Integration Points: Gateways, Tunnels, and Relays

وب ثابت کرده است که ابزاری باورنکردنی برای انتشار محتوا است. با گذشت زمان، تمایل افراد در ایجاد محتوا از قرار دادن اسناد ثابت آنلاین به تمایل به اشتراک گذاری منابع پیچیده‌تر، مانند محتوای پایگاه داده یا صفحات HTML به صورت پویا تغییر کرده است. برنامه‌های کاربردی HTTP، مانند مرورگرهای وب، ابزار یکپارچه‌ای را برای دسترسی به محتوا از طریق اینترنت در اختیار کاربران قرار داده اند.

HTTP همچنین به یک بلوک اساسی برای توسعه دهندگان برنامه تبدیل شده است، که پروتکل‌های دیگر را در بالای HTTP پشت سر می‌گذارند (به عنوان مثال، استفاده از HTTP برای تونل کردن یا انتقال ترافیک پروتکل‌های دیگر از طریق فایروال‌ها، با قرار دادن آن ترافیک در HTTP). HTTP به عنوان یک پروتکل برای تمام منابع وب استفاده می‌شود و همچنین پروتکلی است که سایر برنامه‌ها و پروتکل‌های اپلیکیشن از آن برای انجام کارهای خود استفاده می‌کنند.

این فصل نگاهی کلی به برخی از روش‌هایی دارد که توسعه‌دهندگان برای استفاده از HTTP برای دسترسی به منابع مختلف ارائه کرده‌اند و بررسی می‌کند که چگونه توسعه‌دهندگان از HTTP به عنوان چارچوبی برای فعال کردن سایر پروتکل‌ها و ارتباطات برنامه‌ها استفاده می‌کنند.

در این فصل به بحث زیر می‌پردازیم:

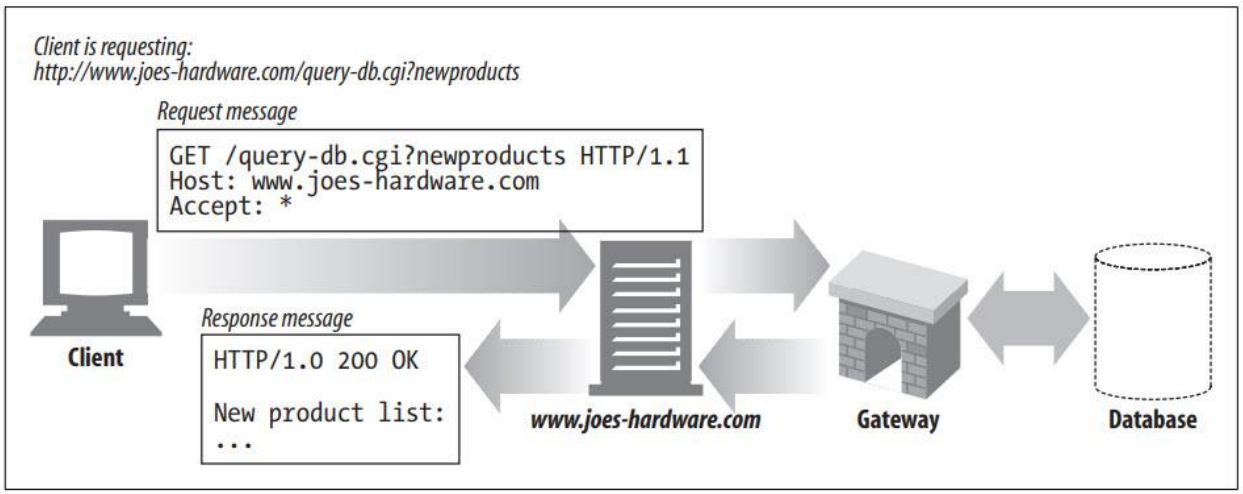
- Gatewayهایی که HTTP را با سایر پروتکل‌ها و برنامه‌ها مرتبط می‌کنند.
- Interface های برنامه، که به انواع مختلف برنامه‌های کاربردی وب اجازه می‌دهد تا با یکدیگر ارتباط برقرار کنند.
- تونل‌هایی که به شما امکان می‌دهند ترافیک غیر HTTP را از طریق اتصالات HTTP ارسال کنید.
- Relay یا رله‌ها، که یک نوع پراکسی HTTP ساده‌شده هستند و برای ارسال داده‌ها در یک بار پرش(Hop) استفاده می‌شوند.

Gateways

تاریخچه پشت Extention ها و رابط‌های HTTP بر اساس نیازهای افراد ایجاد شده است. هنگامی که تمایل به قرار دادن منابع پیچیده‌تر در وب پدیدار شد، به سرعت مشخص شد که هیچ برنامه کاربردی واحدی نمی‌تواند تمام منابع قابل تصور را مدیریت کند.

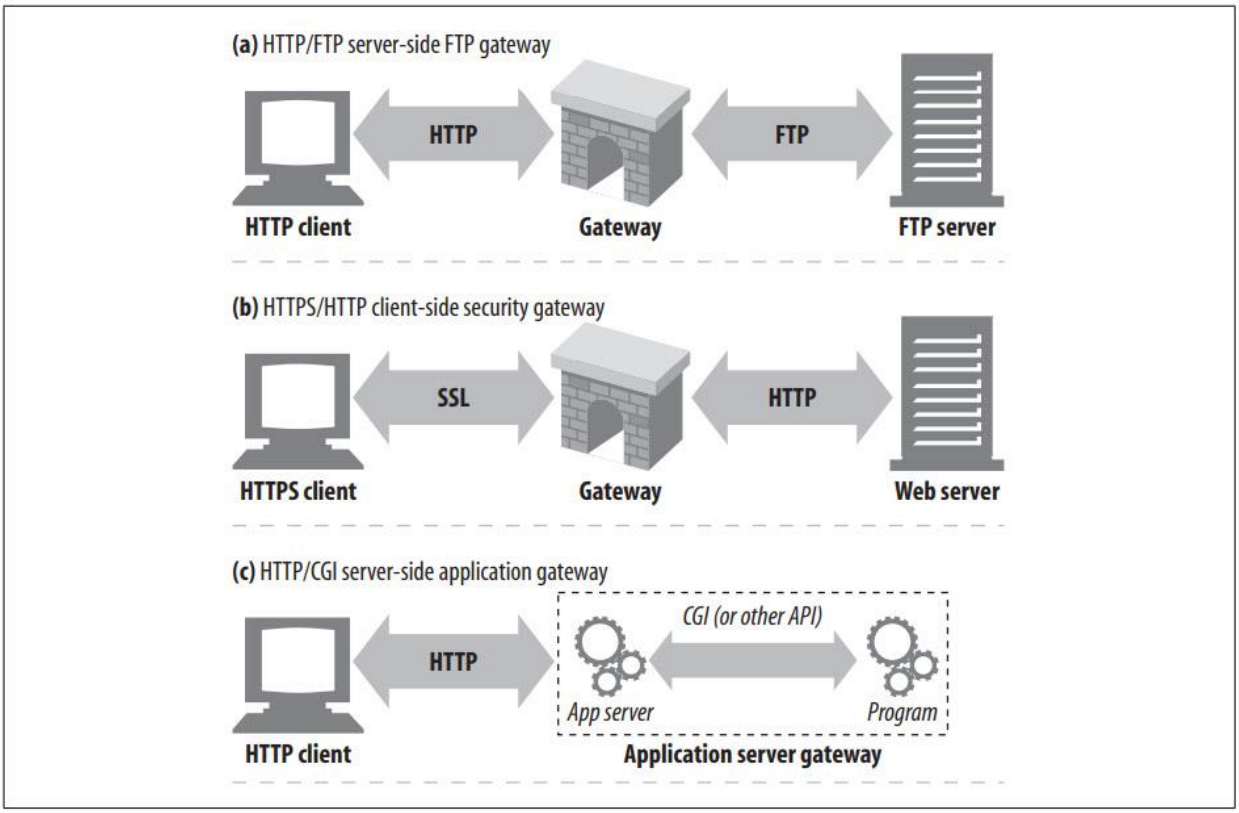
برای دور زدن این مشکل، توسعه‌دهندگان به مفهوم Gateway رسیدند که می‌تواند به‌عنوان نوعی مفسر عمل کند و راهی را برای دستیابی به منبع ارائه نماید. Gateway در واقع یک چسب بین منابع و برنامه‌ها است. یک برنامه کاربردی می‌تواند (از طریق HTTP یا یک Interface تعریف شده دیگر) از یک Gateway برای رسیدگی به درخواست، بپرسد و Gateway می‌تواند پاسخی ارائه دهد. Gateway می‌تواند با زبان پرس و جو به پایگاه داده صحبت کند یا محتوای پویا را تولید کند و مانند یک پورتال عمل کند: یک درخواست وارد می‌شود و یک پاسخ بیرون می‌آید.

شکل زیر نوعی Resource Gateway را نشان می‌دهد. در اینجا، سرور سخت افزار Joe به عنوان دروازه ای برای محتوای پایگاه داده عمل می‌کند - توجه داشته باشید که کلاینت به سادگی از طریق HTTP منبعی را درخواست می‌کند و سرور سخت افزار Joe با یک Gateway برای دسترسی به منبع ارتباط برقرار می‌کند.



برخی از Gateway ها به‌طور خودکار ترافیک HTTP را به پروتکل‌های دیگر ترجمه می‌کنند، بنابراین کلاینت‌های HTTP می‌توانند با برنامه‌های کاربردی دیگر ارتباط برقرار کنند بدون اینکه کلاینت‌ها نیازی به دانستن پروتکل‌های دیگر داشته باشند.

شکل زیر سه نمونه از Gateway ها را نشان می‌دهد:



در بخش a از شکل بالا، Gateway درخواست‌های HTTP را برای URL های FTP دریافت می‌کند. سپس دروازه اتصالات FTP را باز می‌کند و دستورات مناسب را به سرور FTP صادر می‌کند. سند از طریق HTTP به همراه هدرهای صحیح HTTP بازگردانده می‌شود.

در بخش b از شکل بالا، Gateway یک درخواست وب رمزگذاری شده را از طریق SSL دریافت می‌کند، درخواست را رمزگشایی می‌کند و یک درخواست HTTP معمولی را به سرور مقصد ارسال می‌کند. این شتاب دهنده‌های امنیتی را می‌توان مستقیماً در مقابل سرورهای وب (معمولاً در همان محل) قرار داد تا رمزگذاری با کارایی بالا را برای سرورهای مبدا فراهم کند.

در بخش c شکل بالا، Gateway، مشتریان HTTP را از طریق یک Application Server Gateway API، به برنامه‌های کاربردی سمت سرور متصل می‌کند. وقتی از فروشگاه‌های تجارت الکترونیک در وب خرید می‌کنید، پیش‌بینی آب و هوا را بررسی می‌کنید یا قیمت‌های سهام را دریافت می‌کنید، در حال بازدید از Application Server Gateway ها هستید.

Client-Side and Server-Side Gateways

Gateway های وب از یک طرف HTTP و در طرف دیگر با پروتکل متفاوتی صحبت می‌کنند.

<client-protocol>/<server-protocol>

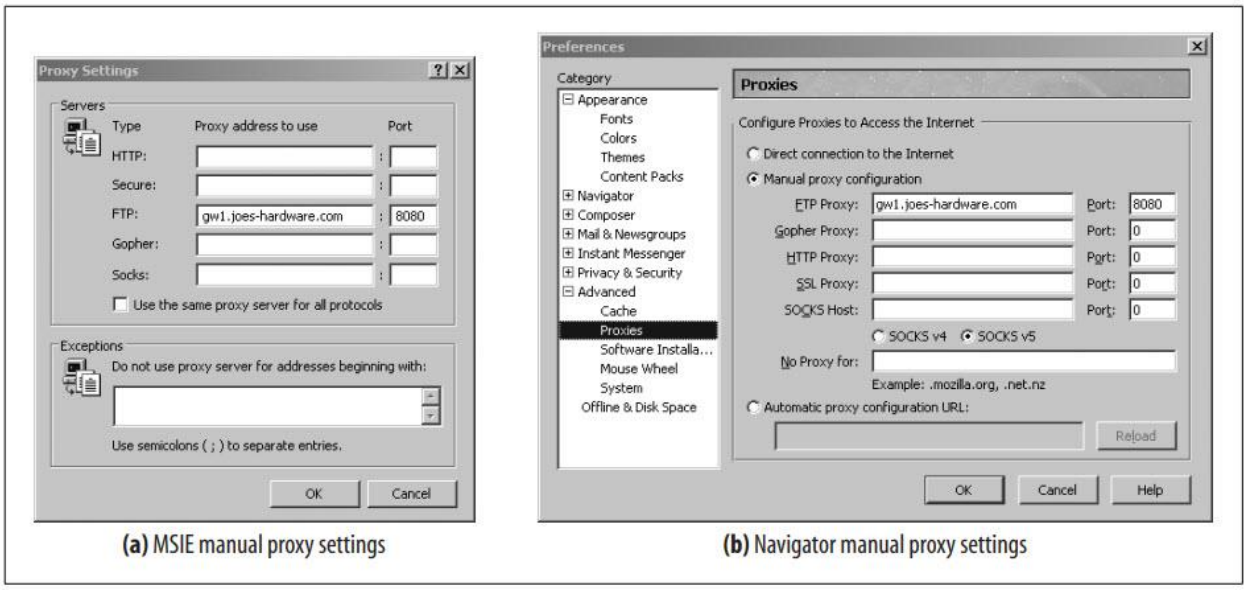
بنابراین Gateway ای که کلاینت‌های HTTP را به سرورهای خبری NNTP پیوند می‌دهد، یک HTTP/NNTP Gateway است. ما از اصطلاحات "Server-Side Gateway" و " Client-Side Gateway" برای توضیح اینکه تبدیل برای کدام سمت Gateway انجام می‌شود استفاده می‌کنیم:

- Server-Side Gateway ها بوسیله HTTP با کلاینت‌ها و یک پروتکل خارجی با سرورها (HTTP/*) صحبت می‌کنند.
- Client-Side Gateway ها بوسیله پروتکل‌های خارجی با کلاینت‌ها و HTTP با سرورها (*/HTTP) صحبت می‌کنند.

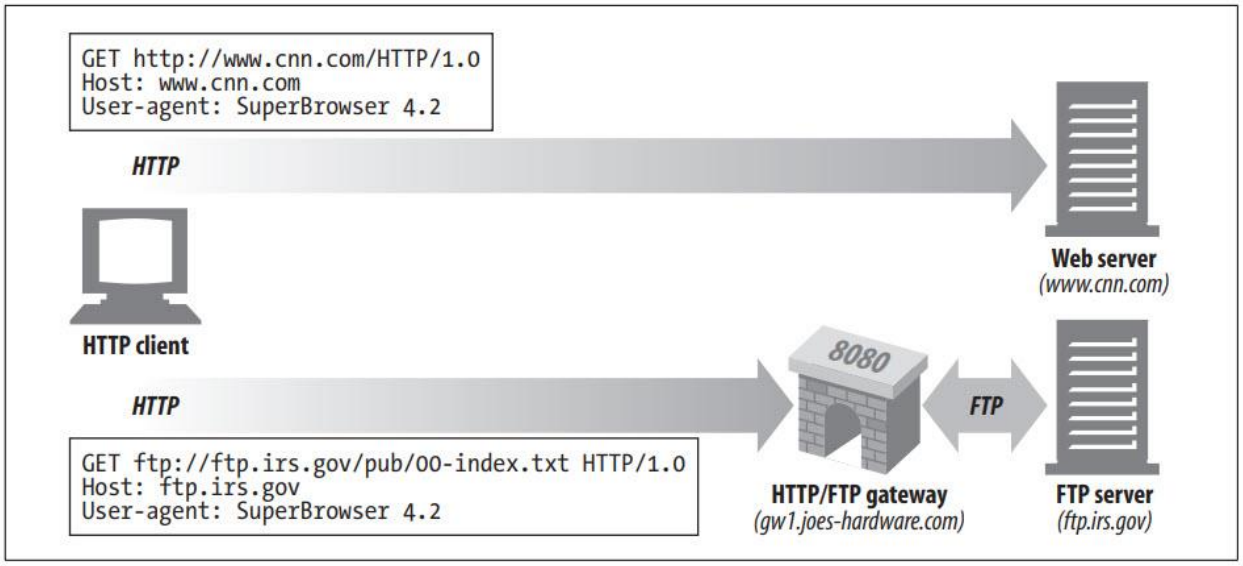
Protocol Gateways

شما می‌توانید ترافیک HTTP را به همان روشی که ترافیک را به پراکسی‌ها هدایت می‌کنید به Gateway ها هدایت کنید. معمولاً، شما صراحتاً مرورگرها را برای استفاده از Gateway ها، رهگیری ترافیک شفاف، یا پیکربندی Gateway ها به عنوان جانشین (پراکسی‌های معکوس) پیکربندی می‌کنید.

شکل زیر کادرهای محاوره ای مورد استفاده برای پیکربندی مرورگر برای استفاده از Gateway های FTP سمت سرور را نشان می‌دهد. در پیکربندی نشان داده شده، مرورگر برای استفاده از gw1.joeshardware.com به عنوان دروازه HTTP/FTP برای همه URL های FTP پیکربندی شده است. مرورگر به جای ارسال دستورات FTP به سرور FTP، دستورات HTTP را به دروازه HTTP/FTP gw1.joes-hardware.com در پورت ۸۰۸۰ ارسال می‌کند.



نتیجه این پیکربندی Gateway در شکل زیر نشان داده شده است.



ترافیک HTTP معمولی تحت تأثیر قرار نمی‌گیرد و به طور مستقیم به سرورهای مبدا جریان می‌یابد. اما درخواست‌ها برای URL های FTP به Gateway مربوط به gw1.joes-hardware.com در درخواست‌های HTTP ارسال می‌شوند. Gateway، تراکنش‌های FTP را از طرف کلاینت انجام می‌دهد و نتایج را توسط HTTP به کلاینت باز می‌گرداند.

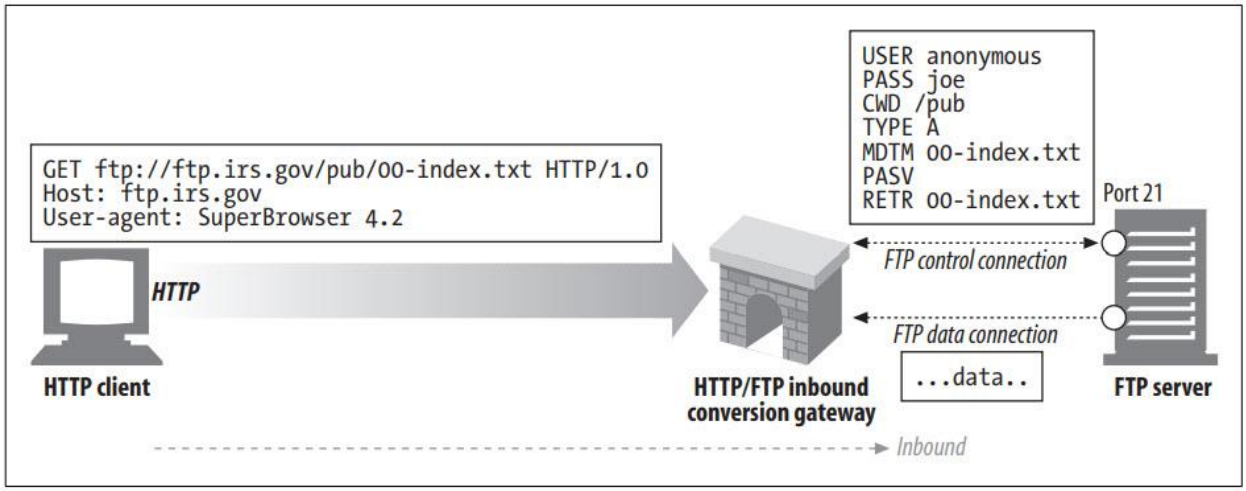
در ادامه به انواع رایج Gateway ها را می‌پردازیم.

HTTP/*: Server-Side Web Gateways

Server-Side Web Gateways، درخواست‌های HTTP سمت کلاینت را به یک پروتکل خارجی تبدیل می‌کنند، زیرا درخواست‌ها به سمت سرور مبدا وارد می‌شوند.

در شکل زیر، Gateway یک درخواست HTTP برای یک منبع FTP دریافت می‌کند:

ftp://ftp.irs.gov/pub/00-index.txt



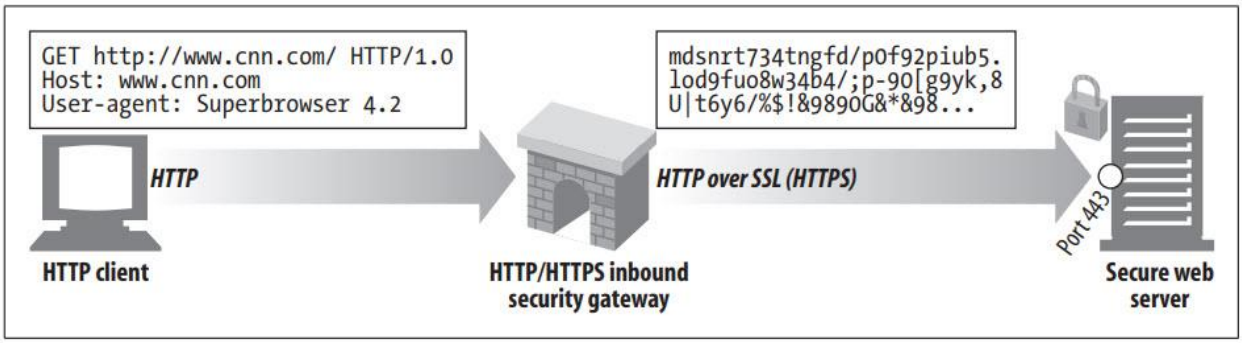
Gateway شروع به باز کردن اتصال FTP به پورت FTP در سرور مبدا (پورت ۲۱) می‌کند و با پروتکل FTP برای واکشی شی صحبت می‌کند. Gateway کارهای زیر را انجام می‌دهد:

- دستورات USER و PASS را برای ورود به سرور ارسال می‌کند.
- دستور CWD را برای تغییر به دایرکتوری مناسب روی سرور صادر می‌کند.
- نوع دانلود را روی ASCII تنظیم می‌کند.
- آخرین زمان اصلاح سند را با MDTM واکشی می‌کند.
- به سرور می‌گوید که با استفاده از PASV انتظار بازیابی اطلاعات غیرفعال را داشته باشد.
- درخواست بازیابی شی با استفاده از RETR
- اتصال داده‌ای را به سرور FTP در پورت بازگردانده شده در کانال کنترل باز می‌کند. به محض باز شدن کانال داده، محتوای شیء به Gateway باز می‌گردد.

هنگامی که بازیابی کامل شد، شی در یک پاسخ HTTP برای کلاینت ارسال می‌شود.

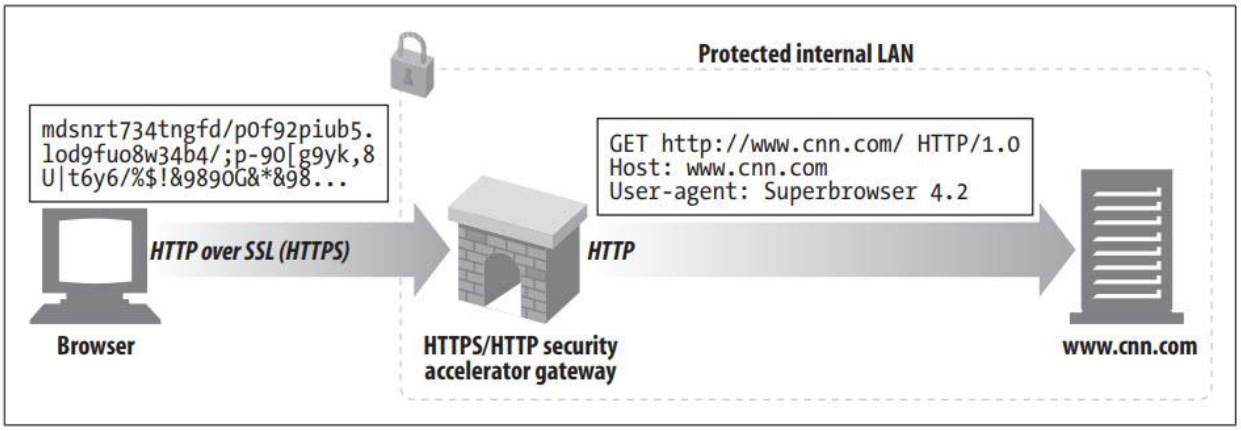
HTTP/HTTPS: Server-Side Security Gateways

از Gateway ها می‌توان برای ایجاد حریم خصوصی و امنیت بیشتر برای یک سازمان، با رمزگذاری تمام درخواست‌های وب ورودی استفاده کرد. کلاینت‌ها می‌توانند با استفاده از HTTP معمولی، وب را مرور کنند، اما Gateway به طور خودکار جلسات کاربر را رمزگذاری می‌کند (شکل زیر).



HTTPS/HTTP: Client-Side Security Accelerator Gateways

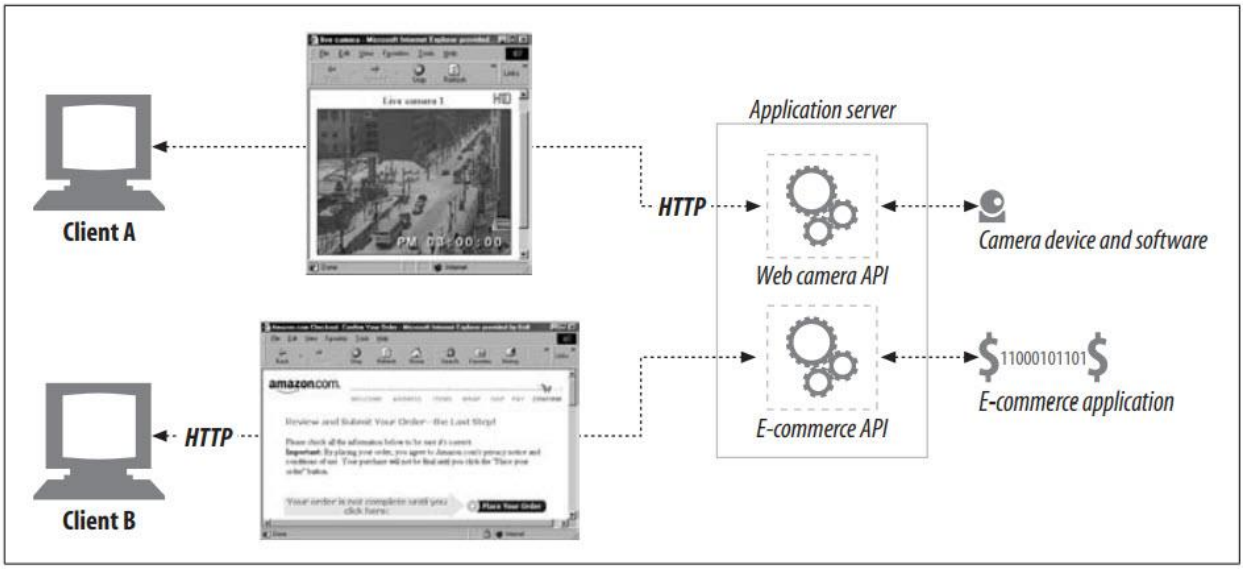
اخیرا Gatewayهای HTTPS/HTTP به عنوان شتاب دهنده‌های امنیتی محبوب شده‌اند. این Gatewayهای HTTPS/HTTP که در مقابل سرور وب قرار می‌گیرند، معمولاً به عنوان یک Gateway رهگیری نامرئی یا یک پروکسی معکوس عمل می‌کنند. آن‌ها ترافیک امن HTTPS را دریافت می‌کنند، ترافیک امن را رمزگشایی می‌کنند و درخواست‌های HTTP معمولی را به سرور وب می‌دهند (شکل زیر).



این Gatewayها اغلب شامل سخت‌افزار رمزگشایی ویژه برای رمزگشایی ترافیک امن بسیار کارآمدتر از سرور مبدا هستند و بار را از سرور مبدا حذف می‌کنند. از آنجا که این Gatewayها ترافیک رمزگذاری نشده را بین Gateway و سرور مبدا ارسال می‌کنند، باید احتیاط کنید تا مطمئن شوید شبکه بین Gateway و سرور مبدا ایمن است.

Resource Gateways

تا کنون، ما در مورد Gatewayهایی صحبت کرده‌ایم که کلاینت‌ها و سرورها را در یک شبکه به هم متصل می‌کنند. با این حال، رایج ترین شکل Gateway، سرور برنامه، سرور مقصد و Gateway را در یک سرور واحد ترکیب می‌کند. سرورهای برنامه Gatewayهای سمت سرور هستند که با سرویس گیرنده HTTP صحبت می‌کنند و به یک برنامه کاربردی در سمت سرور متصل می‌شوند (شکل زیر را ببینید).



در شکل بالا، دو کلاینت با استفاده از HTTP به یک سرور برنامه متصل می‌شوند. اما، به جای ارسال فایل‌ها از سرور، سرور برنامه درخواست‌ها را از طریق یک رابط برنامه‌نویسی برنامه‌نویسی دروازه (API) به برنامه‌های در حال اجرا روی سرور ارسال می‌کند:

- درخواست کلاینت A دریافت شده و بر اساس URI از طریق یک API به یک برنامه دوربین دیجیتال ارسال می‌شود. تصویر دوربین به‌دست‌آمده در یک پیام پاسخ HTTP قرار می‌گیرد و برای نمایش در مرورگر کلاینت به وی ارسال می‌شود.
- URI کلاینت B برای یک برنامه تجارت الکترونیکی است. درخواست‌های کلاینت B از طریق API دروازه سرور به نرم افزار تجارت الکترونیک ارسال می‌شود و نتایج به مرورگر بازگردانده می‌شود. نرم افزار تجارت الکترونیک با کلاینت تعامل دارد و کاربر را از طریق دنباله ای از صفحات HTML برای تکمیل خرید راهنمایی می‌کند.

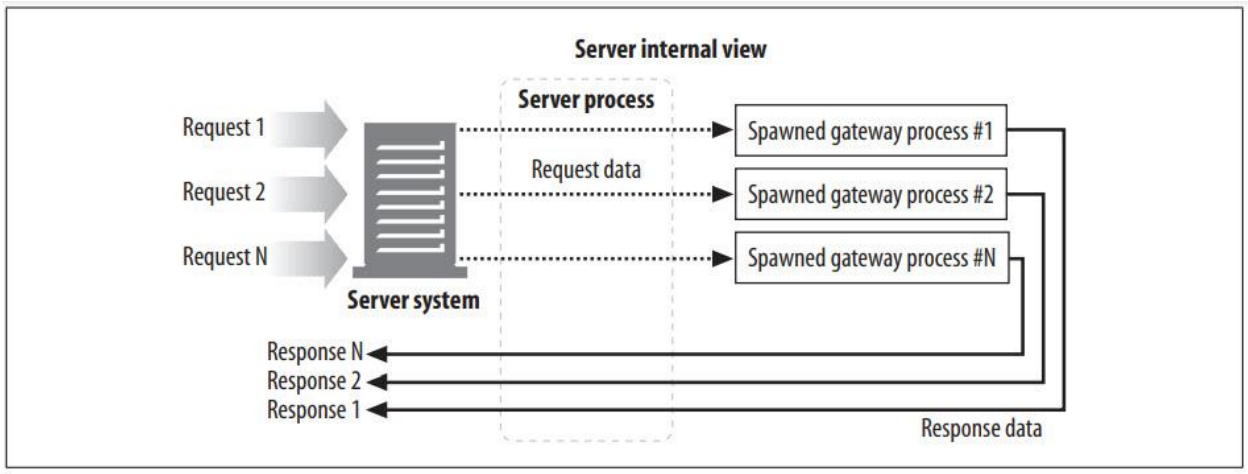
اولین API محبوب برای Application Gateway ها، Common Gateway Interface یا CGI بود. CGI مجموعه استاندارد شده‌ای از اینترفیس‌ها است که سرورهای وب برای راه اندازی برنامه‌ها در

پاسخ به درخواست‌های HTTP برای URL های خاص، جمع آوری خروجی برنامه و ارسال آن در پاسخ‌های HTTP استفاده می‌کنند. در چند سال گذشته، وب سرورهای تجاری، اینترفیس‌های پیچیده‌تری را برای اتصال سرورهای وب به برنامه‌ها ارائه کرده اند.

وب سرورهای اولیه ساخته‌های نسبتاً ساده ای بودند و رویکرد ساده ای که برای پیاده سازی یک اینترفیس برای Gateway ها در نظر گرفته شد تا به امروز باقی مانده است.

هنگامی که یک درخواست برای منبعی وارد می‌شود که به یک Gateway نیاز دارد، سرور برنامه کمکی را برای رسیدگی به درخواست ارسال می‌کند. برنامه کمکی داده‌های مورد نیاز خود را ارسال می‌کند. اغلب این فقط کل درخواست یا چیزی شبیه پرس و جوی است که کاربر می‌خواهد در پایگاه داده اجرا کند.

سپس یک پاسخ یا داده پاسخ را به سرور برمی‌گرداند که آن را به کلاینت منتقل می‌کند. سرور و Gateway برنامه‌های جداگانه ای هستند، بنابراین خطوط مسئولیت روشن نگه داشته می‌شوند. شکل زیر مکانیزم اساسی در پشت تعاملات برنامه کاربردی سرور و Gateway را نشان می‌دهد.



این پروتکل ساده (درخواست وارد کردن، تحویل دادن و پاسخ دادن) جوهره پشت قدیمی‌ترین و یکی از رایج‌ترین اینترفیس‌ها یعنی CGI است.

Common Gateway Interface (CGI)

Common Gateway Interface اولین و احتمالا هنوز هم پرکاربردترین افزونه سرور است و در سرتاسر وب برای مواردی مانند HTML پویا، پردازش کارت اعتباری و جستجو در پایگاه داده استفاده می‌شود.

از آنجایی که برنامه‌های CGI جدا از سرور هستند، تقریباً می‌توان آن‌ها را در هر زبانی از جمله Perl، Tcl، C و زبان‌های مختلف Shell پیاده سازی کرد. از آنجایی که CGI ساده است، تقریباً تمام سرورهای HTTP از آن پشتیبانی می‌کنند. مکانیزم اصلی مدل CGI در شکل بالا نشان داده شده است.

پردازش CGI برای کاربران نامرئی است. از دیدگاه کلاینت، این فقط یک درخواست عادی بوده و وی کاملاً از روند دستیابی بین سرور و برنامه CGI بی اطلاع است. تنها اشاره ای در کلاینت که نشان دهنده استفاده یک برنامه از CGI است، وجود حروف "cgi" و شاید "؟" در URL است.

بنابراین CGI فوق العاده است، درست است؟ خوب، بله و نه. این یک شکل ساده و کاربردی از چسب بین سرورها و تقریباً هر نوع منبعی را فراهم نموده و هر ترجمه‌ای را که نیاز به انجام دارد مدیریت می‌کند. این رابط همچنین برای محافظت از سرور در برابر برنامه‌های افزودنی دارای باگ ظریف است (اگر برنامه افزودنی روی خود سرور قرار گیرد، ممکن است باعث خطا شود که ممکن است سرور را از کار بیندازد).

با این حال، این جداسازی هزینه در عملکرد دارد. هزینه سربار ایجاد یک فرآیند جدید برای هر درخواست CGI بسیار زیاد است و عملکرد سرورهایی را که از CGI استفاده می‌کنند محدود می‌کند و منابع ماشین سرور را هدر می‌دهد. برای تلاش برای دور زدن این مشکل، شکل جدیدی از CGI - که به درستی Fast CGI نامیده می‌شود - ایجاد شده است. این رابط از CGI تقلید می‌کند، اما به‌عنوان یک شبح پایدار اجرا می‌شود و جریمه‌ی عملکردی را برای راه‌اندازی و از بین بردن یک فرآیند جدید برای هر درخواست حذف می‌کند.

Server Extension APIs

پروتکل CGI روشی تمیز برای اتصال مفسرهای خارجی با سرورهای HTTP موجود ارائه می‌کند، اما اگر بخواهید رفتار خود سرور را تغییر دهید، یا فقط می‌خواهید آخرین قطره عملکردی را که می‌توانید از سرور خود خارج کنید، چه باید کرد؟ برای این دو نیاز، توسعه دهندگان Server Extension APIs را ارائه کرده اند که یک رابط قدرتمند برای توسعه دهندگان وب فراهم می‌کند تا ماژول‌های خود را مستقیماً با یک سرور HTTP ارتباط برقرار کنند. Extension APIs به برنامه نویسان اجازه می‌دهند تا کد خود را به سرور پیوند بزنند یا به طور کامل یک جزء از سرور را تعویض کرده و آن را با کد خود جایگزین کنند.

اکثر سرورهای محبوب یک یا چند API افزونه را برای توسعه دهندگان ارائه می‌کنند. از آنجایی که این پسوندها اغلب با معماری خود سرور مرتبط هستند، اکثر آن‌ها مختص یک نوع سرور هستند. مایکروسافت، نت اسکیپ، آپاچی و سایر سرورها همگی دارای رابطهای API هستند که به توسعه دهندگان اجازه می‌دهد رفتار

سرور را تغییر دهند یا رابط‌های سفارشی را برای منابع مختلف ارائه دهند. این رابط‌های سفارشی یک رابط قدرتمند برای توسعه دهندگان فراهم می‌کند.

یکی از نمونه‌های Server Extension، افزونه فرانت پیج سرور مایکروسافت (FPSE) است که از خدمات انتشار وب برای نویسندگان فرانت پیج پشتیبانی می‌کند. FPSE قادر است دستورات Remote Procedure Call یا RPC ارسال شده توسط کلاینت فرانت پیج را تفسیر کند. این دستورات بر روی HTTP (به طور خاص، روی متد POST همپوشانی دارند).

Application Interfaces and Web Services

ما Gateway های منابع را به عنوان راه‌هایی برای وب سرورها برای برقراری ارتباط با برنامه‌ها مورد بحث قرار داده‌ایم. به طور کلی‌تر، با برنامه‌های کاربردی وب که انواع بیشتری از خدمات را ارائه می‌دهند، مشخص می‌شود که HTTP می‌تواند بخشی پایه‌ای برای پیوند دادن برنامه‌ها به یکدیگر باشد. یکی از مسائل پیچیده‌تر در سیم‌کشی برنامه‌ها، مذاکره روی رابط پروتکل بین دو برنامه است تا بتوانند داده‌ها را مبادله کنند - اغلب این کار بر اساس برنامه به برنامه انجام می‌شود.

برای کار با هم، برنامه‌ها معمولاً نیاز به تبادل اطلاعات پیچیده‌تر با یکدیگر دارند که در هدرهای HTTP قابل بیان است. چند نمونه از گسترش پروتکل‌های HTTP یا لایه بندی در بالای HTTP به منظور تبادل اطلاعات سفارشی شده در فصل ۱۹ توضیح داده خواهد شد.

جامعه اینترنتی مجموعه‌ای از استانداردها و پروتکل‌ها را توسعه داده است که به برنامه‌های کاربردی وب اجازه می‌دهد با یکدیگر صحبت کنند. این استانداردها به‌طور ساده تحت عنوان Web Service شناخته می‌شوند، اگرچه این اصطلاح می‌تواند به معنای خود برنامه‌های کاربردی وب (ساختمان) مستقل باشد. پیش فرض سرویس‌های وب جدید نیست، اما مکانیسم جدیدی برای برنامه‌های کاربردی جهت به اشتراک گذاری اطلاعات است. وب سرویس‌ها بر اساس فناوری‌های استاندارد وب مانند HTTP ساخته شده‌اند.

سرویس‌های وب با استفاده از XML از طریق SOAP اطلاعات را مبادله می‌کنند. زبان نشانه گذاری توسعه پذیر (XML) راهی برای ایجاد و تفسیر اطلاعات سفارشی شده در مورد یک شی داده فراهم می‌کند. پروتکل دسترسی ساده به اشیا (SOAP) استانداردی برای افزودن اطلاعات XML به پیام‌های HTTP است.

جهت کسب اطلاعات بیشتر، به http://www.w3.org/TR/2001/WD-soap12-part0-20011217/ مراجعه کنید. برنامه نویسی وب سرویس با SOAP، توسط داگ تیدول، جیمز اسنل، و پاول کولچنکو (O'Reilly) نیز منبع عالی اطلاعات در مورد پروتکل SOAP است.

Tunnels

ما روش‌های مختلفی را مورد بحث قرار داده‌ایم که از HTTP می‌توان برای فعال کردن دسترسی به انواع منابع (از طریق Gateway ها) و فعال کردن ارتباط برنامه به برنامه استفاده کرد. در این بخش، نگاهی به کاربرد دیگری از HTTP خواهیم داشت، تونل‌های وب، دسترسی به برنامه‌هایی که از پروتکل‌های غیر HTTP استفاده می‌کنند را از طریق برنامه‌های HTTP امکان‌پذیر می‌سازد.

تونل‌های وب به شما امکان می‌دهند ترافیک غیر HTTP را از طریق اتصالات HTTP ارسال کنید و به پروتکل‌های دیگر اجازه می‌دهد در بالای HTTP به عقب بروند. رایج‌ترین دلیل استفاده از تونل‌های وب، تعبیه ترافیک غیر HTTP در یک اتصال HTTP است، بنابراین می‌توان آن را از طریق فایروال‌هایی ارسال کرد که فقط به ترافیک وب اجازه وارد شدن می‌دهند.

Establishing HTTP Tunnels with CONNECT

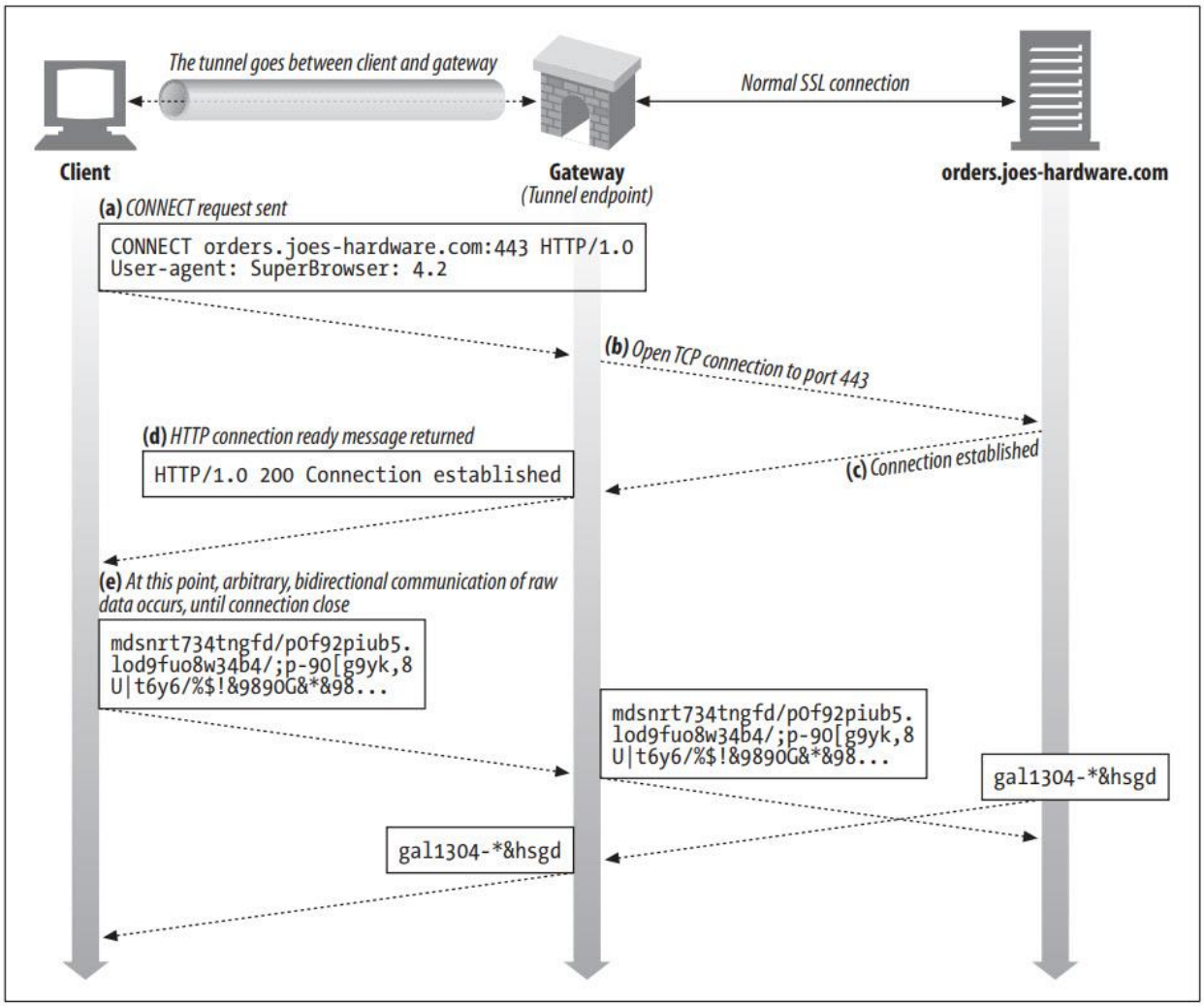
تونل‌های وب با استفاده از روش HTTP's CONNECT ایجاد می‌شوند. پروتکل CONNECT بخشی از مشخصات اصلی HTTP/1.1 نیست، اما یک برنامه افزودنی است که به طور گسترده پیاده سازی شده است. مشخصات فنی را می‌توان در مشخصات پیش‌نویس اینترنت منقضی شده آری لوتونن، «تونل‌سازی پروتکل‌های مبتنی بر TCP از طریق سرورهای پروکسی وب» یا در کتاب سرورهای پروکسی وب او یافت که هر دو در پایان این فصل ذکر شده‌اند.

متد CONNECT از یک Tunnel Gateway می‌خواهد که یک اتصال TCP به سرور و پورت مقصد دلخواه ایجاد کند و داده‌های بعدی را کورکورانه بین کلاینت و سرور ارسال نماید.

شکل زیر نحوه عملکرد متد CONNECT برای ایجاد یک تونل به یک دروازه را Gateway می‌دهد:

- در بخش a شکل، کلاینت یک درخواست CONNECT را به Tunnel Gateway ارسال می‌کند. متد CONNECT کلاینت از Tunnel Gateway می‌خواهد که یک اتصال TCP را باز کند (در اینجا، به میزبانی به نام orders.joes-hardware.com در پورت ۴۴۳، پورت پیش فرض SSL).
- اتصال TCP در بخش b و بخش c شکل ایجاد شده است.
- هنگامی که اتصال TCP برقرار شد، دروازه با ارسال یک پاسخ HTTP 200 Connection Established به کلاینت (بخش d شکل) اطلاع می‌دهد.

- در این مرحله، تونل راه اندازی می‌شود. هر داده‌ای که توسط کلاینت از طریق تونل HTTP ارسال می‌شود، مستقیماً به اتصال TCP خروجی رله می‌شود و هر داده‌ای که توسط سرور ارسال می‌شود از طریق تونل HTTP به کلاینت منتقل می‌گردد.



مثال در موجود در شکل بالا یک تونل SSL را توصیف می‌کند که در آن ترافیک SSL از طریق یک اتصال HTTP ارسال می‌شود، اما روش CONNECT می‌تواند برای ایجاد یک اتصال TCP به هر سرور با استفاده از هر پروتکل استفاده شود.

CONNECT requests

ساختار CONNECT از نظر شکل با سایر متدهای HTTP یکسان است، به استثنای خط شروع. URI درخواست با یک نام میزبان و به دنبال آن یک دونقطه و به دنبال آن یک شماره پورت جایگزین می‌شود. هر دو میزبان و پورت باید مشخص شوند:

CONNECT home.netscape.com:443 HTTP/1.0

User-agent: Mozilla/4.0

بعد از خط شروع، مانند سایر پیام‌های HTTP، فیلدهای هدر درخواست HTTP صفر یا بیشتر وجود دارد. طبق معمول، خطوط به CRLF ختم می‌شوند و فهرست هدرها با CRLF خالی به پایان می‌رسد.

CONNECT responses

پس از ارسال درخواست، کلاینت منتظر پاسخ از Gateway می‌شود. همانند پیام‌های HTTP معمولی، کد پاسخ 200 نشان دهنده موفقیت است. طبق قرارداد، Reason Pharse یا عبارت دلیل در پاسخ معمولاً روی "Connection Established" تنظیم می‌شود:

HTTP/1.0 200 Connection Established

Proxy-agent: Netscape-Proxy/1.1

برخلاف پاسخ‌های معمولی HTTP، پاسخ نیازی به هدر Content-Type نداشته و هیچ نوع محتوایی مورد نیاز نیست. زیرا اتصال به جای حامل پیام به یک رله بایت خام تبدیل می‌شود.

Data Tunneling, Timing, and Connection Management

از آنجایی که داده‌های تونل شده نسبت به Gateway مات هستند(شفاف نیستند)، Gateway نمی‌تواند هیچ فرضی در مورد ترتیب و جریان بسته‌ها داشته باشد. پس از ایجاد تونل، داده‌ها در هر زمانی در هر جهتی جریان دارند.

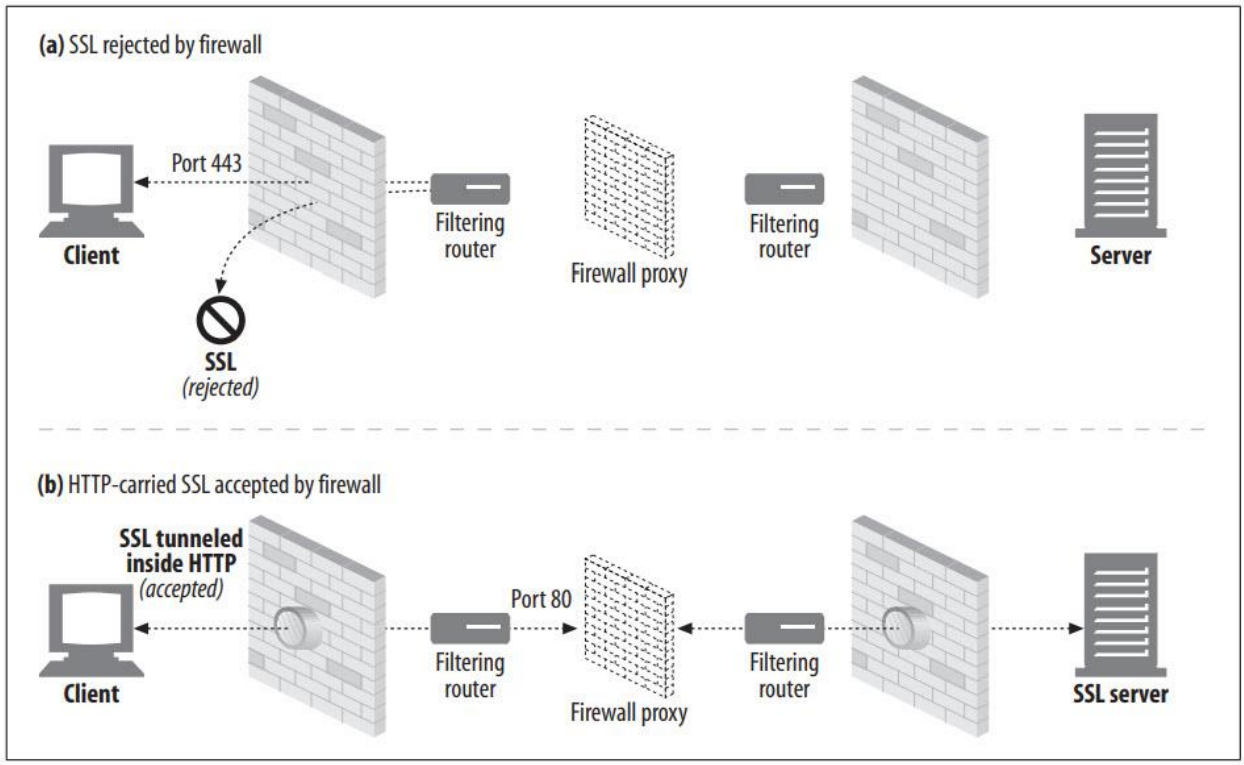
به عنوان یک بهینه سازی عملکرد، کلاینت‌ها اجازه دارند پس از ارسال درخواست CONNECT، اما قبل از دریافت پاسخ، داده‌های تونل را ارسال کنند. این داده‌ها را سریع‌تر به سرور می‌رساند، اما به این معنی است که Gateway باید بتواند داده‌های پس از درخواست را به درستی مدیریت کند. به طور خاص، Gateway نمی‌تواند فرض کند که درخواست ورودی/خروجی شبکه فقط داده‌های هدر را برمی‌گرداند و Gateway باید مطمئن باشد که هر داده ای را که با هدر خوانده شده است، زمانی که اتصال آماده است، به سرور ارسال می‌کند.

اگر پاسخ به عنوان چالش احراز هویت یا سایر وضعیت‌های غیر 200 و nonfatal بازگردد، کلاینت‌هایی که داده‌ها را پس از درخواست ارسال می‌کنند، باید آماده باشند تا داده‌های درخواست را مجدداً ارسال کنند.

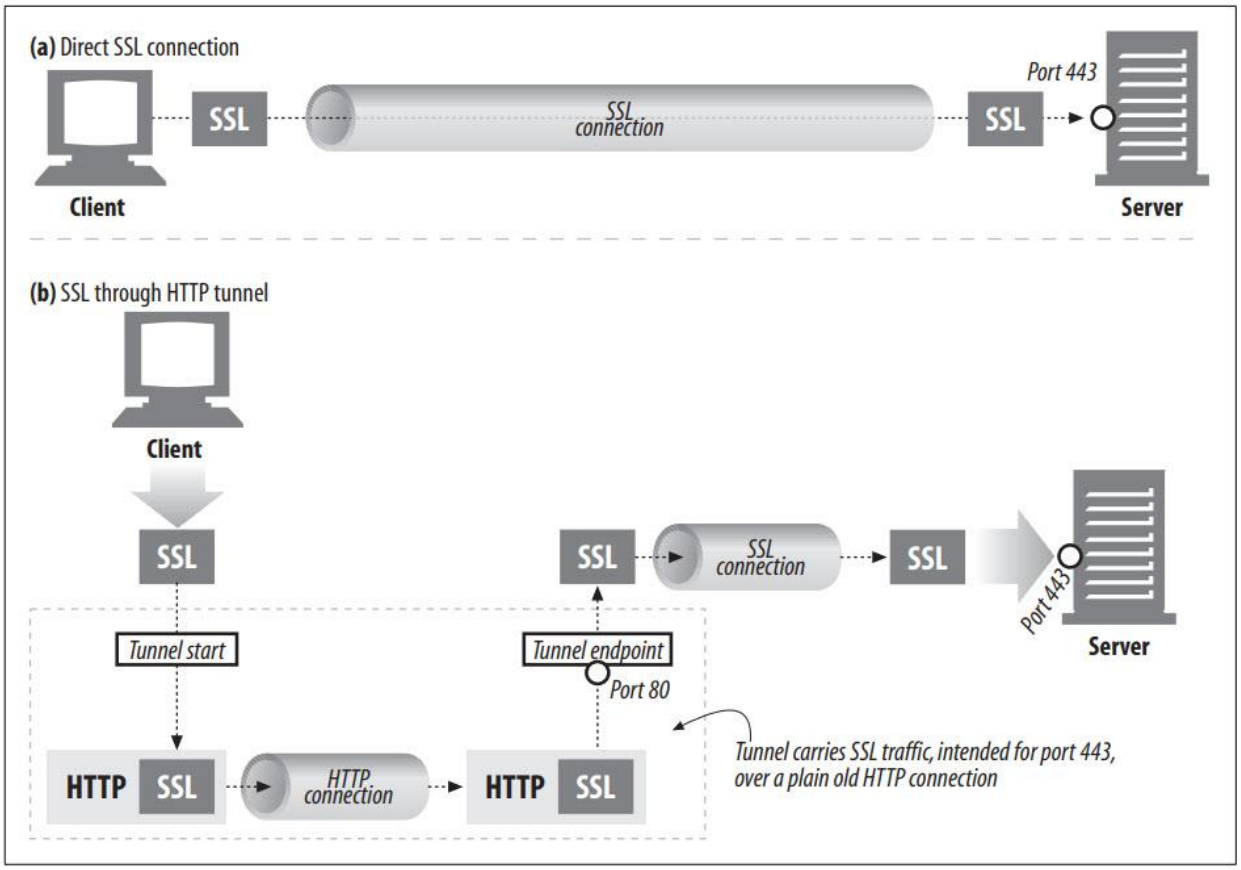
اگر در هر نقطه از یک Endpoint تونل قطع شود، هر داده باقی مانده‌ای که از آن Endpoint آمده است به دیگری منتقل شده و پس از آن نیز اتصال دیگر توسط پراکسی قطع می‌شود. اگر داده‌های تحویل‌نشده برای Endpoint بسته وجود داشته باشد، آن داده‌ها کنار گذاشته می‌شوند.

SSL Tunneling

تونل‌های وب ابتدا برای انتقال ترافیک رمزگذاری شده SSL از طریق فایروال‌ها توسعه یافتند. بسیاری از سازمان‌ها تمام ترافیک را از طریق روترهای Packet Filter و سرورهای پروکسی برای افزایش امنیت انتقال می‌دهند. اما برخی از پروتکل‌ها، مانند SSL رمزگذاری شده، توسط سرورهای پروکسی سنتی قابل پروکسی نیستند، زیرا اطلاعات رمزگذاری شده است. تونل‌ها به ترافیک SSL اجازه می‌دهند از طریق فایروال HTTP پورت ۸۰ با انتقال آن از طریق یک اتصال HTTP حمل شود.



برای اجازه دادن به ترافیک SSL از طریق فایروال‌های پراکسی موجود، یک ویژگی تونل سازی به HTTP اضافه شد که در آن داده‌های خام و رمزگذاری شده در پیام‌های HTTP قرار می‌گیرند و از طریق کانال‌های HTTP معمولی ارسال می‌شوند.



در بخش a از شکل بالا، ترافیک SSL مستقیماً به یک وب سرور امن (در پورت 443 SSL) ارسال می‌شود. در بخش b، ترافیک SSL در پیام‌های HTTP کپسوله شده و از طریق اتصالات پورت 80 HTTP ارسال می‌شود، تا زمانی که دوباره به اتصالات SSL معمولی کپسوله شود.

در تونل‌ها اغلب برای عبور ترافیک غیر HTTP، از فایروال‌های فیلتر کننده پورت استفاده می‌شود. از این موضوع می‌توان به خوبی استفاده کرد، به عنوان مثال، اجازه می‌دهد تا ترافیک امن SSL از طریق فایروال‌ها جریان یابد. با این حال، این ویژگی می‌تواند مورد سوء استفاده قرار گیرد و به پروتکل‌های مخرب اجازه می‌دهد تا از طریق تونل HTTP به یک سازمان سرازیر شوند.

SSL Tunneling Versus HTTP/HTTPS Gateways

پروتکل HTTPS (HTTP روی SSL) می‌تواند به روشی مشابه پروتکل‌های دیگر دروازه‌سازی شود: داشتن Gateway (به جای کلاینت) جلسه SSL را با سرور HTTPS راه دور آغاز می‌کند و سپس تراکنش HTTPS را در قسمت کلاینت انجام می‌دهد. پاسخ توسط پراکسی دریافت و رمزگشایی می‌شود و از طریق HTTP (ناامن) برای کلاینت ارسال می‌شود. این راهی است که Gateway ها با FTP مدیریت می‌کنند.

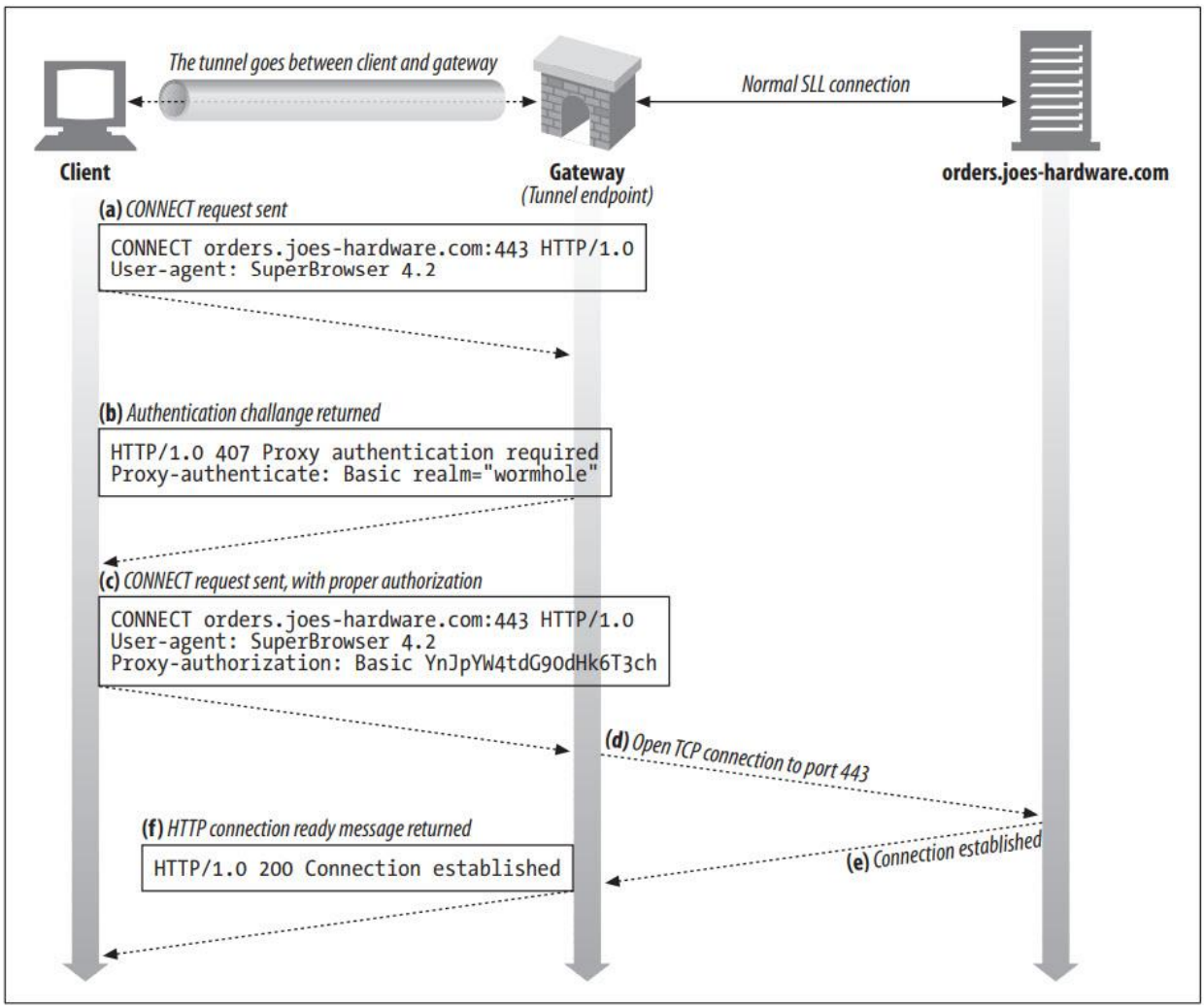
با این حال، این روش دارای چندین معایب است:

- اتصال کلاینت به دروازه HTTP عادی و ناامن است.
- کلاینت نمی‌تواند احراز هویت کلاینت SSL (تأیید هویت بر اساس گواهی‌های X509) را در سرور راه دور انجام دهد، زیرا پراکسی طرف احراز هویت است.
- دروازه باید از اجرای کامل SSL پشتیبانی کند.

توجه داشته باشید که این مکانیسم، اگر برای تونل سازی SSL استفاده شود، نیازی به پیاده سازی SSL در پروکسی ندارد. جلسه SSL بین • کلاینت ایجاد کننده درخواست و وب سرور مقصد (ایمن) برقرار می‌شود. سرور پروکسی در این میان فقط داده‌های رمزگذاری شده را تونل می‌کند و هیچ بخش دیگری در تراکنش امن نمی‌گیرد.

Tunnel Authentication

ویژگی‌های دیگر HTTP نیز می‌تواند در صورت لزوم با تونل‌ها استفاده شود. به طور خاص، پشتیبانی از احراز هویت پراکسی می‌تواند با تونل‌ها برای احراز هویت حق استفاده از یک تونل استفاده شود. (client’s right to use a tunnel)



Tunnel Security Considerations

به طور کلی، Tunnel Gateway نمی‌تواند تأیید کند که پروتکلی که صحبت می‌شود واقعاً همان چیزی است که قرار است تونل کند یا خیر. بنابراین، برای مثال، کاربران مخرب ممکن است از تونل‌های در نظر گرفته شده برای SSL برای تونل کردن ترافیک بازی‌های اینترنتی از طریق فایروال شرکتی استفاده کنند، یا کاربران مخرب ممکن است از تونل‌ها برای باز کردن جلسات Telnet یا ارسال ایمیلی استفاده کنند که اسکنرهای ایمیل شرکتی را دور می‌زند.

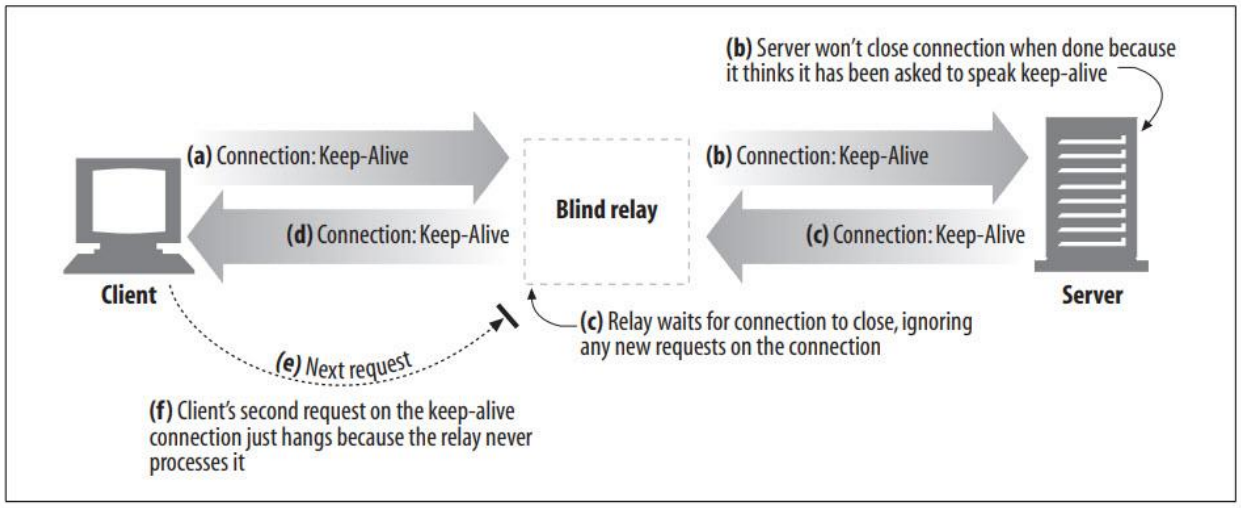
برای به حداقل رساندن سوء استفاده از تونل‌ها، Gateway باید تونل‌ها را فقط برای پورت‌های شناخته شده خاص، مانند ۴۴۳ برای HTTPS باز کند.

Relays

رله‌های HTTP پروکسی‌های ساده HTTP هستند که به طور کامل به مشخصات HTTP پایبند نیستند. رله‌ها HTTP کافی را برای ایجاد اتصالات پردازش می‌کنند، سپس کورکورانه بایت‌ها را به جلو ارسال می‌کنند.

از آنجایی که HTTP پیچیده است، گاهی اوقات پیاده‌سازی پروکسی‌های بی‌سابقه‌ای که فقط به‌طور کورکورانه ترافیک را جلو می‌برند، بدون انجام تمام منطق هدر و متد، مفید است. از آنجایی که رله‌های کور به راحتی قابل پیاده‌سازی هستند، گاهی اوقات از آن‌ها برای ارائه فیلتر ساده، تشخیص یا تغییر محتوا استفاده می‌شود. اما به دلیل پتانسیل جدی برای مشکلاتی که ممکن است استفاده از آن‌ها به همراه داشته باشد، باید با احتیاط فراوان به کار گرفته شوند.

یکی از مشکلات رایج (و بدنام) در برخی از پیاده‌سازی‌های رله‌های کور ساده، به پتانسیل آن‌ها برای قطع keep-Live مربوط می‌شود، زیرا هدر Connection را به درستی پردازش نمی‌کنند. این وضعیت در شکل زیر نشان داده شده است.



این چیزی است که در این شکل اتفاق می‌افتد:

در بخش a از شکل بالا، یک سرویس گیرنده وب پیامی را به رله ارسال می‌کند، از جمله هدر Connection: Keep-Alive، و در صورت امکان درخواست اتصال keep-alive می‌کند. کلاینت منتظر پاسخ می‌ماند تا بداند آیا درخواستش برای کانال keep-alive پذیرفته شده است یا خیر.

رله، درخواست HTTP را دریافت می‌کند، اما هدر Connection را نمی‌فهمد، بنابراین پیام را کلمه به کلمه از زنجیره به سرور ارسال می‌کند (بخش b). با این حال، هدر Connection یک هدر hop-by-

hop است و فقط برای یک لینک حمل و نقل اعمال می‌شود و نباید به زنجیره منتقل شود. اتفاقات بد در شرف شروع شدن است!

در بخش b از شکل بالا، درخواست HTTP رله شده به وب سرور می‌رسد. هنگامی که وب سرور هدر Connection: Keep-Alive را دریافت می‌کند، به اشتباه نتیجه می‌گیرد که رله (که مانند هر کلاینت دیگری برای سرور به نظر می‌رسد) می‌خواهد با keep-alive صحبت کند! این برای وب سرور خوب است – می‌پذیرد که به طور keep-alive صحبت کند و یک هدر پاسخ Connection: Keep-Alive را (در بخش c) ارسال می‌کند. بنابراین، در این مرحله، وب سرور تصور می‌کند که با رله در حال صحبت با keep-alive است و به قوانین keep-alive پایبند است. اما رله چیزی در مورد keep-alive نمی‌داند.

در بخش d، رله پیام پاسخ سرور وب را به سمت کلاینت ارسال می‌کند و از هدر Connection: Keep-Alive از وب سرور عبور می‌کند. کلاینت این هدر را می‌بیند و فرض می‌کند که رله پذیرفته است که keep-alive بماند. در این مرحله، هم کلاینت و هم سرور بر این باورند که دارند keep-alive صحبت می‌کنند، اما رله‌ای که با آن صحبت می‌کنند، موارد اولیه مربوط به keep-alive را نمی‌داند.

از آنجایی که رله چیزی در مورد keepalive نمی‌داند، تمام داده‌هایی را که دریافت می‌کند به کلاینت ارسال می‌کند و منتظر می‌ماند تا سرور مبدا اتصال را ببندد. اما سرور مبدا اتصال را نمی بندد، زیرا معتقد است رله از سرور خواسته است که اتصال را باز نگه دارد! بنابراین، رله منتظر بسته شدن اتصال خواهد ماند.

هنگامی که کلاینت پیام پاسخ را در (بخش d) دریافت می‌کند، مستقیماً به درخواست بعدی حرکت نموده و درخواست دیگری را به رله در اتصال keep-alive ارسال می‌کند (بخش e). رله‌های ساده معمولاً هرگز انتظار درخواست دیگری در همان اتصال را ندارند. در این حالت مرورگر فقط می‌چرخد و هیچ پیشرفتی ندارد.

راه‌هایی برای هوشمندتر کردن رله‌ها برای حذف این خطرات وجود دارد، اما هر گونه ساده‌سازی پراکسی‌ها خطر مشکلات عملکردی را به همراه دارد. اگر رله‌های HTTP ساده را برای هدف خاصی می‌سازید، مراقب نحوه استفاده از آن‌ها باشید. برای هر استقرار در مقیاس وسیع، باید قویاً به جای آن از یک سرور پروکسی واقعی و سازگار با HTTP استفاده کنید.

For More Information

http://www.w3.org/Protocols/rfc2616/rfc2616.txt

Web Proxy Servers (Ari Luotonen, Prentice Hall Computer Books.)

http://www.alternic.org/drafts/drafts-l-m/draft-luotonen-web-proxy-tunneling-01.txt

http://cgi-spec.golux.com

http://www.w3.org/TR/2001/WD-soap12-part0-20011217/

Programming Web Services with SOAP (James Snell, Doug Tidwell, and Pavel Kulchenko, O'Reilly & Associates, Inc.)

http://www.w3.org/TR/2002/WD-wsa-reqs-20020429

Web Services Essentials (Ethan Cermai, O'Reilly & Associates, Inc.)