بسم الله الرحمن الرحیم

آشنایی با حافظه Stack
یا پشته

مقدمه :

نوشتن یک اکسپلویت و یا حتی پیدا کردن یک آسیب پذیری در نرم افزارهای گوناگون ، بسیار مورد توجه هکرها است.استفاده از روش های گوناگون برای یافتن یک آسیب پذیری یکی از اصلی ترین مواردی است که باید به آن پرداخت.اما در مورد چگونگی رخ دادن این نوع آسیب پذیری ها در حافظه کمتر بحث ککرده ایم.در این مقاله سعی شده است شما را با حافظه Stack که چندین مقاله مختلف در ارتباط با آن در اینترنت وجود دارد ، در سطح مقدماتی آشنا کنیم.

دسته بندی پردازش ها:

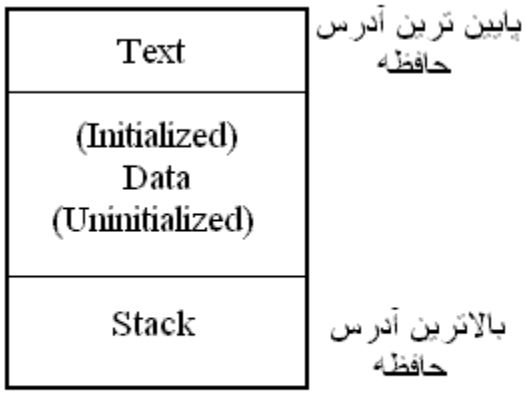
قبل از بحث در مورد حافظه ی Stack باید با ساختار کلی دسته بندی پردازش ها یا Process ها در حافظه آشنا شد.پردازش ها قسمتی از برنامه های مختلف هستند که وظیفه ی خاصی را انجام می دهند.پردازش ها به سه قسمت تقسیم می شوند:

1) نوشته ها یا Text
2) اطلاعات یا Data
3) حافظه ی Stack

ما در ادامه بر نحوه کار حافظه ی Stack به طور کامل تر آشنا خواهیم شد اما قبل از آن در مورد قسمت های دیگر که در بالا آمده اند ، توضیح خواهیم داد.

قسمت Text توسط برنامه و کدهای داخل آن ((آموزنده یا Instructions)) و داده های فقط خواندنی تنظیم می شود.همچنین این قسمت بخشی از ناحیه ی Text و نوشته ها در برنامه ی اجرایی است.این قسمت به طور معمول فقط خواندنی است و دسترسی به آن برای نوشتن با خطای منطقه ی غیر قابل دسترسی مواجه خواهد شد.

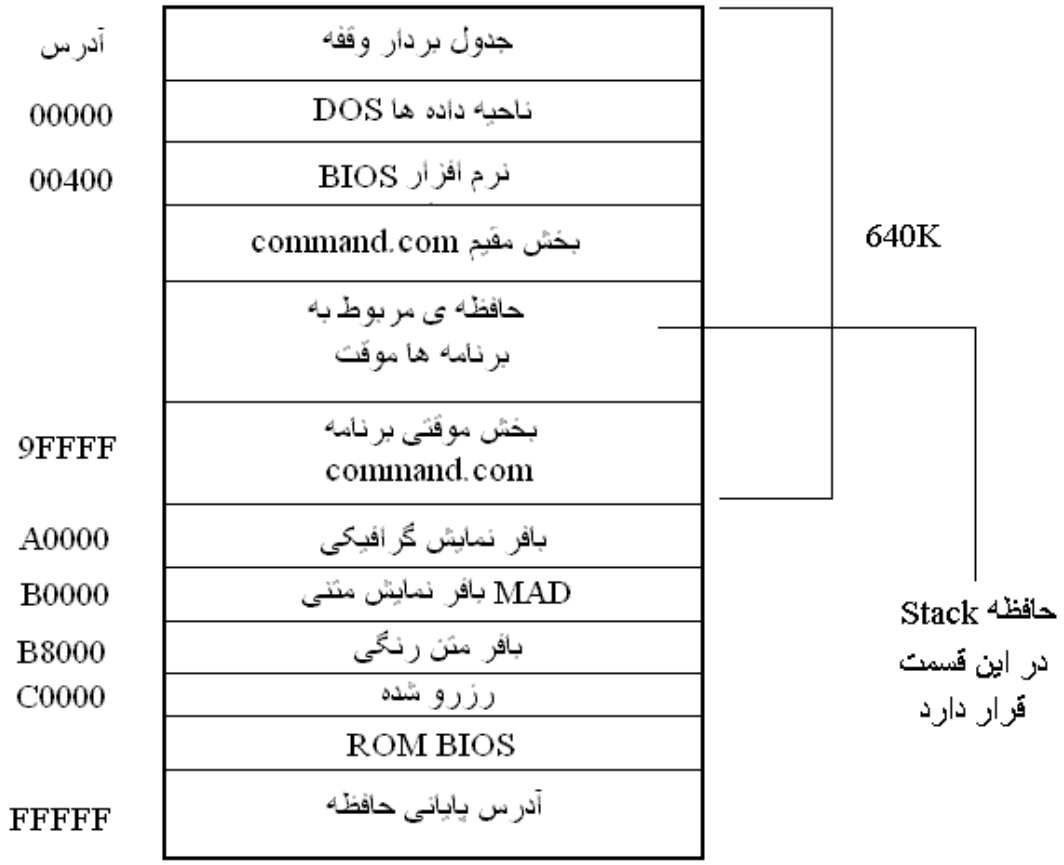
قسمت دیگر یعنی قسمت اطلاعات یا Data شامل اطلاعات شروع کننده((Initialized)) و خاتمه دهنده ((Uninitialized)) است.متغیرهای استاتیک در این قسمت نگه داری می شوند.این متغیرها در زمان بارگذاری برنامه مقداردهی می شوند.قسمت اطلاعات بخشی از قسمت Data-bss فایل اجرایی نیز می باشد.سایز این قسمت می تواند تغییر کند.اگر قسمتی از اطلاعات bss تخصیص یافته شده بیش از حد پر شود و یا Stack مورد استفاده نیز فضای خالی کمی داشته باشد ، در صورت فراخوانی پردازشی جدید ، این پردازش به طور موقت بسته می شود و مجددا برای اجرای دوباره با فضای حافظه ی بیشتر ، زمان بندی خواهد شد.فضای اختصاص یافته ی جدید بین قسمت های Stack و اطلاعات اضافه خواهد شد.



شکل ۱) قسمت های مختلف پردازش

آدرس حافظه :

حافظه ی سیستم برای انجام کارهای گوناگون آدرس بندی شده است که هر کدام از یک آدرس تا یک آدرس با یک اندازه مشخص و برای انجام کاری مشخص دسته بندی شده است.
به عنوان مثال آدرس حافظه ی RAM از 00000 تا B8000 و حافظه ی ROM نیز از آدرس C0000 تا آدرس FFFFF دسته بندی می شود.این اندازه ی یک مگابایت از حافظه است که شامل اطلاعات مختلفی است.این نوع آدرس بندی در سیستم های 32 بیتی است.بعضی از این قسمت ها قابل نوشتن و بعضی فقط قابل خواندن هستند.اطلاعات توسط سیستم عامل بر اساس نوع آن ها وارد قسمت خاص خودشان در این آدرس ها می شوند.در واقع سیستم عامل تنظیم می کند که اطلاعات در کجا و چگونه ذخیره شوند.بعضی برنامه ها نیز قسمت های مشخصی از آدرس های بالا را می گیرند تا اطلاعات و مقادیر متغیرهای موجود در برنامه را در این آدرس ها نگه داری کنند.به طور کل تمامی اطلاعات این آدرس ها اگر در هارد سیستم ذخیره نشوند ، پس از قطع برق از بین خواهند رفت.با پیدایش سری جدید پردازشگرهای Pentium و ویندوز XP سیستم ها توانستند تا 4 گیگابایت حافظه ی اصلی یا RAM را پشتیبانی کنند.در لینوکس و ویندوز این آدرس ها تفاوت زیادی ندارند و تقریبا یکسان هستند اما نحوه ی تقسیم بندی آن ها برای اطلاعات مختلف تفاوت دارد.هنگامی که تابعی فراخوانده می شود ، با توجه به این که تابع مقدار بازگشتی دارد یا نه ، آدرسی برای ذخیره ی مقدار برگشتی از تابع استفاده می شود که به این آدرس Return Address گویند. متغیرها و مقادیر بازگشتی و همچنین آدرس های بازگشتی در برنامه های مختلف برای ایجاد حملات Stack Overflow باید دستکاری شوند و مقدار متغیر را به یک مقدار نا معتبر که باعث ایجاد اختلال در برنامه می شود ، تغییر داد یا ، آدرس بازگشتی را به آدرس دیگری که حاوی مقداری نا معتبر است تغییر داد.می توان با به وجود آوردن یک متغیر ((با به وجود آمدن یک متغیر آدرسی برای آن نیز در نظر گرفته می شود)) با مقداری نا معبتر و دستکاری Offset و یا Return Address یک تابع و تغییر آن به آدرس مورد نظراین کار را انجام داد.



شکل۲) آدرس بندی حافظه در سیستم های ویندوزی و ۳۲ بیتی

چرا ما از Stack استفاده می کنیم؟

کامپیوترهای مدرن امروزی بر اساس احتیاجات زبان های سطح بالا طراحی شده اند.مهمترین تکنیک ها برای ساختن برنامه های ساخته شده با زبان های سطح بالا استفاده از توابع ((Function)) و سازنده ها ((Procedure)) هستند.یک فراخوان سازنده یا ((Procedure Call)) ریز کنترل را فقط هدایت می کند تا به دستور اصلی خود برسد اما بر خلاف آن، یک تابع هنگامی که کارش تمام می شود یک مقدار را به فراخوانی که تابع را فراخوانده است ، بر می گرداند و یا هیچ مقداری را بر نمی گرداند.این خاصیت زبان های سطح بالا به کمک Stack شکل گرفته است.همچنین Stack به صورت دینامیکی متغیرهای محلی استفاده شده در توابع را در حافظه Allocate می کند ، و برای ارسال کردن پارامترها به توابع ، و برگرداندن مقادیر از توابع استفاده می شود.

ناحیه Stack**:**

اگر بخواهیم تعریفی از Stack داشته باشیم ، Stack قسمتی از حافظه است که شامل اطلاعات گوناگونی می شود.Stack خلاصه ای از داده ها و اطلاعات عمومی استفاده شده در کامپیوتر است.این قسمت بخشی از حافظه است که توسط سیستم عامل و برنامه های گوناگون تنظیم می شود.متغیرهای Dynamic در این حافظه قرار دارند و در زمان اجرای برنامه مقداردهی می شوند.Stack دارای خاصیتی می باشد که اطلاعاتی آخر وارد شده اند ، به عنوان اولین خروجی خارج می شوند.این خاصیت به طور خلاصه LIFO یا Last in First Out است.چندین عملگر مختلف در Stack جای گذاری شده اند.دو تا از مهمترین این عملگرها ، که بسیار توسط برنامه نویسان و هکرها مورد استفاده قرار می گیرند ، عملگرهای PUSH و POP هستند.عملگر PUSH اطلاعات را به بالای Stack اضافه می کند.عملگر POP اندازه ی Stack را برگرداندن آخرین اطلاعات وارد شده که در ابتدای Stack قرار دارد ، کم می کند.این همان سیستم LIFO است.

نکته : عملگر یک دستور و یا علامت است که با مقادیر متغیرهای متفاوت سر و کار دارد و عمل هایی بر روی آن ها انجام می دهد.به عنوان مثال :

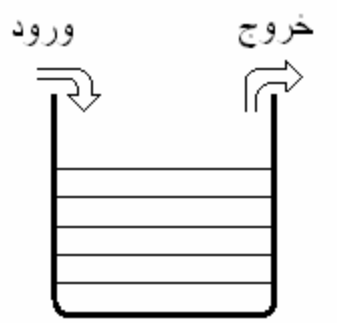
2+2=4

علامت + که دو عدد دو را با هم جمع می کند ، یک عملگر است.هر کدام از دو ها نیز یک عملوند هستند که عملگرها بر روی آن ها عملی را انجام داده اند.عدد 4 هم حاصل است.در مورد اندازه ی Stack نیز که توسط دو عملگر PUSH و POP کم و زیاد می شود ، مقادیر اضافه شده عملوند و حاصل ، کم و زیاد شدن اندازه ی Stack است.

یک اشاره گر به نام SP به بالای Stack اشاره می کند.سر Stack یا Stack Bottom در یک آدرس تنظیم شده است.سایز Stack نیز به صورت دینامیکی توسط هسته یا Kernel در زمان اجرا تنظیم می شود.دو عملگر که مقادیر موجو در Stack سر و کار دارند ، PUSH و POP هستند.

Stack علاوه بر موارد بالا ترکیبی از Frame های منطقی Stack است که در زمان فراخوانی توابع اطلاعات را نگه می دارند و در هنگام بازگشت مقادیر را برمی گردانند.Frame های Stack شامل پارامترهایی که توسط برنامه به توابع ارسال می شوند ، متغیرهای محلی برنامه ، و اطلاعات ضروری برای بازیابی یا Recovery کردن Frame های قبلی Stack است که شامل مقادیر اشاره گرهای آموزنده یا instruction در زمان فراخوانی توابع می شوند.

نکته:در هنگام فراخوانی توابع ، برنامه ی اصلی یا تابع دیگری را فراخوانده است پردازش دستورات آن به صورت موقت نگه داشته می شود و تابع فراخوانده شده مورد پردازش قرار می گیرد.حال اگر از این تابع خارج شویم به تابع یا برنامه ی قبلی و اصلی بر می گردیم و باید اطلاعات آن را پردازش کنیم.اما این اطلاعات کجا هستند؟ این اطلاعات به صورت موقت در Frame های Stack قرار گرفته اند.طبق چیزی که در بالا گفته شد.



شکل ۳) پشته

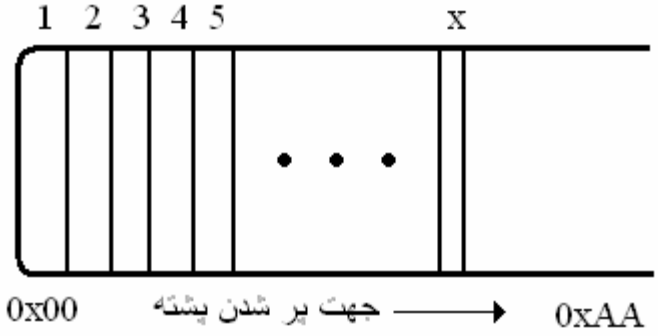
انواع پشته:

به صورت کلی می توان انواع پشته را به 3 دسته ، دسته بندی کرد:

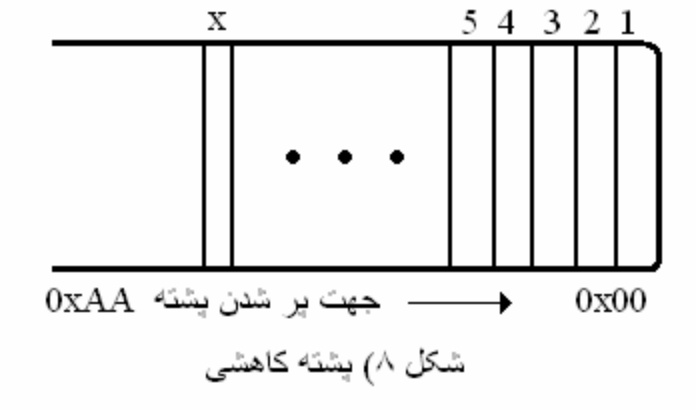
1) از نحوه آدرس دهی حافظه
2) از دید هویت
3) و نوع کاربرد SP

1) پشته از نحوه آدرس دهی در حافظه :

در مورد پر کردن حافظه از اطلاعات و داده ها ، ما دو نوع پشته داریم : پشته ی افزایشی((Grows Up)) و پشته ی کاهشی ((Grows Down)).پشته های افزایشی از پایین ترین آدرس حافظه به سمت بالا ترین آدرس حافظه شروع به مقداردهی و قراردادن اطلاعات در حافظه می کنند.پشته های کاهشی نیز از بالاترین آدرس حافظه به سمت پایین ترین آدرس حافظه شروع به مقداردهی و قراردادن اطلاعات در حافظه می نمایند.



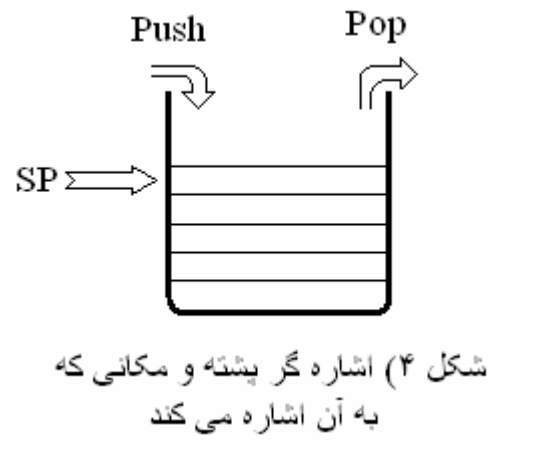
شکل ۷) پشته افزایشی



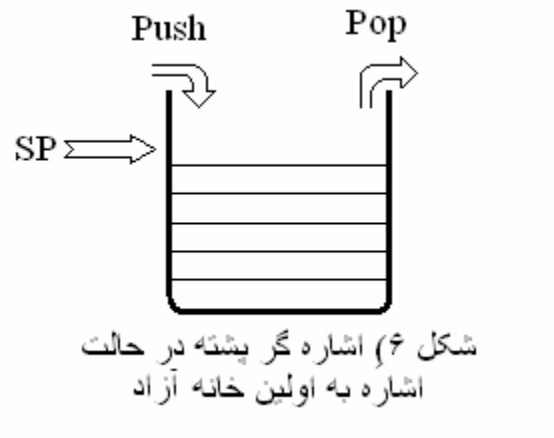
شکل ۸) پشته کاهشی

2)پشته از دید هویت :
همان طور که در برنامه های کامپیوتری برنامه های اجرا شدنی ((Executable)) و برنامه های غیر اجرا شدنی ((Non Executable)) داریم ، مانند برنامه هایی با پسوند exe و com به عنوان برنامه های اجرا شدنی و txt به عنوان فایل های متنی ، در پشته نیز دو نوع اجرایی و غیر اجرایی داریم.پشته های اجرایی شامل کدها و دستوراتی هستند که سیستم می تواند آن ها را مانند کدهای موجود در سگمنت کد یا DS ((قسمتی از سگمنت های پشته)) اجرا کند.در پشته ی غیر اجرایی حتی اگر کد قابل اجرا وجود داشته باشد ، سیستم اجازه ی اجرای آن را ندارد و در صورت تلاش برای انجام این کار نیز ، با خطای اجرایی از طرف سیستم عامل مواجه خواهد شد.
یک اکسپلویت نویس همواره از پشته ی اجرایی برای اجرا کردن Shellcode استفاده می کند زیرا بیشتر سیستم های عامل از پشته ی اجرایی استفاده می کنند.

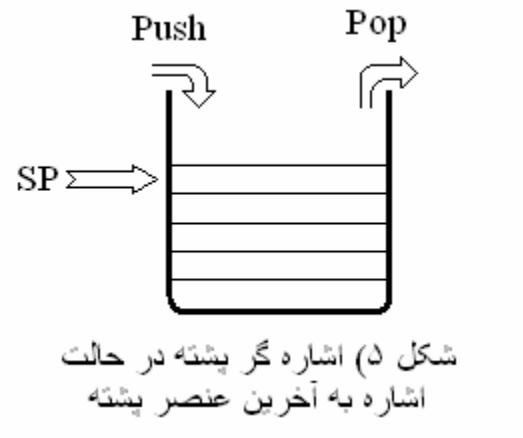
3)استفاده از SP:
هنگامی که اطلاعات را وارد پشته می کنید ، چگونه متوجه می شوید که پشته تا کجا پر شده است؟ میزان پر شدن پشته با اشاره گر پشته یا Stack Pointer مشخص می شود.این اشاره گر همواره به بالای پشته یعنی قسمتی که آخرین داده وارد آن می شود اشاره می کند.SP را می توان به دو صورت مورد استفاده قرار داد که نشان دهنده ی : بالاترین عنصر پشته و یا اولین خانه آزاد موجود در پشته باشد.در نوع اول که اشاره گر مقدار 1- و یا مقادیر مختلفی می گیرد که نشان دهنده ی آخرین عنصر پشته است.در نوع دوم اشاره گر مقدار0 می گیرد که نشان دهنده ی اولین خانه آزاد است.



شکل ۴) اشاره گر پشته و مکانی که
به آن اشاره می کند



شکل ۶) اشاره گر پشته در حالت
اشاره به اولین خانه آزاد

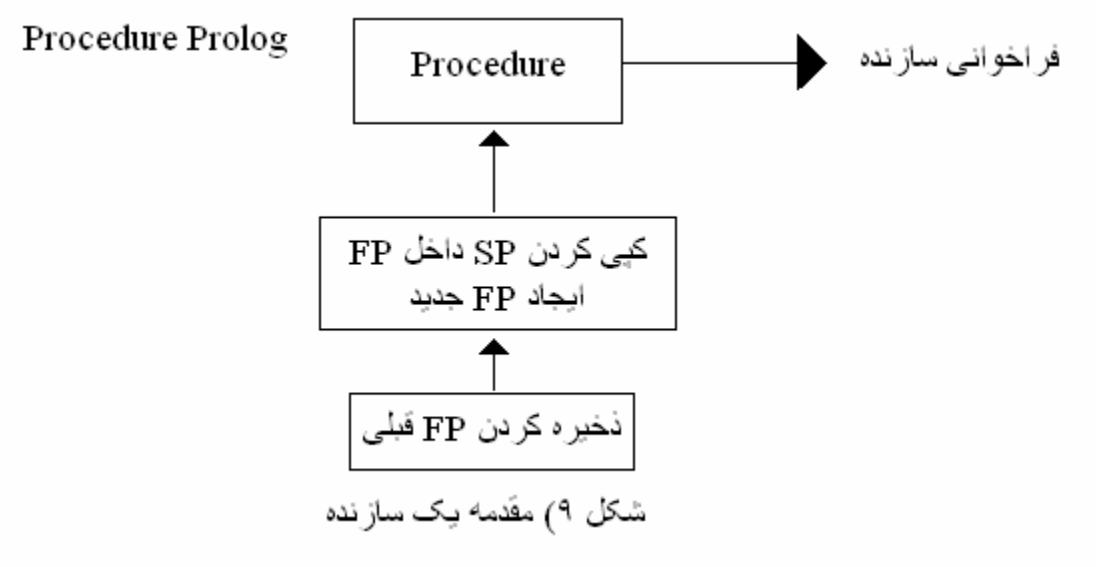


شکل ۵) اشاره گر پشته در حالت
اشاره به آخرین عنصر پشته

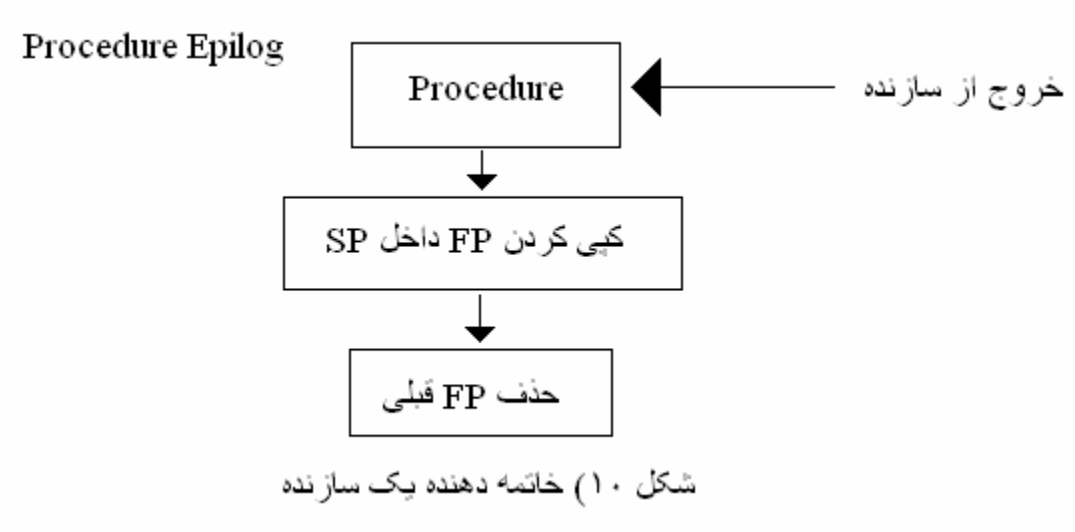
در پیاده سازی پردازشگرها یا CPU اندازه های مختلفی برای خانه های پشته در نظر گرفته شده است.به طور مثال اندازه ی یک حرف در یک ماشین با ماشین دیگر با CPU دیگری متفاوت است.ممکن است یک حرف در ماشینی به اندازه ی 4 بایت و در ماشین دیگری 2 بایت باشد.هم اکنون بیش از 92% کامپیوتر های جهان از استاندارد شرکت IBM که اندازه ی حروف را 2 بایت در نظر گرفته است ، پیروی می کنند.حال که با مقدمات و مفاهیم پایه ای Stack آشنا شده ایم می توانیم وارد قسمت اصلی Stack شویم.

در مورد SP باید این نکته را گفت که ، اشاره به بالای Stack ((پایین ترین آدرس)) ، برای راحتی کار یک اشاره گر Frame یا Frame Pointer دارد که به محل تنظیم شده ای در داخل یک Frame اشاره می کند.در واقع ، متغیرهای محلی می توانند پس از اتمام کار تابعی دوباره به جای اصلی خودشان در برنامه بر گردند و مقادیر خود را به دست آورند.این کار به وسیله ی گرفتن Offset قبلی شان از SP است.پس اگر محتویات این Offset دستکاری شود متغیر می تواند مقادیر غلطی را در خود نگه داری کند.این کار یعنی دست کاری مقدار موجود در Offset مشخصی اصلی ترین کار اکسپلویت نویس برای منحرف کردن برنامه از اجرای درست است.به هرحال ، همان گونه که کلمات داخل Stack قرار داده شده اند و از آن خارج می شوند ، این

Offset ها هم می توانند تغییر کنند و در بعضی حالت ها هم نمی تواند.در بعضی از ماشین ها مانند آن هایی که با پردازشگر Intel کار می کنند ، برای دسترسی به متغیری در محلی باید چندین instruction داشت.
اولین کاری که یک سازنده بعد از فراخوانی باید انجام دهد ذخیره کردن FP قبلی است که شامل مقادیر و اطلاعات موجود در برنامه است ((که به این صورت اطلاعات برنامه همان طور که گفته شد مجددا بازیابی می شوند)).بعد از این کار کپی کردن SP فعلی در یک FP دیگر برای ایجاد یک FP جدید است.و به این صورت SP قبلی فضا و Offset های متغیرهای محلی را حفظ می کند.این اطلاعات ذخیره شده را مقدمه ی سازنده یا Procedure Prolog گویند.بعد از اتمام کار Procedure ، Stack باید یکبار دیگر اطلاعات را از بارگذاری کند.این کار را خاتمه ی دهنده ی سازنده یا Procedure Epilog گویند.instruction های ENTER و LEAVE پردازشگر Intel و LINK و UNLINK پردازشگر Motorola ، برای انجام بیشتر کارهای Prolog و Epilog درست شده اند.



شکل ۹) مقدمه یک سازنده



شکل ۱۰) خاتمه دهنده یک سازنده

مقدماتی که در مورد حافظه ی Stack گفته شد شما را تا حدودی با این حافظه در حد مقدماتی آشنا کرد.برای نوشتن اکسپلویت های Stack Overflow باید کمی بیشتر با این حافظه آشنا شد.

Copyright© by Siahacker
2005-2006 All Rights Reserved
Email:Siahacker@gmail.com