پست الکترونیکی ، خدمات تابلو اعلانات ، بازاریابی از راه دور و سیستم های تدارکاتی روی خط ایجاب می کند که برای مخابره و دریافت اسناد تجاری و سایر اطلاعات طبقه بندی شده ،سکوهای مورداعتمادی فراهم شود . بطور کلی سیستم جدید در عین حالی که دقت و سرعت پردازش را بطور قابل ملاحظه ای بالا برده است ،باید همچنین پاسخگوی مسائلی باشد که در زیر ، همراه با مخابرات بالقوه و تدابیر مقابله با آنها توضیح داده می شود:

1- قابلیت دسترسی :

باید دسترسی و قابلیت حصول داده ها در زمان و مکان مناسب را همراه با مصونیت از دسترسی غیر مجاز به داده ها تامین کند.

- خطرات :

عبارت از خطای شبکه ، قطع برق ، اشتباهات عملیاتی ،اشتباهات کاربردی ،خطای سخت افزار و نرم افزار سیستم و
ویروسها می باشد .

- تدابیر مقابله :

عبارت از انتخاب مسیرهای ارتباطی جایگزین ،جلوگیری از قطع برق ، آزمایش کیفیت نرم افزارها و سخت افزارها، محدود ساختن دسترسی و تامین سیستم پشتیبانی داده ها .

2- محرمانه بودن :

حفاظت پیامها در مقابل سوء استفاده ، رهگیری و استراق سمع .

- خطرات :

دسترسی غیر مجاز بوسیله افراد درون سازمان ،مزدوران یا بوسیله رهگیری حین مخابره.

- تدابیر مقابله :

رمز نگاری پیام ها.

3- تمامیت :

جلوگیری از دخل و تصرف یا حذف ناخواسته پیام . این موضوع همچنین شامل تمامیت توالی برای جلوگیری از تکرار و نیز از دست رفتن پیام می شود .

- خطرات :

بروز اشتباهات تصادفی یا ناشی از تقلب در مرحله وارد کردن داده ها ،تخریب خروجی .

- تدابیر مقابله :

تایید انتها به انتهای پیام ،استفاده از توالی پیام .

4- اعتبار و انکار ناپذیری پیام :

Whois× : در سرویس‌های خدمات اینترنتی کاربر را قادر می‌سازد که آدرس‌های e-mail و دیگر اطلاعات مربوط به کاربران لیست شده در بانک اطلاعاتی آن حوزه را پیدا می‌کند

تامین اطمینان از هویت فرستنده و گیرنده و امکان اثبات مخابره و وصول پیام .

-خطرات :

جعل هویت .

- تدابیر مقابله :

تایید اصالت پیام بوسیله ترکیبی از آنچه کاربر می داند ، آنچه که کاربر در اختیار دارد و یا ویژگیهای فیزیکی کاربر.

5- قابلیت بازرسی و رسیدگی :

ثبت داده های بازرسی بر اساس شرایط از پیش تعیین شده ، محرمانه بودن و تمامیت .

-خطرات و تدابیر مقابله :

خطرات و راههای مقابله با آنها همان است که در بالا در مورد محرمانه بودن و تمامیت گفته شد .

سطوح ایمنی اطلاعات :

ایمنی اطلاعات باید در سه سطح مختلف زیر مورد بررسی قرارگیرد :

-کاربرد:

کاربرد مسئله درون سازمانی شرکتهاست و حق دسترسی به سطوح مختلف اطلاعات بوسیله کاربردهای مختلف را تعریف و تبیین می کند .

- شبکه :

حصول اطمینان از اینکه تسهیلات شبکه ای فقط بوسیله کاربران مجاز و سازمان مجاز قابل دسترسی است .

- پیام :

حصول اطمینان از اینکه یک نسخه مجاز پیام (و فقط یک نسخه)فرستاده می شود و این یک نسخه نیز تنها بوسیله گیرنده موردنظر دریافت می شود .

فصل سوم : پروتکلهای امنیت در تجارت الکترونیک

1- پروتکل SET

پروتکل SET که بوسیله ویزا و کارت های اصلی توسعه داده شده است قصد داشت رشد روزافزون پرداخت توسط کارت های کردیت در اینترنت را امن کند . نسخه 5/1 آن در سال 1997 شناخته شد. این فصل مروری بر ویژگیها و ملزومات ، قوانین و خصوصیات پروتکلهای امنیت می باشد

1-1-ویژگیهای پروتکل SET

1-1-1- رمز گذاری :

یکی از مهمترین ویژگیهای پروتکل SET رمزگذاری است که یک فرآیند ریاضی می باشد و می تواند یک پیغام رمز را از یک پیغام اصلی بدست آورد .

2-1-1- امضای الکترونیکی:

هدف از امضای الکترونیکی اثبات هویت خواننده پیغام است . در پروتکل SET امضا امنیت بیشتری به سیستم می دهد . گیرنده متنهای دیجیتالی را با کلید خارجی علامت می زند و همچنین می تواند بوسیله کلید متقابل به خواننده اعتبار دهد . این امضا ها تمامیت یک پیغام را برای بازبینی مجاز میشمرد.

3-1-1- ضمانت نامه ها :

ضمانت نامه بخشی از یک سند رسمی مطمئن است . در اینجا یک کلید عمومی به یک شخص بخصوص تعلق دارد و دارای اطلاعاتی مثل هویت مالک ضمانت نامه ، کلید عمومی مالک ضمانت نامه ، هویت صادرکننده ضمانت نامه ، دوره اعتبار ضمانت نامه و امضای صادرکننده ضمانت نامه می باشد . . اگر یک شخص یک متن علامت‌دار را پیشاپیش بفرستد ضمانتنامه پیش‌بینی می‌کند که او مالک کلید دوتایی است. پس گیرنده می‌تواند از کلید امنیتی عمومی گیرنده استفاده کند تا اینکه بازبینی امضا، روی سند را انجام دهد.

2-1- ملزومات پروتکل SET

1-2-1- مشتری :

اولین نیاز مشتری ، چگونگی پرداخت مبلغ SET و داشتن یک نمایشگر استاندارد و یک کارت اعتباری مخصوص Set.wallet است . و یک ضمانتنامه را صدا می زند و در این محیط کاربر توسعه می یابد .

ضمانت نامه کاربر SET به یک زوج کلید احتیاج دارد که کلید خارجی اجازه می دهد که مستندات را به صورت دیجیتالی علامت بزند و کلید عمومی به گیرنده اجازه بازبینی امضای پیوست سند رسیده از طرف کاربر را می دهد . ضمانت نامه SET توسط شرکتی که کارت اعتباری مشتری را صادر می کند ، ارسال می شود و برای پرداخت SET یک ضمانت نامه برای هر کارت اعتباری درخواست خواهد شد . بعد از رمزدهی اعداد کارت اعتباری مشتری ، کارت مشتری شامل این اعداد وکلید عمومی دارنده کارت در ضمانت نامه SET صادر می شود .پس اعداد رمز تنها توسط ساختار کارت اعتباری قابل رمزگشایی است و نهایتاً علامت ضمانت نامه SET ارسال می شود . صادرکنندگان ضمانت نامه ها ، کلید عمومی را متعلق به مشتری می کنند و ضمانت می کنند که کاربر اجازه خواهد یافت از طریق این کارت اعتباری خرید کند . مورد دیگر Set.wallet است که کلید رمز و ضمانت نامه دارنده کارت اعتباری را ذخیره می کند و مشتری را مجاز به پرداخت وجه SET می نماید .

2-2-1- بازرگان :

بازرگان به یک ضمانت نامه نرم افزار مخصوص برای فروش خوب و سرویس دهی در پروتکل SET نیاز دارد . شرکت مورد نظر وجه کارت اعتباری را پردازش می کند و یک گواهی برای تاجر تولید می کند و اعتبار تاجر در پرداخت SET در این گواهی استفاده شده می تواند

تهیه گردد. این گواهی ما را مطمئن می کند که تاجر اجـازه دارد اجـاره پرداخـت را دریافـت کنـد . مورد دیگر نرم افزار POS است که بازرگان را مجاز می کند پروتکل SET را پردازش نماید .

3-2-1- مدخل پرداخت :

اجازه پرداخت بین خریدار و تاجر در این مکان است و شـامل گواهینامـه کـه اعتبـار بانـک را تضمین می کند و مدخل پرداخت SET میباشد.

4-2-1- سیستم اعتباربخشی :

این سیستم آخرین کارش برای اعتبارپرداخت SET بین مشتری و تاجر را پردازش می کنـد . در اینجا قوانین یک پرداخت روی اینترنت وجود دارد .

5-2-1- خصوصیات :

پروتکل امنیت تجارت الکترونیکی دارای ویژگیهایی اسـت . یکـی از ایـن ویژگیهـا وجـود SET مطمئن است که از داده های سری مراقبـت مـی کنـد . جامعیـت داده هـا ، درسـتی همـه داده‌هـای پراکنده پرداخت تهیه شده بین اجزاء انتقال ، تشکر برای استفاده از امضاهای دیجیتـالی سـرانجام هر مشترک هر پیغام رسـیده از مشـترک بعـدی را اصـلی (درسـت) تعهـد میکنـد. همینطوربحـث اعتباربخشی مطرح است که با استفاده از امضاهای دیجیتالی به خریدار اجازه می دهند کـه تـاجر را اعتبار بخشد و یک تاجر خریدار خود را اعتبار بخشد . در خاتمه پروتکل SET یک سطح بالای امنیت برای همه کاربران خودش تهیه کرده است .یک تاجر دربرابر کلاهبرداری خریدار محافظـت شده است و یک خریدار هم در برابر شماره کارتهای اعتباری دزدیده شده بوسیله تاجر محافظـت شده است .

2- پروتکل SSL

هدف اولیه پروتکل SSL اینست که بین دو کاربر ارتبـاطی ، ارتبـاط قابـل اعتمـاد و محرمانـه فراهم آورد . این پروتکل از دو لایه تشکیل شده است که در پایین ترین سطح لایه لایه شده است . در بالای برخی از پروتکل های حمل و نقل مانند TCP ، پروتکـل SSL.RECORD وجـود دارد .

این پروتکل لایه های متفاوت سطح پروتکلها را کپسوله می کند . پروتکل مشابه دیگری به نام SSL.Handshake اجازه می دهد که سرویس دهنده و مشتری برای شناسایی یکدیگر و بحث ،یک الگوریتم رمز و کلیدهای رمز قبل از پروتکل متقاضی و اولین بایت داده ها را انتقال دهد و یا دریافت کند . مزیت SSL در اینست که پروتکل متقاضی مستقل است و پروتکل سطح بالاتر می تواند آشکارا در بالای پروتکل SSL لایه شود . پروتکل SSL امنیتی فراهم می کند که دارای ویژگیهای زیر است :

- ارتباط محرمانه است
- تشخیص همتا ،که می تواند با استفاده از نامتقارن بودن یا کلید عمومی رمز را شناسایی کند .
- ارتباط قابل اعتماد است .

2-1- اهداف پروتکل SSL

2-1-1- امنیت پنهان سازی :

SSL برای ایجاد یک ارتباط محفوظ بین دو طرف استفاده می شود .

2-1-2- توسعه پذیر بودن :

پیگردهای SSL محدوده ای را داخل کلید عمومی جدید فراهم می کند و روشهای رمز توده ای ضرورتاً تشکیل می شود . همچنین برای جلوگیری کردن ، احتیاج به ایجاد یک پروتکل جدید و اجتناب ، نیاز به اجرای یک کتابخانه امنیتی جدید را دارد .

2-1-3- کارایی نسبی :

عملیات رمزگذاری منجر به استفاده زیاد CPU می گردد . برای همین SSL تشکیل شده از یک طرح پنهانی جلسه اختیاری برای کاهش تعداد ارتباطاتی که احتیاج دارند از قسمت Scatch حافظه ارتباط برقرار کنند .

2-1-4- اهداف این سند :

پروتکل SSL نسخه Specification 3.0 در اصل برای خوانندگانی که پروتکل را اجرا می‌کنند و آنهایی که آنالیز رمز آنرا اجرا می‌کنند در نظرگرفته خواهد شد. مشخصات به این طریق در ذهن نوشته شده و برای انعکاس احتیاجات آن دوگروه در نظرگرفته می‌شود. به این دلیل تعدادی از ساختارهای داده‌ای وابسته به الگوریتم و قوانین درکالبد متن به جهت فراهم کردن دسترسی آسانتر به آنها در برگرفته میشود. (بعنوان نقطه مقابل در یک ضمیمه).

این مدارک نـه بـرای فـراهم کـردن جزئیـات تعیـین خـدمات و نـه تعیـین رابـط در نظرگرفتـه نمی‌شود، گرچه مناطق انتخابی سیاسی را می‌پوشاند، همانطورکه آنها برای حفـظ امنیـت اسـتوار درخواست می‌کنند.

5-1-2- زبان نمایش :

این سند به فرمت داده ها در یک نمایندگی خارجی رسیدگی می کند . از این به بعـد از دسـتور نمایش تعیین استفاده می شود . دسـتور زبـان چنـدین منبـع در سـاختارش رسـم مـی کنـد و در گرامرش شبیه زبان برنامه نویسی است .

- اندازه بلوک پایه :

نمایش همه اقلام داده‌ها صریحاً مشخص میشود. اندازه بلوک داده‌های اساسی یک بایت است .(یعنی 8 بیت) اقلام داده‌های بایتهای مضاعف از اتصال بایتها از چپ به راست، از بـالا بـه پـایین تشکیل شده‌اند.

- گوناگون

توضیحات با " ×/ " شروع وبا " /× " پایان می‌یابد .

اجزاء اختیاری توسط ضمیمه کردن آنها در پرانتز ایتالیک// دلالت داده می‌شود.

وجود بایتهای منفرد محتوی داده‌های غیر تفسیری از نوع opaque.

- یک بردار

آرایه یک بعدی یک رشته از عناصر داده‌های یکنواخت است. اندازه بردار ممکن است در زمان مستند مشخص شود یا تا زمان اجرا نامعین بماند، در هر مرحله در بردار، طـول، تعـداد بایتهـا را اعلان می‌کند نه تنها عناصر را.

دستور تشخیص یک نوع جدید Tاست که یک بردار با طول ثابت نوعT هست.

T T'[n]؛

در اینجا ً T، n بایت را در رشته داده‌ها اشغال می‌کند، جایی که n یـک مضـرب از انـدازهT اسـت. طول بردار در جریان کد دهی شامل نمی‌شود.

طول تغییرپذیر بردارها توسط تشخیص یک زیرمحدوده طولهای قانونی تعیین می‌شود، بطورکلی با استفاده از نماد floor..ceiling < > وقت کد دهی طول واقعـی مقـدم بـر محتویـات بـردار در جریان بایت است. طول به صورت یک عـدد مصـرف‌کننـده خواهـد بـود بصـورت تعـدادی بایـت درخواستی جهت نگهداری ماکزیمم طول تعیین شده بردار(ceiling) طول تغییرپذیر بردار بـا یـک طول واقعی نباید صفر، بعنوان یک بردار خالی، اشاره شده است.

T T'<floor..ceiling>؛

- اعداد

نوع داده‌های پایه (مبنا) یک بایت بدون علامت است0 (8 unit) همه انواع داده‌های عددی بزرگتر از طول ثابت سریهای بایتهای اتصال یافته همچنین بدون علامت هستند.

unit16 [2] ;unit8
unit24 [3] ;unit8
unit32 [4] ;unit8
unit64 [[8] ;unit8

- شمارشی‌ها :

یک نوع داده پراکنده اضافی که قابل دسترسی است enum نامیده می شود. یک فیلد نوع enum فقط مقادیری را که درهنگام تعریف تقبل کرده را می‌پذیرد. هر تعریفی یک نوع متفاوت است. فقط شمارشی‌های همان نوع ممکن است گمارده یا مقایسه شوند. هر جزئی از یک شمارش باید بصورت نمایش داده شده درمثال بعدی یک مقدار راتخصیص بدهد. ازآنجایی که اجزای شمارشی مرتب نمی‌شوند آنها میتوانند هرمقدار یونیک درهرترتیب را (سفارش) تخصیص دهند.

enum { $e_1(v_1), e_2(v_1), \ldots, e_n(v_N), [(n)]$ } T_e;

زمانی ممکن است یک مقدار را بدون دنباله پیوندی آن تعیین کند. این کار به منظور محصور کردن تعیین عرض بدون تعیین یک جزء زائد است.

برای شمارشی‌هایی که هرگز به نمایش خارجی تبدیل نمیشوند اطلاعات اعدادی ممکن است حذف شوند.

-انواع ساخته شده :

ساختارهای تعیین شده ممکن است از انواع اولیه به راحتی شناخته شود. هر مشخصه‌ای یک نوع جدید و یونیک را بیان می‌کند گرامر برای تعریف، مشابه گرامرC است.

```
struct {
      T1 f1;
       T2 f2;
       ...
      Tn fn;
} [T];
```

فیلدها در طی یک ساختار ممکن است با استفاده از نام نوع یک دستور مشابه همانی کـه قابـل دسترس برای شمارشـی‌هاسـت مشـخص شـوند. بـرای مثـال T.f2 بـه فیلـد دوم اظهـارات قبلـی برمی‌گردد. تعریف یک ساختار ممکن است درگیومه محصور شده باشد.

- واریانت‌ها (متغیرها)

ساختارهای تعیین شده ممکن است تغییراتی براساس برخـی دانـش‌هـا کـه قابـل دسـترس در داخل محیط است داشته باشند. انتخاب کننده باید یک نوع شمارشی باشـد کـه تغییـرات ممکـن در ساختار تعریف شده را تعیین می‌کند. آنها باید بازویی برای هـر جـزء شمارشـی اعـلان شـده در قسمت انتخاب باشند. بدنه ساختار متغییر ممکن است برچسبی را برای مراجعـه بدهـد مکـانیزمی که واریانت در زمان اجرا انتخاب میشود توسط زبان نمایش توصیف نمیشود.

```
struct {
        T1 f1;
        T2 f2;
.... Tn fn;
select (E) {
            case e1: Te1;
            case e2: Te2;
            ....
            case en: Ten;
}[fv];
```

- خصوصیات رمز :

چهـار عملگـر رمـز digital signing، stream cipher encryption، block cipher encryption، public key وبه ترتیب به digitally – signed، stream - ciphered، block - ciphered و public – key - encrypted تعیین میشود. یک مرحله رمز فیلـد قبـل از مشخصـات نوع فیلد بوسیله موکول کردن به معرفی لغت کلید مناسـب تعیـین میشـود. کلیـدهای رمـز توسـط وضعیت جلسه فعلی اجرا میشود

در علامتهای دیجیتالی یـک روش عملکـردی بـه عنـوان ورودی بـرای یـک الگـوریتم علامتـی استفاده میشود. در علامتگذاری RSA ساختار36 بایتی2 تاHASH (یکی SHA و دیگـریMDS) علامتگذاری می‌شود (رمز شده با کلید خصوصی). در DSS، 20 بایت HASH، SHAمستقیماً از طریق الگوریتم علامت دیجیتال بدون HASHاضافی اجرا می‌شود.

در رمز تراشه رشته متن معمولی exclusive - ored است با یک مقدار یکسان در خروجی تولیدی.

در رمز تراشه بلوک هربلوک متن معمولی به یک بلوک با متن رمزی تبدیل شود. زیرا بعید است که متن معمولی (هر چند که داده‌ها را بفرستند) در داخل مقدار بلوک ضروری (64 بیت) نفوذ کند، ضروری بنظر میرسد پایان بلوکهای کوتاه با برخی از الگوهای مجاز pad out شود معمولاً همه صفر. در رمز کلید معمولی یک عملکرد با رمز« trapdoors » برای رمز داده‌های صادر شونده استفاده میشود. رمز داده‌ها با کلید عمومی یک جفت کلید داده شده فقط میتواند با کلید خصوصی از حالت رمز درآورده شود و بالعکس .

- ثابت‌ها:

ثابتهای چاپی را میتوان برای مقاصد مشخصات یا اعلان یک نمونه از نوع مطلوب و گماردن مقادیری برای آن تعریف کرد. انواع مقرر (opaque ، بردارهایی با طول متغییر ساختارهایی که محتوی opaque است) را نمیتوان تقبل کرد . هیچ فیلدی از یک ساختار چند جزئی یا بردار ، ممکن نیست حذف شود.

```
} struct
; unit8    field1
; unit8    field2
; Example1 }
```

2-1-6 پروتکل SSL :

SSL یک پروتکل لایه‌ای است. در هرلایه پیامها ممکن است شامل فیلدهایی برای طول ، توضیحات ، و ظرفیت باشد . SSL پیامها را به منظور انتقال ، پراکندگی داده‌ها به داخل بلوکهای اداره کردنی می‌گیرد بطور اختیاری داده‌ها را فشرده میکند یک MAC را اجراء میکند رمز دهی میکند و نتیجه را انتقال میدهد . داده‌های دریافتی رمزگشایی می‌شوند، بررسی شده از حالت فشردگی درمی‌آیند ومجدداً جفت وجور می‌شوند سپس به مشتریهای سطح بالاتر تحویل داده می‌شوند .

- جلسه وحالتهای ارتباطی :

یک جلسه SSL، stateful است ، مسئول پروتکل SSL handshake است . بمنظور همکاری حالتهای مشتری و خدمتگزار ، بموجب آن اجازه میدهد وضعیت پروتکل ماشینهای هرکدام بطور پیوسته عمل کند .

علیرغم اینکه وضعیت دقیقاً مقایسه نمی‌شود . بطور منطقی وضعیت دو بار نمایش داده میشود یکی بعنوان وضعیت عملگر فعلی و(طی پروتکل hand.shake) مجدداً به عنوان وضعیت نامعلوم بعلاوه مراحل خواندن و نوشتن به طور جداگانه پشتیبانی میشوند . وقتی مشتری یک پیام chang cipher spec (مشخصات رمز را تغییر بدهید) را دریافت میکند وضعیت خواندن مطالب نامعلوم را در داخل وضعیت خواندن فعلی کپی میکند .وقتی مشتری یا خدمتگزار یک پیام chang cipher spec را ارسال میکند حالت نوشتن نامعلومی را در داخل حالت نوشتن فعلی کپی میکند وقتی بحث و مذاکره hand shake کامل است که مشتری و خدمتگزار پیامهای chang cipher spec را تغییر میدهند و سپس با استفاده از مشخصات cipher توافق جدیدی ابلاغ می‌کند . یک جلسه SSL ممکن است اطلاعات امن چندتایی را دربرگیرد . علاوه برآن قسمتهایی ممکن است دارای جلسات همزمان چندتایی باشد .

-لایه ثبت :

لایه ثبت SSL داده‌های غیر تفسیری را از لایه‌های بالاتر از بلوکهای پرگنجایش دلخواه دریافت میکند .

- پروتکل تغییر مشخصه رمز :

این پروتکل برای سیگنالهای انتقال در استراتژی رمزگذاری بوجود آمده است ، پروتکل شامل یک پیغام تکی است ، که تحت Ciper Spec جاری رمزگذاری وفشرده میشود پیغام شامل یک بایت با مقدار یک است .

پیغام تغییر مشخصه رمز هم به وسیله مشتری و هم به وسیله خدمتگزار فرستاده میشود برای اینکه به دسته رسیده اطلاع دهد که رکوردهای بعدی تحت انتقال درست ciper spec وکلیدها ، محافظت خواهند شد . دریافت این پیغام باعث نیشود دریافت کننده وضعیت (خواندن نامعلوم) را داخل وضعیت خواندن جاری کپی کند. مراحل خواندن و نوشتن به طور جداگانه توسط مشتری و خدمتگزار SSL نگهداری می‌شوند . وقتی مشتری یا خدمتگزار یک پیغام تغییر مشخصه رمزرا دریافت میکند وضعیت نامعلوم خواندن را داخل وضعیت جاری خواندن کپی میکند . وقتی مشتری یا خدمتگزار یک پیغام تغییر مشخصه رمز را مینویسد ، وضعیت نوشتن نامعلوم را داخل وضعیت نوشتن جاری کپی میکند .

مشتری یک پیغام تغییر مشخصه رمز را به دنبال پیغام‌های تغییرکلید Hand shake و تائید گواهینامه (اگر باشد) ارسال می‌کند وخدمتگزار میفرستد بعد ازپردازش موفقیت‌آمیز پیغام هشدار غیرمترقبه تولید می‌کند .

- پروتکل هشدار :

یکی از انواع ظرفیت که توسط لایه رکورد SSL پشتیبانی میشود نوع هشدار (Alert) است . پیغام Alert سختی پیغام و یک توضیح از هشدار را عبور میدهد پیغام‌های هشدار با یک سطح از نتایج مخرب درپایان فوری ارتباط در این مرحله دیگر مطابقتهای ارتباطات به جلسه ممکن است ادامه پیدا کند اما شناسه جلسه بایدجلسه مخرب را باطل کند . باز دارد از اینکه ارتباط جدیدی را برقرار کند مانند دیگر پیغام‌ها ،پیغام‌های هشدار رمزگذاری و فشرده میشوند ، همانطوری که بوسیله مرحله ارتباط جاری مشخص شده است .

- بررسی اجمالی پروتکل Handshake

پارامترهای رمزنگار مرحله جلسه توسط پروتکل SSl handshake تولید میشوند که بالای لایه رکورد SSL عمل میکند . وقتی یک مشتری و خدمتگزار در SSL برای بار نخست ارتباط را آغاز میکنند ، آنها روی یک ورژن ، انتخاب الگوریتم رمزگذار رسمیت دادن دلخواه به یکدیگر و استفاده از تکنیکهای رمزی کردن Public – Key برای تولید رمزهای مشترک تطابق حاصل میکنند . این پردازشها در پروتکل Handshake انجام میشوند که به صورت خلاصه شده در زیر آمده‌اند :

مشتری یک پیغام Client hello به خدمتگذار میفرستد و او با یک پیغام Server hello جواب خواهد داد یا اینکه یک خطای کشنده رخ خواهد داد و ارتباط اشتباه است . این دو پیغام برای برقراری امنیت با قابلیتهای بیشتر بین مشتری و خدمتگزار (خدمتگزار) استفاده میشوند . این دو پیغام خواص زیر را برقرار میکنند :

ورژن پرتکل شناسه جلسه مجموعه برنامه‌های رمز و متد فشرده‌سازی به علاوه دو مقدار تصادفی تولید و رد و بدل میشوند . Randdom . Client Hello و . Serner Hello . Randdom .

به دنبال پیغامهای سلام ، خدمتگزار گواهینامه خود را ارسال خواهد کرد اگر تصدیق شده باشد علاوه بر آن یک پیغام تغییر کلید خدمتگزار ممکن است ارسال شود ، درصورت نیاز اگر خدمتگزار تصدیق شد اگر که این مجموعه برنامه رمز منتخب مقتضی باشد یک گواهینامه از مشتری درخواست کند .

اکنون خدمتگزار یک پیغام Server hello . done را ارسال خواهد کرد . مشتمل بر اینکه فاز Hello - message در Handshake کامل شده است . سپس خدمتگزار منتظر دریافت جوابی ازمشتری خواهد بود .

اگر Server یک پیغام درخواست گواهینامه ارسال کرده باشد ، مشتری باید پیغـام گواهینامـه یا هشدار عدم وجود گواهینامه را ارسال کنـد . پیغـام تغییـر کلیـد مشـتری Client key change اکنون ارسال میشود و اندازه این پیغام به الگوریتم کلید عمومی انتخاب شـده بـین Client Hello و Server Hello بستیگ دارد . اگر مشتری یک گواهینامه بـا قابلیـت علامتگـذاری ارسـال کـرده باشد یک پیغام تائید گواهینامه برای تصدیق صریح گواهینامه ارسال میشود . در این مرحلـه یـک پیغام « تغییر مشخصه رمز » به وسیله مشتری ارسال میشود و مشتری مشخصـه رمـز زائـد را داخل مشخصه رمز جاری کپی میکند ، سپس مشـتری فـوراً پیغـام خاتمـه را تحـت الگوریتمهـا ، کلیدها و رمزهای جدید ارسال میکند ، در پاسخ خدمتگزار پیغام « تغییر مشخصه رمـز » خـود را ارسال خواهد کرد زوائد را به مشخصه رمز جـاری انتقـال میدهـد و پیغـام خاتمـه خـودرا تحـت مشخصه رمزجدید ارسال میکند . اکنون Handshake ماکل است و مشتری و خدمتگزار میتوانند داده لایه Application را مبادله کنند .

وقتی مشتری وخدمتگزار تصمیم میگیرند که یک جلسه قبل را دوباره از سر بگیرند یـا جلسـه موجود را دو برابر کنند نمودار پیغام مانند زیر است (به جای مبادله پارماترهای امنیتی جدید) :

مشتری یک پیغام Cient hello با استفاده از شناسه جلسه جلسه‌ایـی کـه از سـرگرفته شـده ارسال میکند . خدمتگزار سپس جلسه پنهان خود را برای تطبیق برسی میکند . اگر تطابق بر قـرار شد و خدمتگزار بخواهد تحت وضعیت جلسه مشخص شده ارتبـاط را دوبـاره برقـرار کنـد ، یـک server hello با همان مقدار شناسه جلسه ارسال می‌کند . اکنون هـم مشـتری و هـم خـدمتگزار باید پیغام‌های « تغییرمشخصه رمز» را ارسال کنند و مستقیماً برای پیغام های خاتمـه اقـدام کننـد به محض اینکه ارتباط مجدد کامل شد مشتری و خدمتگزار بـه مبادلـه داده در لایـه application اقدام میکنند . اگر تطابق شناسه جلسه پیدا نشد ، خدمتگزار یک شناسه جلسه جدیـد تولیـد میکنـد ومشتری وخدمتگزار SSL یک hand shake کامل را نمایش میدهند .

- پروتکل کاربرد داده

پیغام‌های کاربرد داده بوسیلهٔ لایهٔ رکورد حمل می‌شـوند و قسـمت‌بنـدی شـده، فشـرده و بـر مبنای مرحلهٔ ارتباط جاری رمزگذاری شده هستند. با پیغام‌ها ماننـد دادهٔ شـفاف بـه لایـهٔ رکـورد رفتار می‌شود.

7-1-2 محاسبات رمزگذاری

تغییر کلید، تصدیق، رمزگذاری و الگوریتم‌های MAC با chipher.suite انتخاب شده بوسیلهٔ خدمتگزار خاتمه می‌یابند و در پیغام server hello آشکار می‌شوند.

2-1-7-1 محاسبات رمزگذاری نامتقارن

الگوریتم‌های نامتقارن در پروتکل handshake برای تصدیق دسته‌ها و تولید کلیدهای تقسیم شده و سرّی استفاده شده‌اند.

برای Diffie-Hellman، RSA و Fortezza، همان الگوریتم برای تبدیل pre_maste_secret به master_secret استفاده می‌شود. pre_master_secret باید از حافظه پاک بشود تا master_secret محاسبه شده باشد.

-RSA

وقتی RSA برای تصدیق خدمتگزار و تغییر کلید استفاده می‌شود، یک pre_master_secret 48 بایتی به وسیلهٔ مشتری تولید می‌شود، تحت کلید عمومی خدمتگزار رمزگذاری می‌شود و به خدمتگزار ارسال می‌شود. هر دو دسته سپس pre_master_secret را درون master_secret کپی می‌کنند، همانطور که در بالا مشخص شده است.

امضاهای دیجیتالی RSA با استفاده از بلوک نوع یک PKCS1] 1 # PKCS] اجرا می‌شوند. رمز کلید عمومی RSA با استفاده از بلوک نوع دو PKCS # 1 اجرا می‌شود.

- Diffie-Hellman

یک محاسبهٔ Diffie-Hellman قراردادی اجرا می‌شود. کلید انتقال (Z) مانند pre_master_secret بکار می‌رود، و درون master_secret تبدیل می‌شود، همانطور که در بالا مشخص شده است.

توجه: پارامترهای Diffie-Hellman بوسیلهٔ خدمتگزار مشخص شده‌اند و ممکن است هر کدام بی‌دوام باشند یا همراه گواهینامه خدمتگزار شامل شوند.

- Fortezza

یک 48 pre_master_secret بایتی فرستاده شدهٔ رمزگذاری شده تحت TEK و IV آن. خدمتگزار pre_master_secret را از رمز خارج می‌کند و آن را درون master_secret تبدیل می‌کند، همانطور که در بالا مشخص شده است. کلیدهای رمز حجیم و IVها برای رمزگذاری توسط Token مشتری تولید می‌شوند و در پیغام تغییر کلید تغییر می‌کنند؛ master_secret فقط برای محاسبات MAC استفاده می‌شود.

2-1-7-2 محاسبات رمزگذاری متقارن و مشخصهٔ رمز

تکنیک استفاده از رمز و تائید درستی رکوردهای SSL بوسیلهٔ مشخصهٔ رمز فعال جاری مشخص می‌شود. یک مثال واقعی به رمز درآوردن داده با استفاده از DES و تولید کدهای تصدیق شده با استفاده از MD5 خواهد بود. رمزگذاری و الگوریتم‌های MAC که به SSL_NULL_WITH_NULL_NULL در ابتدای پروتکل SSL handshake تنظیم شده‌اند، دلالت دارد بر اینکه هیچ پیغام تصدیق یا رمزی نمایش داده نشده است. پروتکل handshake برای انتقال یک تغییر مشخصهٔ سرُی یا بیشتر و برای تولید کلیدهای رمزگذاری استفاده می‌شود.

- رمز اصلی

قبل از رمزگذاری سرُی یا اینکه درستی تصدیق بتواند روی رکوردها اجرا شود، مشتری و خدمتگزار نیاز دارند که اطلاعات سرُی تقسیم شده را تولید کنند که فقط برای خودشان شناخته شده است. این مقدار یک کمیت 48 بایتی است که رمز اصلی نامیده می‌شود. رمز اصلی برای تولید کلیدها و رمزها برای رمزگذاری و محاسبات MAC بکار می‌رود. بعضی از الگوریتم‌ها، مانند Fortezza، ممکن است زیررویهٔ خودشان را برای تولید کلیدهای رمز داشته باشند.

- تبدیل رمز اصلی به کلیدها و رمزهای MAC

رمز اصلی دورن یک سلسله از بایتهای رمزی خرد می‌شود، که به رمزهای MAC، کلیدها و IVهای غیر خروجی مورد نیاز بوسیلهٔ مشخصهٔ رمز جاری، منتسب می‌شوند.

مشخصهٔ رمز یک رمز MAC نوشته شده توسط مشتری، یک رمز MAC نوشته شده توسط خدمتگزار، یک کلید نوشته شده توسط مشتری، یک کلید نوشته شده توسط خدمتگذار، یک IV نوشتن مشتری و یک IV نوشتن خدمتگذار، نیاز دارد که برای رمز اصلی برای آن منظور تولید شده‌اند. مقادیر غیرقابل استفاده مانند کلیدهای Fortezza ابلاغ شده در پیغام تغییر کلید، خالی هستند. ورودی‌های زیر برای پردازش تعریف کلید موجود هستند.

وقتی کلیدها و رمزهای MAC تولید می‌شوند، رمز اصلی مانند یک منبع Entropy بکار می‌رود، و مقادیر تصادفی مواد نگهدارندهٔ غیررمزی و IVها را برای رمزهای قابل خروج تهیه می‌کنند.

برای تولید مواد کلید، محاسبه کنید

تا زمانی که خروجی‌های کافی تولید شده باشند. سپس key_block به صورت زیر قسمت بندی می‌شود.

هر مادهٔ key_block اضافی دور انداخته می‌شود.

الگوریتم‌های رمزگذاری قابل صادر کردن (برای هر کدام از CipherSpec.is_exportable درست است) پردازشهای اضافی نیاز دارد مانند زیر برای اشتقاق کلیدهای نوشتنی پایانی:

الگوریتم‌های رمزگذاری قابل صادر کردن IVهایشان را برای پیغام‌های تصادف مشتق می‌کنند.

خروجیهای MD5 برای اختصاص اندازه بوسیلهٔ دور ریختن کمترین بایتهای مهم، مرتب می‌شوند.

نتیجه گیری :

برای حصول اطمینان از کارایی و قابلیت اعتماد سیستم مبادله الکترونیکی داده ها در یک سازمان باید خطرات احتمالی مورد تجزیه و تحلیل قرار گیرد ، منشأ بالقوه تهدیدها شناسایی شود و در صورت بروز واقعی خطر ، هزینه هایی که ایجاد خواهد شد مشخص گردد . سپس برای مقابله با این خطرات تدابیر لازم اندیشیده شود . برای ایمن سازی مبادلات اطلاعات بین سازمانهایی که به شبکه ارتباطی متصل هستند ، امروزه فن آوری لازم موجود و در دسترس است اما سطح امنیتی که باید فراهم شود با محدودیتهایی روبرو است زیرا هزینه اجرای تدابیر امنیتی نباید از ارزش اطلاعاتی که ایمن سازی آن موردنظر است ،تجاوز کند .

مراجع:

1) پایان نامه تجارت الکترونیکی – دانشگاه فنی مهندسی تهران جنوب – شماره CT/1102

2 Secure Electronic Transactions Protocol - http://www.set.ch

3)Security of eCommerce Systems - A Netcraft White Paper - May 2001

4) SSL 3.0 Specification - http://www.smartcard.com/technical

امنیت در تجارت الکترونیکی

Security of e_ commerce

عنوان درس :شبکه های کامپیوتری

نام استاد :جناب آقای مهندس فیروزبخت

نویسنده :اشرف جباری

کد درس : د

1. Status of this memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.
Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or made obsolete by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as work in progress.
To learn the current status of any Internet-Draft, please check the 1id-abstracts.txt listing contained in the Internet Drafts Shadow Directories on ds.internic.net (US East Coast), nic.nordu.net (Europe), ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific Rim).

---

2. Abstract

This document specifies Version 3.0 of the Secure Sockets Layer (SSL V3.0) protocol, a security protocol that provides communications privacy over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

---

3. Introduction

The primary goal of the SSL Protocol is to provide privacy and reliability between two communicating applications. The protocol is composed of two layers. At the lowest level, layered on top of some reliable transport protocol (e.g., TCP[TCP]), is the SSL Record Protocol. The SSL Record Protocol is used for encapsulation of various higher level protocols. One such encapsulated protocol, the SSL Handshake Protocol, allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. One advantage of SSL is that it is application protocol independent. A higher level protocol can layer on top of the SSL Protocol transparently.
The SSL protocol provides connection security that has three basic properties:

- The connection is private. Encryption is used after an initial handshake to define a secret key. Symmetric cryptography is used for data encryption (e.g., DES[DES], RC4[RC4], etc.)
- The peer's identity can be authenticated using asymmetric, or public key, cryptography (e.g., RSA[RSA], DSS[DSS], etc.).
- The connection is reliable. Message transport includes a message integrity check using a keyed MAC. Secure hash functions (e.g., SHA, MD5, etc.) are used for MAC computations.

---

4. Goals

The goals of SSL Protocol v3.0, in order of their priority, are:

1. Cryptographic security

   SSL should be used to establish a secure connection between two parties.

2. Interoperability

   Independent programmers should be able to develop applications utilizing SSL 3.0 that will then be able to successfully exchange cryptographic parameters without knowledge of one another's code.

Note:It is not the case that all instances of SSL (even in the same application domain) will be able to successfully connect. For instance, if the server supports a particular hardware token, and the client does not have access to such a token, then the connection will not succeed.

3. Extensibility

   SSL seeks to provide a framework into which new public key and bulk encryption methods can be incorporated as necessary. This will also accomplish two sub-goals: to prevent the need to create a new protocol (and risking the introduction of possible new weaknesses) and to avoid the need to implement an entire new security library.

4. Relative efficiency

   Cryptographic operations tend to be highly CPU intensive, particularly public key operations. For this reason, the SSL protocol has incorporated an optional session caching scheme to reduce the number of connections that need to be established from scratch. Additionally, care has been taken to reduce network activity.

---

5. Goals of this document

The SSL Protocol Version 3.0 Specification is intended primarily for readers who will be implementing the protocol and those doing cryptographic analysis of it. The spec has been written with this in mind, and it is intended to reflect the needs of those two groups. For that reason, many of the algorithm-dependent data structures and rules are included in the body of the text (as opposed to in an Appendix), providing easier access to them.
This document is not intended to supply any details of service definition nor interface definition, although it does cover select areas of policy as they are required for the maintenance of solid security.

---

6. Presentation language

This document deals with the formatting of data in an external representation. The following very basic and somewhat casually defined presentation syntax will be used. The syntax draws from several sources in its structure. Although it resembles the programming language 'C' in its syntax and XDR [XDR] in both its syntax and intent, it would be risky to draw too many parallels. The purpose of this presentation language is to document SSL only, not to have general application beyond that particular goal.

6.1 Basic block size

The representation of all data items is explicitly specified. The basic data block size is one byte (i.e. 8 bits). Multiple byte data items are concatenations of bytes, from left to right, from top to bottom. From the bytestream a multi-byte item (a numeric in the example) is formed (using C notation) by:

```
value = (byte[0] << 8*(n-1)) | (byte[1] << 8*(n-2)) | ... | byte[n-1];
```

This byte ordering for multi-byte values is the commonplace network byte order or big endian format.

6.2 Miscellaneous

Comments begin with "/*" and end with "*/".

Optional components are denoted by enclosing them in italic "[ ]" brackets.

Single byte entities containing uninterpreted data are of type opaque.

6.3 Vectors

A vector (single dimensioned array) is a stream of homogeneous data elements. The size of the vector may be specified at documentation time or left unspecified until runtime. In either case the length declares the number of bytes, not the number of elements, in the vector.

The syntax for specifying a new type T' that is a fixed length vector of type T is

```
T T'[n];
```

Here T' occupies n bytes in the data stream, where n is a multiple of the size of T. The length of the vector is not included in the encoded stream.

In the following example, Datum is defined to be three consecutive bytes that the protocol does not interpret, while Data is three consecutive Datum, consuming a total of nine bytes.

```
opaque Datum[3]; /* three uninterpreted bytes of data */
Datum Data[9];   /* 3 consecutive 3 byte vectors */
```

Variable length vectors are defined by specifying a subrange of legal lengths, inclusively, using the notation <floor..ceiling>. When encoded, the actual length precedes the vector's contents in the byte stream. The length will be in the form of a number consuming as many bytes as required to hold the vector's specified maximum (ceiling) length. A variable length vector with an actual length field of zero is referred to as an empty vector.

```
T T'<floor..ceiling>;
```

In the following example, mandatory is a vector that must contain between 300 and 400 bytes of type opaque. It can never be empty. The actual length field consumes two bytes, a uint16, sufficient to represent the value 400 (see Section 6.4). On the other hand, longer can represent up to 800 bytes of data, or 400 uint16 elements, and it may be empty. Its encoding will include a two byte actual length field prepended to the vector.

```
opaque mandatory<300..400>; /* length field is 2 bytes, cannot be empty */
uint16 longer<0..800>;      /* zero to 400 16-bit unsigned integers */
```

6.4 Numbers

The basic numeric data type is an unsigned byte (uint8). All larger numeric data types are formed from fixed length series of bytes concatenated as described in Section 6.1 and are also unsigned. The following numeric types are predefined.

```
uint8 uint16[2];
```

```
uint8 uint24[3];
uint8 uint32[4];
uint8 uint64[8];
```

6.5 Enumerateds

An additional sparse data type is available called enum. A field of type enum can only assume the values declared in the definition. Each definition is a different type. Only enumerateds of the same type may be assigned or compared. Every element of an enumerated must be assigned a value, as demonstrated in the following example. Since the elements of the enumerated are not ordered, they can be assigned any unique value, in any order.

enum { $e_1(v_1)$, e_2 (v_1), ... , e_n (v_N), [(n)] } T_e;

Enumerateds occupy as much space in the byte stream as would its maximal defined ordinal value. The following definition would cause one byte to be used to carry fields of type Color.

```
enum { red(3), blue(5), white(7) } Color;
```

One may optionally specify a value without its associated tag to force the width definition without defining a superfluous element. In the following example, Taste will consume two bytes in the data stream but can only assume the values 1, 2 or 4.

```
enum { sweet(1), sour(2), bitter(4), (32000) } Taste;
```

The names of the elements of an enumeration are scoped within the defined type. In the first example, a fully qualified reference to the second element of the enumeration would be Color.blue. Such qualification is not required if the target of the assignment is well specified.

```
Color color = Color.blue;       /* overspecified, but legal */
Color color = blue;     /* correct, type is implicit */
```

For enumerateds that are never converted to external representation, the numerical information may be omitted.

```
enum { low, medium, high } Amount;
```

6.6 Constructed types

Structure types may be constructed from primitive types for convenience. Each specification declares a new, unique type. The syntax for definition is much like that of C.

struct {
    T_1 f_1;
    T_2 f_2;
    ...
    T_n f_n;
} [T];

The fields within a structure may be qualified using the type's name using a syntax much like that available for enumerateds. For example, $T.f_2$ refers to the second field of the previous declaration. Structure definitions may be embedded.

6.6.1 Variants

Defined structures may have variants based on some knowledge that is available within the environment. The selector must be an enumerated type that defines the possible variants the structure defines. There must be a case arm for every element of the enumeration declared in the select. The body of the variant structure may be given a label for reference. The mechanism by which the variant is selected at runtime is not prescribed by the presentation language.

```
struct {
      T1 f1;
      T2 f2;
         ....
      Tn fn;
      select (E) {
         case e1: Te1;
         case e2: Te2;
            ....
         case en: Ten;
      } [fv];
} [Tv];
```

For example

```
enum { apple, orange } VariantTag;
struct {
      uint16 number;
      opaque string<0..10<;     /* variable length */
} V1;
struct {
      uint32 number;
      opaque string[10];      /* fixed length */
} V2;
struct {
      select (VariantTag) {   /* value of variant selector is implicit */
         case apple: V1;     /* definition of VariantBody, tag = apple */
         case orange: V2;    /* definition of VariantBody, tag = orange */
      } variant_body; /* optional label on the variant portion */
} VariantRecord;
```

Variant structures may be qualified (narrowed) by specifying a value for the selector prior to the type. For example, a

```
orange VariantRecord
```

is a narrowed type of a VariantRecord containing a variant_body of type V2.

6.7 Cryptographic attributes

The four cryptographic operations digital signing, stream cipher encryption, block cipher encryption, and public key encryption are designated digitally-signed, stream-ciphered, block-ciphered, and public-key-encrypted, respectively. A field's cryptographic processing is specified by prepending an appropriate key word designation before the field's type specification. Cryptographic keys are implied by the current session state (see Section 7.1).

In digital signing, one-way hash functions are used as input for a signing algorithm. In RSA signing, a 36-byte structure of two hashes (one SHA and one MD5) is signed (encrypted with the private key). In DSS, the 20 bytes of the SHA hash are run directly through the Digital Signing Algorithm with no additional hashing.

In stream cipher encryption, the plaintext is exclusive-ORed with an identical amount of output generated from a cryptographically-secure keyed pseudorandom number generator.

In block cipher encryption, every block of plaintext encrypts to a block of ciphertext. Because it is unlikely that the plaintext (whatever data is to be sent) will break neatly into the necessary block size (usually 64 bits), it is necessary to pad out the end of short blocks with some regular pattern, usually all zeroes.

In public key encryption, one-way functions with secret "trapdoors" are used to encrypt the outgoing data. Data encrypted with the public key of a given key pair can only be decrypted with the private key, and vice-versa.

In the following example:

```
stream-ciphered struct {
      uint8 field1;
      uint8 field2;
      digitally-signed opaque hash[20];
} UserType;
```

The contents of hash are used as input for a signing algorithm, then the entire structure is encrypted with a stream cipher.

6.8 Constants

Typed constants can be defined for purposes of specification by declaring a symbol of the desired type and assigning values to it. Under-specified types (opaque, variable length vectors, and structures that contain opaque) cannot be assigned values. No fields of a multi-element structure or vector may be elided.

For example,

```
struct {
      uint8 f1;
      uint8 f2;
} Example1;
Example1 ex1 = {1, 4};  /* assigns f1 = 1, f2 = 4 */
```

---

7. SSL protocol

SSL is a layered protocol. At each layer, messages may include fields for length, description, and content. SSL takes messages to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, and transmits the result. Received data is decrypted, verified, decompressed, and reassembled, then delivered to higher level clients.

7.1 Session and connection states

An SSL session is stateful. It is the responsibility of the SSL Handshake protocol to coordinate the states of the client and server, thereby allowing the protocol state machines of each to operate consistently, despite the fact that the state is not exactly parallel. Logically the state is represented twice, once as the current operating state, and (during the handshake protocol) again as the pending state. Additionally, separate read and write states are maintained. When the client or server receives a change cipher spec message, it copies the pending read state into the current read state. When the client or server sends a change cipher spec message, it copies the pending write state into the current write state. When the handshake negotiation is complete, the client and server exchange change cipher spec messages (see Section 7.3), and then communicate using the newly agreed-upon cipher spec.

An SSL session may include multiple secure connections; in addition, parties may have multiple simultaneous sessions.

The session state includes the following elements:

session identifier

An arbitrary byte sequence chosen by the server to identify an active or resumable session state

peer certificate

X509.v3[X509] certificate of the peer. This element of the state may be null.

compression method

The algorithm used to compress data prior to encryption.

cipher spec

Specifies the bulk data encryption algorithm (such as null, DES, etc.) and a MAC algorithm (such as MD5 or SHA). It also defines cryptographic attributes such as the hash_size. (See Appendix A.7 for formal definition.)

master secret

48-byte secret shared between the client and server.

is resumable

A flag indicating whether the session can be used to initiate new connections.

The connection state includes the following elements:

server and client random

Byte sequences that are chosen by the server and client for each connection.

server write MAC secret

The secret used in MAC operations on data written by the server.

client write MAC secret

The secret used in MAC operations on data written by the client.

server write key

The bulk cipher key for data encrypted by the server and decrypted by the client.

client write key

The bulk cipher key for data encrypted by the client and decrypted by the server.

initialization vectors

When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the SSL handshake protocol. Thereafter the final ciphertext block from each record is preserved for use with the following record.

sequence numbers

Each party maintains separate sequence numbers for transmitted and received messages for each connection. When a party sends or receives a change cipher spec message, the appropriate sequence number is set to zero. Sequence numbers are of type uint64 and may not exceed $2^{64}-1$.

7.2 Record layer

The SSL Record Layer receives uninterpreted data from higher layers in non-empty blocks of arbitrary size.

7.2.1Fragmentation

The record layer fragments information blocks into SSLPlaintext records of 2^{14} bytes or less. Client message boundaries are not preserved in the record layer (i.e., multiple client messages of the same ContentType may be coalesced into a single SSLPlaintext record).

```
struct {
      uint8 major, minor;
} ProtocolVersion;
enum {
      change_cipher_spec(20), alert(21), handshake(22),
      application_data(23), (255)
} ContentType;
struct {
      ContentType type;
      ProtocolVersion version;
      uint16 length;
      opaque fragment[SSLPlaintext.length];
} SSLPlaintext;
```

type

The higher level protocol used to process the enclosed fragment.

version

The version of protocol being employed. This document describes SSL Version 3.0 (See Appendix A.1.1).

length

The length (in bytes) of the following SSLPlaintext.fragment. The length should not exceed 2^{14}.

fragment

The application data. This data is transparent and treated as an independent block to be dealt with by the higher level protocol specified by the type field.

Note: Data of different SSL Record layer content types may be interleaved. Application data is generally of lower precedence for transmission than other content types.

7.2.2 Record compression and decompression

All records are compressed using the compression algorithm defined in the current session state. There is always an active compression algorithm, however initially it is defined as CompressionMethod.null. The compression algorithm translates an SSLPlaintext structure into an SSLCompressed structure. Compression functions erase their state information whenever the CipherSpec is replaced.

Note: The CipherSpec is part of the session state described in Section 7.1. References to fields of the CipherSpec are made throughout this document using presentation syntax. A more complete description of the CipherSpec is shown in Appendix A.7.

Compression must be lossless and may not increase the content length by more than 1024 bytes. If the decompression function encounters an SSLCompressed.fragment that

would decompress to a length in excess of 2^{14} bytes, it should issue a fatal decompression_failure alert (Section 7.4.2).

```
struct {
    ContentType type;           /* same as SSLPlaintext.type */
    ProtocolVersion version;    /* same as SSLPlaintext.version */
    uint16 length;
    opaque fragment[SSLCompressed.length];
} SSLCompressed;
```

length

The length (in bytes) of the following SSLCompressed.fragment. The length should not exceed $2^{14} + 1024$.

fragment

The compressed form of SSLPlaintext.fragment.

Note: A CompressionMethod.null operation is an identity operation; no fields are altered. (See Appendix A.4.1)

Implementation note: Decompression functions are responsible for ensuring that messages cannot cause internal buffer overflows.

7.2.3 Record payload protection and the CipherSpec

All records are protected using the encryption and MAC algorithms defined in the current CipherSpec. There is always an active CipherSpec, however initially it is SSL_NULL_WITH_NULL_NULL, which does not provide any security.

Once the handshake is complete, the two parties have shared secrets which are used to encrypt records and compute keyed message authentication codes (MACs) on their contents. The techniques used to perform the encryption and MAC operations are defined by the CipherSpec and constrained by CipherSpec.cipher_type. The encryption and MAC functions translate an SSLCompressed structure into an SSLCiphertext. The decryption functions reverse the process. Transmissions also include a sequence number so that missing, altered, or extra messages are detectable.

```
struct {
    ContentType type;
    ProtocolVersion version;
    uint16 length;
    select (CipherSpec.cipher_type) {
       case stream: GenericStreamCipher;
       case block: GenericBlockCipher;
    } fragment;
} SSLCiphertext;
```

type

The type field is identical to SSLCompressed.type.

version

The version field is identical to SSLCompressed.version.

length

The length (in bytes) of the following SSLCiphertext.fragment. The length may not exceed $2^{14} + 2048$.

fragment

The encrypted form of SSLCompressed.fragment, including the MAC.

7.2.3.1 Null or standard stream cipher

Stream ciphers (including BulkCipherAlgorithm.null - See Appendix A.7) convert SSLCompressed.fragment structures to and from stream SSLCiphertext.fragment structures.

```
stream-ciphered struct {
      opaque content[SSLCompressed.length];
      opaque MAC[CipherSpec.hash_size];
} GenericStreamCipher;
```

The MAC is generated as:

```
hash(MAC_write_secret + pad_2 +
    hash (MAC_write_secret + pad_1 + seq_num + length + content));
```

where "+" denotes concatenation.

pad_1

The character 0x36 repeated 48 time for MD5 or 40 times for SHA.

pad_2

The character 0x5c repeated the same number of times.

seq_num

The sequence number for this message.

hash

The hashing algorithm derived from the cipher suite.

Note that the MAC is computed before encryption. The stream cipher encrypts the entire block, including the MAC. For stream ciphers that do not use a synchronization vector (such as RC4), the stream cipher state from the end of one record is simply used on the subsequent packet. If the CipherSuite is SSL_NULL_WITH_NULL_NULL, encryption consists of the identity operation (i.e., the data is not encrypted and the MAC size is zero implying that no MAC is used). SSLCiphertext.length is SSLCompressed.length plus CipherSpec.hash_size.

7.2.3.2 CBC block cipher

For block ciphers (such as RC2 or DES), the encryption and MAC functions convert SSLCompressed.fragment structures to and from block SSLCiphertext.fragment structures.

```
block-ciphered struct {
    opaque content[SSLCompressed.length];
    opaque MAC[CipherSpec.hash_size];
    uint8 padding[GenericBlockCipher.padding_length];
    uint8 padding_length;
} GenericBlockCipher;
```

The MAC is generated as described in Section 7.2.3.1.

padding

Padding that is added to force the length of the plaintext to be a multiple of the block cipher's block length.

padding_length

The length of the padding must be less than the cipher's block length and may be zero. The padding length should be such that the total size of the GenericBlockCipher structure is a multiple of the cipher's block length.

The encrypted data length (SSLCiphertext.length) is one more than the sum of SSLCompressed.length, CipherSpec.hash_size, and padding_length.
Note: With CBC block chaining the initialization vector (IV) for the first record is provided by the handshake protocol. The IV for subsequent records is the last ciphertext block from the previous record.

7.3 Change cipher spec protocol

The change cipher spec protocol exists to signal transitions in ciphering strategies. The protocol consists of a single message, which is encrypted and compressed under the current (not the pending) CipherSpec. The message consists of a single byte of value 1.

```
struct {
      enum { change_cipher_spec(1), (255) } type;
} ChangeCipherSpec;
```

The change cipher spec message is sent by both the client and server to notify the receiving party that subsequent records will be protected under the just-negotiated CipherSpec and keys. Reception of this message causes the receiver to copy the read pending state into the read current state. Separate read and write states are maintained by both the SSL client and server. When the client or server receives a change cipher spec message, it copies the pending read state into the current read state. When the client or server writes a change cipher spec message, it copies the pending write state into the current write state. The client sends a change cipher spec message following handshake key exchange and certificate verify messages (if any), and the server sends one after successfully processing the key exchange message it received from the client. An unexpected change cipher spec message should generate an unexpected_message alert (Section 7.4.2). When resuming a previous session, the change cipher spec message is sent after the hello messages.

7.4 Alert protocol

One of the content types supported by the SSL Record layer is the alert type. Alert messages convey the severity of the message and a description of the alert. Alert messages with a level of fatal result in the immediate termination of the connection. In this case, other connections corresponding to the session may continue, but the session identifier must be invalidated, preventing the failed session from being used to establish new connections. Like other messages, Alert messages are encrypted and compressed, as specified by the current connection state.

```
enum { warning(1), fatal(2), (255) } AlertLevel;

enum {
      close_notify(0),
      unexpected_message(10),
      bad_record_mac(20),
      decompression_failure(30),
      handshake_failure(40), no_certificate(41), bad_certificate(42),
         unsupported_certificate(43), certificate_revoked(44),
         certificate_expired(45), certificate_unknown(46),
         illegal_parameter (47)
      (255)
} AlertDescription;

struct {
      AlertLevel level;
      AlertDescription description;
} Alert;
```

7.4.1 Closure alerts

The client and the server must share knowledge that the connection is ending in order to avoid a truncation attack. Either party may initiate the exchange of closing messages.

close_notify

This message notifies the recipient that the sender will not send any more messages on this connection. The session becomes unresumable if any connection is terminated without proper close_notify messages with level equal to warning.

7.4.2 Error alerts

Error handling in the SSL Handshake protocol is very simple. When an error is detected, the detecting party sends a message to the other party. Upon transmission or receipt of an fatal alert message, both

parties immediately close the connection. Servers and clients are required to forget any session-identifiers, keys, and secrets associated with a failed connection. The following error alerts are defined:

unexpected_message

An inappropriate message was received. This alert is always fatal and should never be observed in communication between proper implementations.

bad_record_mac

This alert is returned if a record is received with an incorrect MAC. This message is always fatal.

decompression_failure

The decompression function received improper input (e.g. data that would expand to excessive length). This message is always fatal.

handshake_failure

Reception of a handshake_failure alert message indicates that the sender was unable to negotiate an acceptable set of security parameters given the options available. This is a fatal error.

no_certificate

A no_certificate alert message may be sent in response to a certification request if no appropriate certificate is available.

bad_certificate

A certificate was corrupt, contained signatures that did not verify correctly, etc.

unsupported_certificate

A certificate was of an unsupported type.

certificate_revoked

A certificate was revoked by its signer.

certificate_expired

A certificate has expired or is not currently valid.

certificate_unknown

Some other (unspecified) issue arose in processing the certificate, rendering it unacceptable.

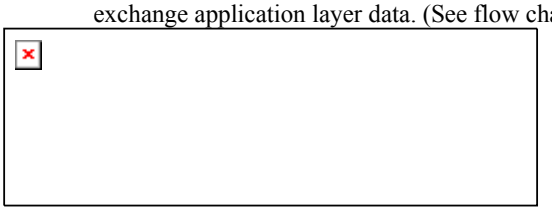
illegal_parameter

A field in the handshake was out of range or inconsistent with other fields. This is always fatal.

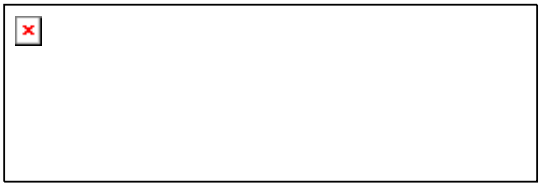
7.5 Handshake protocol overview

The cryptographic parameters of the session state are produced by the SSL Handshake Protocol, which operates on top of the SSL Record Layer. When a SSL client and server first start communicating, they agree on a protocol version, select cryptographic algorithms, optionally authenticate each other, and use public-key encryption techniques to generate shared secrets. These processes are performed in the handshake protocol, which can be summarized as follows:
The client sends a client hello message to which the server must respond with a server hello message, or else a fatal error will occur and the connection will fail. The client hello and server hello are used to establish security enhancement capabilities between client and server. The client hello and server hello establish the following attributes: protocol version, session ID, cipher suite, and compression method. Additionally, two random values are generated and exchanged: ClientHello.random and ServerHello.random.
Following the hello messages, the server will send its certificate, if it is to be authenticated. Additionally, a server key exchange message may be sent, if it is required (e.g. if their server has no certificate, or if its certificate is for signing only). If the server is authenticated, it may request a certificate from the client, if that is appropriate to the cipher suite selected.

Now the server will send the server hello done message, indicating that the hello-message phase of the handshake is complete. The server will then wait for a client response.
If the server has sent a certificate request message, the client must send either the certificate message or a no certificate alert. The client key exchange message is now sent, and the content of that message will depend on the public key algorithm selected between the client hello and the server hello. If the client has sent a certificate with signing ability, a digitally-signed certificate verify message is sent to explicitly verify the certificate.
At this point, a change cipher spec message is sent by the client, and the client copies the pending Cipher Spec into the current Cipher Spec. The client then immediately sends the finished message under the new algorithms, keys, and secrets. In response, the server will send its own change cipher spec message, transfer the pending to the current Cipher Spec, and send its Finished message under the new Cipher Spec. At this point, the handshake is complete and the client and server may begin to exchange application layer data. (See flow chart below.)



Note: To help avoid pipeline stalls, ChangeCipherSpec is an independent SSL Protocol content type, and is not actually an SSL handshake message.
When the client and server decide to resume a previous session or duplicate an existing session (instead of negotiating new security parameters) the message flow is as follows:
The client sends a client hello using the Session ID of the session to be resumed. The Server then checks its session cache for a match. If a match is found, and the server is willing to re-establish the connection under the specified session state, it will send a server hello with the same Session ID value. At this point, both client and server must send change cipher spec messages and proceed directly to finished messages. Once the re-establishment is complete, the client and server may begin to exchange application layer data. (See flow chart below.) If a Session ID match is not found, the server generates a new session ID and the SSL client and server perform a full handshake.



The contents and significance of each message will be presented in detail in the following sections.

7.6 Handshake protocol

The SSL Handshake Protocol is one of the defined higher level clients of the SSL Record Protocol. This protocol is used to negotiate the secure attributes of a session. Handshake messages are supplied to the SSL Record Layer, where they are encapsulated within one or more SSLPlaintext structures, which are processed and transmitted as specified by the current active session state.

```
enum {
      hello_request(0), client_hello(1), server_hello(2),
      certificate(11), server_key_exchange (12), certificate_request(13),
      server_hello_done(14), certificate_verify(15), client_key_exchange(16),
      finished(20), (255)
} HandshakeType;

struct {
      HandshakeType msg_type;        /* type of handshake message */
      uint24 length;  /* # bytes in handshake message body */
```

```
        select (HandshakeType) {
            case hello_request: HelloRequest;
            case client_hello: ClientHello;
            case server_hello: ServerHello;
            case certificate: Certificate;
            case server_key_exchange: ServerKeyExchange;
            case certificate_request: CertificateRequest;
            case server_hello_done: ServerHelloDone;
            case certificate_verify: CertificateVerify;
            case client_key_exchange: ClientKeyExchange;
            case finished: Finished;
        } body;
} Handshake;
```

The handshake protocol messages are presented in the order they must be sent; sending handshake messages in an unexpected order results in a fatal error.

7.6.1 Hello messages

The hello phase messages are used to exchange security enhancement capabilities between the client and server. When a new session begins, the CipherSpec encryption, hash, and compression algorithms are initialized to null. The current CipherSpec is used for renegotiation messages.

7.6.1.1 Hello request

The hello request message may be sent by the server at any time, but will be ignored by the client if the handshake protocol is already underway. It is a simple notification that the client should begin the negotiation process anew by sending a client hello message when convenient.

Note: Since handshake messages are intended to have transmission precedence over application data, it is expected that the negotiation begin in no more than one or two times the transmission time of a maximum length application data message.

After sending a hello request, servers should not repeat the request until the subsequent handshake negotiation is complete. A client that receives a hello request while in a handshake negotiation state should simply ignore the message.

The structure of a hello request message is as follows:

```
struct { } HelloRequest;
```

7.6.1.2 Client hello

When a client first connects to a server it is required to send the client hello as its first message. The client can also send a client hello in response to a hello request or on its own initiative in order to renegotiate the security parameters in an existing connection.

The client hello message includes a random structure, which is used later in the protocol.

```
struct {
  uint32 gmt_unix_time;
  opaque random_bytes[28];
} Random;
```

gmt_unix_time

The current time and date in standard UNIX 32-bit format according to the sender's internal clock. Clocks are not required to be set correctly by the basic SSL Protocol; higher level or application protocols may define additional requirements.

random_bytes

28 bytes generated by a secure random number generator.

The client hello message includes a variable length session identifier. If not empty, the value identifies a session between the same client and server whose security parameters the client wishes to reuse. The session identifier may be from an earlier connection, this connection, or another currently active

connection. The second option is useful if the client only wishes to update the random structures and derived values of a connection, while the third option makes it possible to establish several simultaneous independent secure connections without repeating the full handshake protocol. The actual contents of the SessionID are defined by the server.

opaque SessionID<0..32>;

Warning: Servers must not place confidential information in session identifiers or let the contents of fake session identifiers cause any breach of security.

The CipherSuite list, passed from the client to the server in the client hello message, contains the combinations of cryptographic algorithms supported by the client in order of the client's preference (first choice first). Each CipherSuite defines both a key exchange algorithm and a CipherSpec. The server will select a cipher suite or, if no acceptable choices are presented, return a handshake failure alert and close the connection.

uint8 CipherSuite[2]; /* Cryptographic suite selector */

The client hello includes a list of compression algorithms supported by the client, ordered according to the client's preference. If the server supports none of those specified by the client, the session must fail.

enum { null(0), (255) } CompressionMethod;

Issue: Which compression methods to support is under investigation.

The structure of the client hello is as follows.

```
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites<2..2^16-1>;
    CompressionMethod compression_methods<1..2^8-1>;
} ClientHello;
```

client_version

The version of the SSL protocol by which the client wishes to communicate during this session. This should be the most recent (highest valued) version supported by the client. For this version of the specification, the version will be 3.0 (See Appendix E for details about backward compatibility).

random

A client-generated random structure.

session_id

The ID of a session the client wishes to use for this connection. This field should be empty if no session_id is available or the client wishes to generate new security parameters.

cipher_suites

This is a list of the cryptographic options supported by the client, sorted with the client's first preference first. If the session_id field is not empty (implying a session resumption request) this vector must include at least the cipher_suite from that session. Values are defined in Appendix A.6.

compression_methods

This is a list of the compression methods supported by the client, sorted by client preference. If the session_id field is not empty (implying a session resumption request) this vector must include at least the compression_method from that session. All implementations must support CompressionMethod.null.

After sending the client hello message, the client waits for a server hello message. Any other handshake message returned by the server except for a hello request is treated as a fatal error.

Implementation note: Application data may not be sent before a finished message has been sent. Transmitted application data is known to be insecure until a valid finished message has been received. This absolute restriction is relaxed if there is a current, non-null encryption on this connection.

7.6.1.3 Server hello

The server processes the client hello message and responds with either a handshake_failure alert or server hello message.

```
struct {
    ProtocolVersion server_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suite;
    CompressionMethod compression_method;
} ServerHello;
```

server_version

This field will contain the lower of that suggested by the client in the client hello and the highest supported by the server. For this version of the specification, the version will be 3.0 (See Appendix E for details about backward compatibility).

random

This structure is generated by the server and must be different from (and independent of) ClientHello.random.

session_id

This is the identity of the session corresponding to this connection. If the ClientHello.session_id was non-empty, the server will look in its session cache for a match. If a match is found and the server is willing to establish the new connection using the specified session state, the server will respond with the same value as was supplied by the client. This indicates a resumed session and dictates that the parties must proceed directly to the finished messages. Otherwise this field will contain a different value identifying the new session. The server may return an empty session_id to indicate that the session will not be cached and therefore cannot be resumed.

cipher_suite

The single cipher suite selected by the server from the list in ClientHello.cipher_suites. For resumed sessions this field is the value from the state of the session being resumed.

compression_method

The single compression algorithm selected by the server from the list in ClientHello.compression_methods. For resumed sessions this field is the value from the resumed session state.

7.6.2 Server certificate

If the server is to be authenticated (which is generally the case), the server sends its certificate immediately following the server hello message. The certificate type must be appropriate for the selected cipher suite's key exchange algorithm, and is generally an X.509.v3 certificate (or a modified X.509 certificate in the case of Fortezza [FOR]). The same message type will be used for the client's response to a server certificate request message.

```
opaque ASN.1Cert<1..2^24-1>;

struct {
    ASN.1Cert certificate_list<1..2^24-1>;
} Certificate;
```

certificate_list This is a sequence (chain) of X.509.v3 certificates, ordered with the sender's certificate first and the root certificate authority last.

Note: PKCS #7 [PKCS7] is not used as the format for the certificate vector because PKCS #6 [PKCS6] extended certificates are not used. Also PKCS #7 defines a SET rather than a SEQUENCE, making the task of parsing the list more difficult.

7.6.3 Server key exchange message

The server key exchange message is sent by the server if it has no certificate, has a certificate only used for signing (e.g., DSS [DSS] certificates, signing-only RSA [RSA] certificates), or fortezza/DMS key exchange is used. This message is not used if the server certificate contains Diffie-Hellman [DH1] parameters.

Note: According to current US export law, RSA moduli larger than 512 bits may not be used for key exchange in software exported from the US. With this message, larger RSA keys may be used as signature-only certificates to sign temporary shorter RSA keys for key exchange.

```
enum { rsa, diffie_hellman, fortezza_dms } KeyExchangeAlgorithm;

struct {
      opaque rsa_modulus<1..2^16-1>;
      opaque rsa_exponent<1..2^16-1>;
} ServerRSAParams;
          rsa_modulus The modulus of the server's temporary RSA key.
          rsa_exponent The public exponent of the server's temporary RSA key.
struct {
      opaque dh_p<1..2^16-1>;
      opaque dh_g<1..2^16-1>;
      opaque dh_Ys<1..2^16-1>;
} ServerDHParams;      /* Ephemeral DH parameters */
```

dh_p

The prime modulus used for the Diffie-Hellman operation.

dh_g

The generator used for the Diffie-Hellman operation.

dh_Y_s

The server's Diffie-Hellman public value (g^X mod p).

```
struct {
         opaque r_s [128];
} ServerFortezzaParams;
```

r_s

Server random number for Fortezza KEA (Key Exchange Algorithm).

```
struct {
      select (KeyExchangeAlgorithm) {
         case diffie_hellman:
             ServerDHParams params;
             Signature signed_params;
         case rsa:
             ServerRSAParams params;
             Signature signed_params;
         case fortezza_dms:
             ServerFortezzaParams params;
      };
} ServerKeyExchange;
```

params

The server's key exchange parameters.

signed_params

A hash of the corresponding params value, with the signature appropriate to that hash applied.

md5_hash

MD5(ClientHello.random + ServerHello.random + ServerParams);

sha_hash

SHA(ClientHello.random + ServerHello.random + ServerParams);

```
enum { anonymous, rsa, dsa } SignatureAlgorithm;

digitally-signed struct {
      select(SignatureAlgorithm) {
          case anonymous: struct { };
          case rsa:
              opaque md5_hash[16];
              opaque sha_hash[20];
          case dsa:
              opaque sha_hash[20];
      };
} Signature;
```

7.6.4 Certificate request

A non-anonymous server can optionally request a certificate from the client, if appropriate for the selected cipher suite.

```
opaque CertificateAuthority<0..2^24-1>;

enum {
      rsa_sign(1), dss_sign(2), rsa_fixed_dh(3), dss_fixed_dh(4),
      rsa_ephemeral_dh(5), dss_ephemeral_dh(6), fortezza_dms(20), (255)
} ClientCertificateType;

opaque DistinguishedName<1..2^16-1>;

struct {
      ClientCertificateType certificate_types<1..2^8-1>;
      DistinguishedName certificate_authorities<3..2^16-1>;
} CertificateRequest;
```

certificate_types

This field is a list of the types of certificates requested, sorted in order of the server's preference.

certificate_authorities

A list of the distinguished names of acceptable certificate authorities.

Note: DistinguishedName is derived from [X509].
Note: It is a fatal handshake_failure alert for an anonymous server to request client identification.

7.6.5 Server hello done

The server hello done message is sent by the server to indicate the end of the server hello and associated messages. After sending this message the server will wait for a client response.

```
struct { } ServerHelloDone;
```

Upon receipt of the server hello done message the client should verify that the server provided a valid certificate if required and check that the server hello parameters are acceptable.

7.6.6 Client certificate

This is the first message the client can send after receiving a server hello done message. This message is only sent if the server requests a certificate. If no suitable certificate is available, the client should send a no certificate alert instead. This error is only a warning, however the server may respond with a fatal handshake failure alert if client authentication is required.

Client certificates are sent using the Certificate defined in Section 7.6.2.
Note: Client Diffie-Hellman certificates must match the server specified Diffie-Hellman parameters.

7.6.7 Client key exchange message

The choice of messages depends on which public key algorithm(s) has (have) been selected. See Section 7.6.3 for the KeyExchangeAlgorithm.

```
struct {
    select (KeyExchangeAlgorithm) {
      case rsa: EncryptedPreMasterSecret;
      case diffie_hellman: ClientDiffieHellmanPublic;
      case fortezza_dms: FortezzaKeys;
    } exchange_keys;
} ClientKeyExchange;
```

The information to select the appropriate record structure is in the pending session state (see Section 7.1).

7.6.7.1 RSA encrypted premaster secret message

If RSA is being used for key agreement and authentication, the client generates a 48-byte pre-master secret, encrypts it under the public key from the server's certificate or temporary RSA key from a server key exchange message, and sends the result in an encrypted premaster secret message.

```
struct {
    ProtocolVersion client_version;
    opaque random[46];
} PreMasterSecret;
```

client_version

The latest (newest) version supported by the client. This is used to detect version roll-back attacks.

random

46 securely-generated random bytes.

```
struct {
    public-key-encrypted PreMasterSecret pre_master_secret;
} EncryptedPreMasterSecret;
```

pre_master_secret

This random value is generated by the client and is used to generate the master secret, as specified in Section 8.1.

7.6.7.2 Fortezza key exchange message

Under Fortezza DMS, the client derives a Token Encryption Key (TEK) using Fortezza's Key Exchange Algorithm (KEA). The client's KEA calculation uses the public key in the server's certificate along with private parameters in the client's token. The client sends public parameters needed for the server to generate the TEK, using its own private parameters. The client generates session keys, wraps them using the TEK, and sends the results to the server. The client generates IV's for the session keys and TEK and sends them also. The client generates a random 48-byte premaster secret, encrypts it using the TEK, and sends the result:

```
struct {
    opaque y_c<0..128>;
    opaque r_c[128];
    opaque y_signature[20];
    opaque wrapped_client_write_key[12];
    opaque wrapped_server_write_key[12];
    opaque client_write_iv[24];
    opaque server_write_iv[24];
    opaque master_secret_iv[24];
    block-ciphered opaque encrypted_pre_master_secret[48];
} FortezzaKeys;
```

y_signature

y_singnature is the signature of the KEA public key, signed with the client's DSS private key.

y_c

The client's Y_c value (public key) for the KEA calculation. If the client has sent a certificate, and its KEA public key is suitable, this value must be empty since the certificate already contains this value. If the client sent a certificate without a suitable public key, y_c is used and y_singnature is the KEA public key signed with the client's DSS private key. For this value to be used, it must be between 64 and 128 bytes.

r_c

The client's R_c value for the KEA calculation.

wrapped_client_write_key

This is the client's write key, wrapped by the TEK.

wrapped_server_write_key

This is the server's write key, wrapped by the TEK.

client_write_iv

This is the IV for the client write key.

server_write_iv

This is the IV for the server write key.

master_secret_iv

This is the IV for the TEK used to encrypt the pre-master secret.

pre_master_secret

This is a random value, generated by the client and used to generate the master secret, as specified in Section 8.1. In the above structure, it is encrypted using the TEK.

7.6.7.3 Client Diffie-Hellman public value

This structure conveys the client's Diffie-Hellman public value (Y_c) if it was not already included in the client's certificate. The encoding used for Y_c is determined by the enumerated PublicValueEncoding.

```
enum { implicit, explicit } PublicValueEncoding;
```

implicit

If the client certificate already contains the public value, then it is implicit and Y_c does not need to be sent again.

explicit

Y_c needs to be sent.

```
struct {
    select (PublicValueEncoding) {
        case implicit: struct { };
        case explicit: opaque dh_Yc<1..2^16-1>;
    } dh_public;
} ClientDiffieHellmanPublic;
```

dh_Yc

The client's Diffie-Hellman public value (Y_c).

7.6.8 Certificate verify

This message is used to provide explicit verification of a client certificate. This message is only sent following any client certificate that has signing capability (i.e. all certificates except those containing fixed Diffie-Hellman parameters).

```
struct {
    Signature signature;
} CertificateVerify;
```

CertificateVerify.signature.md5_hash
MD5(master_secret + pad2 + MD5(handshake_messages +
master_secret + pad1));

Certificate.signature.sha_hash
SHA(master_secret + pad2 + SHA(handshake_messages +
master_secret + pad1));
Here handshake_messages refers to all handshake messages starting at client hello up to but not including this message.

7.6.9 Finished

A finished message is always sent immediately after a change cipher specs message to verify that the key exchange and authentication processes were successful. The finished message is the first protected with the just-negotiated algorithms, keys, and secrets. No acknowledgment of the finished message is required; parties may begin sending confidential data immediately after sending the finished message. Recipients of finished messages must verify that the contents are correct.

```
enum { client(0x434C4E54), server(0x53525652) } Sender;

struct {
    opaque md5_hash[16];
    opaque sha_hash[20];
} Finished;
```

md5_hash
MD5(master_secret + pad2 + MD5(handshake_messages +
Sender + master_secret + pad1));

sha_hash
SHA(master_secret + pad2 + SHA(handshake_messages +
Sender + master_secret + pad1));

The hash contained in finished messages sent by the server incorporate Sender.server; those sent by the client incorporate Sender.client. The value handshake_messages includes all handshake messages starting at client hello up to, but not including, the finished messages. This may be different from handshake_messages in Section 7.6.8 because it would include the certificate verify message (if sent).
Note: Change cipher spec messages are not handshake messages and are not included in the hash computations.

7.7 Application data protocol

Application data messages are carried by the Record Layer and are fragmented, compressed and encrypted based on the current connection state. The messages are treated as transparent data to the record layer.

---

8. Cryptographic computations

The key exchange, authentication, encryption, and MAC algorithms are determined by the cipher_suite selected by the server and revealed in the server hello message.

8.1 Asymmetric cryptographic computations

The asymmetric algorithms are used in the handshake protocol to authenticate parties and to generate shared keys and secrets.
For Diffie-Hellman, RSA, and Fortezza, the same algorithm is used to convert the pre_master_secret into the master_secret. The pre_master_secret should be deleted from memory once the master_secret has been computed.

master_secret =
MD5(pre_master_secret + SHA('A' + pre_master_secret +
ClientHello.random + ServerHello.random)) +

MD5(pre_master_secret + SHA('BB' + pre_master_secret +
ClientHello.random + ServerHello.random)) +
MD5(pre_master_secret + SHA('CCC' + pre_master_secret +
ClientHello.random + ServerHello.random));

8.1.1 RSA

When RSA is used for server authentication and key exchange, a 48-byte pre_master_secret is generated by the client, encrypted under the server's public key, and sent to the server. The server uses its private key to decrypt the pre_master_secret. Both parties then convert the pre_master_secret into the master_secret, as specified above.
RSA digital signatures are performed using PKCS #1 [PKCS1] block type 1. RSA public key encryption is performed using PKCS #1 block type 2.

8.1.2 Diffie-Hellman

A conventional Diffie-Hellman computation is performed. The negotiated key (Z) is used as the pre_master_secret, and is converted into the master_secret, as specified above.
Note: Diffie-Hellman parameters are specified by the server, and may be either ephemeral or contained within the server's certificate.

8.1.3 Fortezza

A random 48-byte pre_master_secret is sent encrypted under the TEK and its IV. The server decrypts the pre_master_secret and converts it into a master_secret, as specified above. Bulk cipher keys and IVs for encryption are generated by the client's token and exchanged in the key exchange message; the master_secret is only used for MAC computations.

8.2 Symmetric cryptographic calculations and the CipherSpec

The technique used to encrypt and verify the integrity of SSL records is specified by the currently active CipherSpec. A typical example would be to encrypt data using DES and generate authentication codes using MD5. The encryption and MAC algorithms are set to SSL_NULL_WITH_NULL_NULL at the beginning of the SSL Handshake Protocol, indicating that no message authentication or encryption is performed. The handshake protocol is used to negotiate a more secure CipherSpec and to generate cryptographic keys.

8.2.1 The master secret

Before secure encryption or integrity verification can be performed on records, the client and server need to generate shared secret information known only to themselves. This value is a 48-byte quantity called the master secret. The master secret is used to generate keys and secrets for encryption and MAC computations. Some algorithms, such as Fortezza, may have their own procedure for generating encryption keys (the master secret is used only for MAC computations in Fortezza).

8.2.2 Converting the master secret into keys and MAC secrets

The master secret is hashed into a sequence of secure bytes, which are assigned to the MAC secrets, keys, and non-export IVs required by the current CipherSpec (see Appendix A.7).
CipherSpecs require a client write MAC secret, a server write MAC secret, a client write key, a server write key, a client write IV, and a server write IV, which are generated from the master secret in that order. Unused values, such as Fortezza keys communicated in the KeyExchange message, are empty. The following inputs are available to the key definition process:
opaque MasterSecret[48]
ClientHello.random
ServerHello.random
When generating keys and MAC secrets, the master secret is used as an entropy source, and the random values provide unencrypted salt material and IVs for exportable ciphers.
To generate the key material, compute
key_block =
MD5(master_secret + SHA('A' + master_secret + ServerHello.random +
ClientHello.random)) +
MD5(master_secret + SHA('BB' + master_secret + ServerHello.random +
ClientHello.random)) +
MD5(master_secret + SHA('CCC' + master_secret + ServerHello.random +
ClientHello.random)) + [...];

until enough output has been generated. Then the key_block is partitioned as follows.

```
client_write_MAC_secret[CipherSpec.hash_size]
server_write_MAC_secret[CipherSpec.hash_size]
client_write_key[CipherSpec.key_material]
server_write_key[CipherSPec.key_material]
client_write_IV[CipherSpec.IV_size]          /* non-export ciphers */
server_write_IV[CipherSpec.IV_size]          /* non-export ciphers */
```

Any extra key_block material is discarded.

Exportable encryption algorithms (for which CipherSpec.is_exportable is true) require additional processing as follows to derive their final write keys:

```
final_client_write_key = MD5(client_write_key +
    ClientHello.random + ServerHello.random);
final_server_write_key = MD5(server_write_key +
    ServerHello.random + ClientHello.random);
```

Exportable encryption algorithms derive their IVs from the random messages:

```
client_write_IV = MD5(ClientHello.random + ServerHello.random);
server_write_IV = MD5(ServerHello.random + ClientHello.random);
```

MD5 outputs are trimmed to the appropriate size by discarding the least-significant bytes.

8.2.2.1 Export key generation example

SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5 requires five random bytes for each of the two encryption keys and 16 bytes for each of the MAC keys, for a total of 42 bytes of key material. MD5 produces 16 bytes of output per call, so three calls to MD5 are required. The MD5 outputs are concatenated into a 48-byte key_block with the first MD5 call providing bytes zero through 15, the second providing bytes 16 through 31, etc. The key_block is partitioned, and the write keys are salted because this is an exportable encryption algorithm.

$$client\_write\_MAC\_secret = key\_block_{0..15}$$
$$server\_write\_MAC\_secret = key\_block_{16..31}$$
$$client\_write\_key = key\_block_{32..36}$$
$$server\_write\_key = key\_block_{37..41}$$

$$final\_client\_write\_key = MD5(client\_write\_key + ClientHello.random + ServerHello.random)_{0..15};$$
$$final\_server\_write\_key = MD5(server\_write\_key + ServerHello.random + ClientHello.random)_{0..15};$$

$$client\_write\_IV = MD5(ClientHello.random + ServerHello.random)_{0..7};$$
$$server\_write\_IV = MD5(ServerHello.random + ClientHello.random)_{0..7};$$

Security of eCommerce Systems

A Netcraft White Paper
May 2001

Abstract

In the headlong rush to bring eCommerce applications to market, many companies are neglecting the security of their systems, applications and confidential data. This paper explores some of the most common vulnerabilities that Netcraft security consultants uncover when performing eCommerce security evaluations for clients. It is not intended to be a treatise on computer, application or web security per se, more a collection of examples of real errors and system misconfigurations from which, it is hoped, system

designers, developers and managers can glean a few ideas to prevent their eCommerce systems from becoming yet another statistic for the archives. In Netcraft's experience, it is very often an organization's attitude towards security that is the problem more than the security of their eCommerce offering itself.

Contents

Introduction

One of the hardest aspects of the security evaluation work carried out by Netcraft is explaining to clients, eager to join the "eCommerce revolution", that the web-based system that they have spent many hours planning, designing and implementing, contains so many security vulnerabilities that a major re-working is required before the system will be ready for deployment on the Internet. In many cases, the flaws are so elementary that it is surprising that the system was ever considered good enough to even reach the security evaluation phase. Assuming it exists at all, the security evaluation can usually be found on the last page of the project plan, leaving no option but to re-build the entire project from scratch when security is found to be poor. This is not exaggeration for effect, as this paper will attempt to show. We will outline some of the problems that Netcraft consultants have uncovered in recent months and highlight some of the more obvious traps and pitfalls into which the unwary stumble on an all-too-regular basis.

Few of the problems presented here are inherent in any particular web server or operating system, provided that the basic security guidelines are followed. The major cause of system compromise is that the designers of a web service have only considered security as an afterthought, not as a key component of the system throughout the project's life cycle.

Security, What Security?

OK, so you're reading this to see how naïve some organizations can be when they first encounter the Internet or how even experienced developers fail to apply sufficient thought to the security of web-based services . We'll try not to disappoint, but the main aim of this paper is not to spread doom-and-gloom but to try to highlight some basic principles that will assist you during the development of your web-based applications. Throughout this paper, we use the term application in preference to web-site, since many of the mistakes made by web-site designers would be eliminated if the operating system, web

server, HTML pages, data entry forms, server-side scripts, databases and whatever else makes up your web-site, are treated as an application delivering a service to your customers, with due consideration of the overall security of the system. It is not sufficient to consider the security of parts of a system without looking at the security of the whole system. As we will see, failure to assess the risks from this viewpoint (indeed, from the viewpoint of someone with malicious intent), leads easily to a system open to compromise.

Plan Your Security Carefully

First rule - "Think security". Any computer system that is connected to the Internet, be it a multi-million dollar server farm or a single machine costing a few hundred dollars, is available for anyone to look at. That's as it should be. The ubiquity of the Internet means that your business, your ideas, or merely your idle ramblings, are available to whoever is interested and provides a great opportunity for your business to thrive in the global marketplace. Of course, this visibility comes at a price. Not everyone will like your ideas or your products and some will have the wherewithal to cause disruption to your service. Others represent the digital graffiti artists, keen to increase the spread of their name at your expense.

> Two prestigious organizations have recently scrapped eCommerce applications, written by third-party developers, with less than a week to go to the scheduled launch date. Our consultants were called in late in the day - too late to influence the development of the applications. As a result of our analysis and recommendations, both applications were re-designed and re-written at a cost of tens of thousands of dollars.

At whatever level your eCommerce offering is pitched, make sure that you consider the security of your service from day one. If you don't have the experience or knowledge in-house, call in a professional security consulting firm during the system's design. Work with them through the system development to ensure that you safeguard your investment in your Internet services.

Follow the Basic Guidelines

Sounds simple, doesn't it? With so much publicity about high-profile break-ins, defacement of web-sites and theft of credit card details, one would hope that anyone considering entering this perilous environment would, at least, take basic precautions against unwanted intrusion. Again, in our experience, this is far from the case. Over the past few years, Netcraft has developed its Security Analysis Services, designed specifically, at the simplest level, to detect common web-server misconfigurations and vulnerabilities or, at the most detailed level, to find complex, subtle loopholes in web application designs, implementation or configuration. The basic service has been offered to organizations setting up secure eCommerce environments (based around the use of SSL to encrypt transactions between the client and the web-based application). Given that the organizations concerned have already decided that the security of their customers' transactions is important, it is astonishing how many of the sites examined are still open to attack using well-known exploits.

Of the SSL sites tested in the last

nine months, 62% of sites tested are based around the Microsoft NT/4, IIS/4.0 or Windows2000, IIS/5.0 platforms, of which:

- A staggering 27% of sites are vulnerable to either the Unicode canonicalization or URL double decode vulnerabilities, which leave them wide open for anyone to run arbitrary commands on the server
- 28% of sites still have the IIS sample pages installed, exposing a number of loopholes that can be used by attackers to obtain system information or privileges
- 20% are vulnerable to other loopholes that allow server-side scripts to be examined by potential attackers.

Figures from Alldas show that, in the first 5 months of 2001, over 11,000 web-sites have been defaced, but it is clear that many more systems are "quietly" compromised without publicity (and even without the knowledge of the owners of the systems themselves).

> By not following the basic steps to secure your service, you leave yourself open to a variety of attacks from remote users. Indeed, you may even be contributing to the problem by allowing your system to become a stepping stone from which attacks against other systems can be launched. Many of you will have read about the Distributed Denial of Service (DDoS) attacks on high profile eCommerce sites - the only way for these attacks to continue is the proliferation of badly secured systems.

For IIS/4.0, Microsoft has produced a checklist (and set of guidelines) that all web-site developers should read and follow closely. The guidelines provide precise configuration instructions that must be followed before an eCommerce application is deployed. For IIS/5.0 on Windows 2000, the security configuration tool is a useful starting point for securing your system. More details can be found at http://www.microsoft.com/technet/security/tools.asp. Other eCommerce platform suppliers provide similar security advice, so take advantage of it.

Keep it Simple

Many eCommerce systems that we come across when performing security evaluations have large numbers of (often vulnerable) services open to all and sundry. If the sole purpose of your web-based application is to provide normal and secure channels for your customers to communicate with you, then it is highly likely that you can make do with opening TCP port 80 (HTTP) and TCP port 443 (HTTPS) for access by anyone. Why then, do we see many, many sites with many other ports open to the world, providing many avenues of attack against the site? The answer is obvious - security has not been adequately considered before the system and its application have been launched into the eCommerce world.

> In a recent examination of the security of a new high profile "free" Internet service, the service was found to provide a different kind of freedom, as the NetBIOS services allowed us to remotely map their hard disk systems with full read and write access. Even though we needed a username and password to connect, these were not hard to guess (the "administrator" account password was, ahem, "password".)

> In the rush to embrace the web, many long established computer security principles have been abandoned. Remember, deny access to all services and then allow access to only those services necessary for your application to function correctly. Furthermore, exposed services should be examined in detail to ensure that they conform to an accepted level of security.

What they See is What you Should Protect

That's not to say that you should become complacent, but assuming that your firewall/port filtering systems adequately protect your server by restricting access to all but a limited number of services, it is clear that it is precisely those services that an attacker will target in order to compromise your

application. Indeed, many of the recent high profile attacks have used frailties in server and application software to penetrate a web site.

Many advertisements in the papers or on television press home the fact that setting up a web-site is an easy task. In many cases, it is - but what the adverts fail to mention is that configuring a secure system requires a reasonable understanding of the risks involved. Of course the business world has to balance risk against the cost of protecting itself against that risk. Many professional security companies offer a range of risk assessment and risk management services and these should be factored into your project's budget wherever possible. However, there are some simple steps that can be taken to reduce such risk and most suppliers of Internet-based hardware and software solutions will provide such advice - all you have to do is follow that advice.

If we look at some of the most popular eCommerce platforms, the same risky configurations occur time after time, largely because the default installation of the platform has not been prepared specifically for Internet deployment.

Sample Scripts and Applications

One of the most frequently observed problems does, indeed, relate to the deployment of standard installations without any real understanding of what is being offered to external users. Simply leaving sample files and applications installed on a server can have dire consequences. Very often, example scripts contain code that is designed to show the advanced capabilities of a system but will also allow intruders to abuse these capabilities to view files on the web server and discover the structure of the file-system, thus providing valuable information with which to locate sensitive data files or launch further attacks against the system.

> Netcraft finds that 3 out of every 10 web-sites tested allows access to example server-side scripts with known exploitable weaknesses.

Tidy Up After Yourself

Very many web-based applications are developed and tested on the actual system that will be put into production when the project is ready. A couple of dangers are immediately obvious - the developers will be working on a system that is not designed to be secure during development and may well have an open FTP service, FrontPage facilities, Telnet, remote database connections, etc. and may even have been set up with explicit or implicit trust of other machines in the network. In the rush to bring product to market, the existence of some services is often overlooked and those services remain accessible by anyone, albeit with a much wider audience than before!

> Failure to "tidy up" before a system went live allowed our consultants to retrieve the entire source of a trading application from the web-site of a financial services company, including database logins and passwords and confidential customer data files.

Once the development (and, hopefully, application testing), has been completed, it is imperative that editor backup files, other files used during

development and any non-essential data is removed from the web folders on the server. Many "hacker scripts" test for known back-up files to try and retrieve the source code of scripts and learn more about an application and its weaknesses.

> Files discovered by our consultants in publicly-accessible areas include confidential e-mail messages, router and firewall configuration data, application data files and, of course, database logins and passwords.

Protect Your Sensitive Data

You'd have to say that protecting sensitive data is an obvious requirement of any online system. By now, you won't be surprised to learn that many sites blithely ignore such trivialities. Netcraft often tests sites where confidential customer data is stored unencrypted in files within the web-server's file space or unprotected database tables.

> In a recent eCommerce Security Evaluation, Netcraft managed to use a loophole in a system module to read files in the web-server's file space. One such file contained customer bank account details in an encrypted form. The encryption key, however, was hard-coded into the application's scripts.

Manage Your Sessions Carefully

There have been several cases reported in the last few months where per-session data from one user of a web-site has been erroneously delivered to another user. In each case, the reason for these errors is usually given as a software "glitch" or malfunction, although many are really caused by fundamental flaws in application design. Designing multi-user interactive data-centric applications has always provided some stiff challenges for software systems developers and requires experience and care if the subtleties of complex interactions are to be both understood and handled in a proper fashion. Unless the development of web-based applications is approached with the same degree of care, security and performance problems are bound to occur.

Common design errors seen by Netcraft are mismanagement of session state, usually caused by a failure to use and understand the facilities provided by the run-time environment of the tools used for the application. As an example, we often see hidden fields in HTML forms (or worse still, parameters within URLs) used to select and maintain data on the server, when server-side session variables, or their equivalent, are not used at all. Failure of applications to send correct instructions to caching proxy servers can also lead to unexpected data dissemination.

Shhh, Don't Tell Everyone

Trying to "break-in" to a web-based service involves some automated procedures to locate and test for vulnerabilities in exposed systems and services, followed by manual inspections of those services and the application itself, both from an external (user's) viewpoint and by reviewing application

source code for common design faults and coding errors. Very often, however, the secret to unlocking a system vulnerability lies in the gathering together of fragments of information from (sometimes) unexpected places.

> Using a list of contact names in a "forgotten" part of a company's web-site, Netcraft was able to find several other companies with which the owner of one of the e-mail addresses had previously been associated. Once such association led to his personal home page, where we discovered his surname, his wife's name and even the name of his cat. This candidate information actually gave us an FTP login name on the targeted site. Fortunately for the site in question, their password policy was sufficiently strict that it resisted our attempts to probe further. Nevertheless, the site had provided more information than its creators intended.

People intent on compromising or defacing any web-based system will always have more time than you would believe possible to explore every nook and cranny of your web-site. Comments in HTML pages will often provide useful information to an attacker, such as the developer's login name or clues as to the structure of the application and the web-server. Other sources of useful data include not only the obvious Whois entries, press releases, news items, contact lists, etc., but also the less obvious partner web-sites and personal web-pages. By far the greatest risk, however, comes from badly configured server software and application errors, particularly errors in scripts that are used to interact with back-end databases.

> By providing invalid data that will form part of a database query on the remote service, our security evaluation team can often force error messages to be displayed by the target application. In many cases, these give details of the structure of the web-site, its directories and, most worryingly, the structure of the database tables used by the application. Once the structure has been determined, it is often a simple matter to modify database records to elevate privileges or to steal or disrupt data in the database.

Once again, failure to think about the ways in which your system and application could be abused can lead to simple ways for an attacker to cause disruption. Another common mistake is the failure to properly validate data from end-users and then using this potentially damaging data in the application. All data presented to an application must be considered "tainted" until the application itself has ensured that it is safe. Things may not always be what they seem - HTTP headers, cookies, hidden form variables, IP addresses, and so on, can all be faked; client-side form data validation can be bypassed and URLs can be modified.

> One eCommerce evaluation carried out by Netcraft found that a simple modification to the parameters that formed

part of a URL would allow us to view customer orders and invoices for any user of the system.

Just when you think that you've got your system "just so", there's the final, unexpected configuration problem that makes you wonder how on earth you missed such a simple thing. The classic example is perhaps the use of a search facility on your web-site. It may seem obvious to index (that is, include in the search) web-pages that you want your users to be able to search, but it is very common to find all manner of files indexed. Search results that include file/page summaries will show the contents of system files, configuration files, etc. if they have been mistakenly included in the search database. Even if your search engine only displays lists of pages found, you may still be giving away more information than you expect. Be careful which files are looked at by your search facility.

"Thanks, now I can sleep nights"

Many IT directors, CTOs and system administrators do sleep peacefully - some because they have paid due consideration to the security of their on-line systems and applications during their design, development and deployment, but many because they do not yet understand the risks that their new eCommerce system is facing as they sleep. Compared to the cost of cleaning up the mess after a system has been broken into, the cost of testing the security of your web application is "small beer".

> A system that cost around $5,000 to put together, recently had its home-page defaced. To date, the cost of investigating the defacement, rebuilding and (this time) testing the security of the new system, is in excess of $50,000.

Here are some of the things that we have heard over the past year. How highly these people rated the security of their systems, and at what stage in their assessment of the problem they had reached, is left as an exercise for the reader.

- "We only have a few static web-pages, so we can't be vulnerable."
- "My SSL pages are private, why are you looking at THEM?"
- "Thanks, now I can sleep nights."
- "Why do I need to test my application, I've got a firewall."
- "Wow! That was sneaky. I'm amazed you found that."
- "I know we're vulnerable, but it's not a major issue for us."

Conclusion

In the process of surveying hundreds of web applications, Netcraft sees many security flaws - some surprising, some expected, some so subtle that they aren't immediately obvious - all potentially damaging to the organisation behind the web presence or its customers and investors. Is there a common theme? Well, yes and no. Whilst the variety of design, implementation or configuration

problems never ceases to amaze, the underlying problem is the general lack of consideration of security as a fundamental part of the design of a web-based application from its inception and throughout its life.

You might suppose that the entire blame can be laid at the door of the suppliers of system software, the designers of an application or the developers who put the whole thing together. Whilst these groups often contribute to the overall problem, they are not always solely to blame. Suppliers are nowadays taking responsible steps to ensure that their application environments provide a secure base, upon which a web application can be developed. System designers are (or certainly should be) aware of the perils awaiting their application when it is launched into the hostile Internet environment. Very often, programmers do not have sight of the entire application and cannot involve themselves in the overall security of the system.

Often, however, the last say in bringing an application to the web lies with the organization's management and they, of course, will have investors, customers and competitors influencing the application's time-to-market. Until the people commissioning new web-sites, applications and web-based businesses understand the importance of security and work together with the suppliers, designers and developers to ensure that security is uppermost in everyone's minds, the hackers, crackers, script kiddies - call them what you will - will continue to flourish.

About Netcraft

Netcraft is an internet consultancy based in Bath, England with the majority of its work closely associated to the development of Internet services for its clients or itself. In 1999, Deloitte & Touche ranked Netcraft as one of the fifty fastest growing technology companies in the United Kingdom. In November 1999, the British Department of Trade and Industry short-listed Netcraft for an award for export achievement.

Bob Metcalfe, inventor of Ethernet and co-founder of 3com, has pronounced Netcraft "cool", while Tim O'Reilly called it "the best known example of a site devoted to tracking technology on the Internet".

Clients for Netcraft's services include IBM, Sun, Hewlett Packard, Microsoft, Intel, Verisign, Global Crossing, Energis, British Telecom, Cable & Wireless, Demon Internet, Goldman Sachs, Morgan Stanley, Fidelity, Union Bank of Switzerland, the Post Office and Lloyds of London.