

ShellCodes

Por: Jonas Monteiro Carreira (Consultor em Segurança da Informação)
E-mail: j3l15@hotmail.com

1-Introdução

2- Processos de memória do sistema

3-SysCall

4- ShellCodes

5-Protégendo a memória

5.1-História

5.2-Paz à memória

5.3-As Classes de Exploits

5.4-Classe1

5.5-Classe2

5.6-Classe3

6-Conclusão

1-Introdução

Muito tem se falado sobre vulnerabilidades que conseguem colocar o invasor em situações privilegiadas em sistemas. Porém, muitas vezes, o administrador não tem muito conhecimento para lidar com esse tipo de situação ou, algumas vezes, apenas um patch já facilita a sua vida.

O estudo sobre ShellCodes pode ser algo muito ultrapassado nos dias de hoje, devido ao fato de existirem ferramentas sofisticadas que, de uma maneira ou de outra, podem nos deixar “bitolados”, insipientes de como o mundo real dos invasores funciona. Não tenho a utopia de mudar mentes, mas venho escrever este artigo para expor um assunto, que, creio eu, já esteja bastante difundido para alguns, e não para outros, então estou tentando tornar estes conhecimentos 100% difundido para todos.

Este artigo trata sobre segurança de aplicações que são rodadas na sua máquina. Vou relatar como pode ocorrer uma ShellCode e irei falar da nova moda de proteção da memória da máquina que vem protegendo usuários do Adamantix (os Paxs). Vamos ver como funciona os patches de proteção. E sobre o projeto OpenWall de Linux Torvalds. Boa Leitura!.

2- Processos de memória do sistema

Sabemos que as memórias do nosso sistema estão ocupadas por processos que fazem o sistema ser executado. É na memória onde encontramos as famosas Stacks ou Pilhas e é na pilha da memória onde podemos executar um ShellCode.

No Stack é que estão armazenados os dados do sistema. Ao armazenar dados, os Stack desenvolvem um processo onde o primeiro dado a entrar será o último a sair. Podemos usar um pouco de programação em Assembly e python para visualizar este esquema:

Assembly:

PUSH---->Coloca os dados

POP----->Retira os dados ("o último a entrar será o primeiro a sair").

Python:

Podemos fazer um experimento em python utilizando as listas como o processo de memória da pilha. Onde o primeiro a entrar será o último a sair: Em Python, usamos o comando Append() para adicionar um dado na pilha, e como em Assembly também usamos Pop() para retirar. Veja:

```
Stack = [3,4,5]
Stack.append (6)
stack.append (7)
stack
[3,4,5,6,7]
```

```
#Aqui está uma variável com o nome Stack, recebendo 3 valores.
#Aqui a variável Stack recebe um valor com o comando Append.
#Idem anterior.
#Aqui, executamos a variável. E o resultado seria:
```

Vemos que inserimos os dados 6 e 7 na nossa variável que está representando o Stack do nosso sistema. Agora, usaremos o comando Pop para retirar esses dados da variável:

Ao digitarmos no nosso interpretador Python o comando `Stack.pop()`, veremos que ele retirará o último dado a entrar que é o 7, e assim por diante. Não irei me aprofundar muito em programação, usei o python por ser uma linguagem de fácil aprendizado para o leitor.

3-SysCall

Bem, não podemos deixar de falar de SysCalls, quando o assunto é ShellCodes. As SysCalls são funções usadas pelo sistema operacional em linguagem de máquina, com o objetivo de fazerem chamadas ao sistema.

Uma SysCall serve para facilitar certas tarefas ao se abrir um determinado aplicativo no seu sistema. Sem uma SysCall, teríamos que digitar milhares de códigos em linguagem de máquina para abrir um bloco de notas por exemplo.

Geralmente, quando uma SysCall está sendo executada, ela entra na interrupção `int 80`. Essa interrupção faz o Kernel entrar em Kernel Mode.

Um SysCall em conjunto com as LKMs (Linux Kernel Modules) podem favorecer muito o programador mal-intencionado. Como visto no artigo anterior em que falo sobre Backdoors.

As funções de SysCall são muito utilizadas em Rootkits e ShellCodes bem programadas.

4- ShellCodes

Um estudo de falhas em sistemas não é tão fácil e simples para futuros Crackers. Depois de descobrir a falha, o Cracker terá que escrever o exploit. Depois disso, vem a parte em que ele terá que escrever a ShellCode, ou seja, uma ShellCode lhe proporcionará o controle total do sistema invadido.

Uma ShellCode é apresentação de uma determinada String de sistema em representações Hexa decimal dentro de uma variável do tipo Char. Uma String de sistema seria uma Instrução de baixo nível – podemos dizer em linguagem assembly – para executar um processo de chamada de sistema (SysCall).

Uma String bastante utilizada nos exploits são os diretórios Bin/Sh (responsáveis pelos arquivos executáveis em sistema Linux), mas não só os Bin/Sh possuem possibilidades de Shell.

O passo para se construir uma ShellCode é tentar uma String do tipo Bin/sh na memória da máquina e, junto à string, uma SysCall. E colocá-la em Array de variável do tipo Char, dentro de um Exploits.

5-Protetendo a memória

Sempre tem se falado sobre a proteção de memória ao nosso sistema. Podemos ver várias vulnerabilidades surgindo por aí e com elas surgem os Exploits e as ShellCodes. No meio disso, podemos presenciar os especialistas em segurança correndo atrás de Patches para tampar os furos na segurança de suas aplicações.

Vemos muito pouco se falar sobre os processos de memória em sistema Linux, o que é algo muito importante para a segurança do sistema, pois é nos processos de memória que podemos executar códigos. Podemos observar a existência de poucos recursos de proteção de memória em contraste com um grande número de exploits surgindo pela Internet.

5.1-História

Inicialmente, o mundo da segurança girava em torno da criptografia das aplicações, podendo, assim, ter proteção para a confiabilidade de suas informações, mas a tecnologia da invasão como sempre estava, a cada dia, um passo a frente e formou uma nova técnica de invasão, (Buffer Overflow) tornando vulnerável a integridade de nossas aplicações. O desenvolvimento deste tipo de técnica foi aumentando até se tornar um fator importante em análises de riscos para empresas que querem proteger suas informações.

O grande alvo dos Estouros de Pilhas são os sistema OpenSource, onde seu desenvolvedor, Linus Torvalds, deu início ao projeto OpenWall(<http://old.lwn.net/1998/0806/a/linus-noexec.html>) um patch que seria incluído no Kernel do Linux para proteção de estouros de pilhas. O OpenWall foi um grande passo na história de proteção de memória em sistemas linux, mas isso não satisfazia a comunidade, nem Linus Torvalds. Simplesmente, pelo fato de que havia mais áreas da memória em que deviam ser protegidas.

5.2-Paz à memória

Conhecidos como Pax (paz em latim) (<http://pageexec.virtualave.net>). É um patch para proteção de memória, que vem deixando usuários do Adamantix tranquilos com falhas do tipo que corrompem a memória. Disse Peter Busser (criador do Adamantix).

Tudo começou no início de um projeto que pudesse proteger a memória de todos os tipos de exploits construídos por Crackers.

Usuários do Gentoo já estão portando no Kernel esse Pax originalmente adotado pela Grsecurity (www.grsecurity.net). Com isso, foi se criando um crescimento de programadores para ajudar no desenvolvimento do Software.

O que tornou o Pax famoso na área de segurança é que ele não protege o sistema para apenas um tipo de Exploits, mas para todas as Classes de Exploits existentes. Vamos ver as Três principais classes de falhas que existem.

5.3-As Classes de Exploits

No esquema de proteção de memória, há 3 tipos de casos de vulnerabilidades que envolvem a corrupção da memória:

- 1) Execução arbitrária de código.
- 2) Execução de código existente, mas fora da ordem original do programa.
- 3) Execução de código já existente na ordem original, mas com informações arbitrárias.

Vários exploits espalhados pela internet são de Classe 1. A grande maioria dos patches consegue proteger exploits da classe 1. A Classe 2 foi o alvo do projeto OpenWall de proteção. A classe 3 é menos usada em exploits, mas não deixa de ser ainda um risco.

Bem, são essas Classes que o Pax tenta proteger. Ainda não temos certeza se ele pode proteger contra exploits da classe 3, mas seu criador criou o Paxtest um pacote de exploits que tem o objetivo de testar a eficiência do Pax dentro do Kernel do Adamantix. Por enquanto, o Pax vem sendo bastante útil. Podemos baixar nosso Paxtest no seu site oficial em: <http://pageexec.virtuallave.net> e rodá-lo em seu sistema com o comando `apt-get install paxtest` (Obs: Este comando é para o Kernel Adamantix).

Bem, quando foi o Pax testado, eu pensei que ele deixaria alguns daemons sem funcionar. Muito pelo contrário, todos funcionaram bem. Pensei até que não estava rodando nenhum patch de segurança no Kernel, foi aí que me levei pela curiosidade de baixar o Paxtest e tudo ocorreu bem até agora.

Veremos alguns conceitos de classes de exploits.

5.4-Classe 1

Os ataques de classe 1 são representados das seguintes maneiras:

1 - Sobrescrever códigos que estão escritos na memória da máquina e executá-los como se fossem partes dos originais.

Com a explicação acima podemos citar que é possível escrever algum código malicioso dentro da memória, de maneira vulnerável e executá-lo ao carregar o código dentro do sistema. Ou seja, ao invés do programa executar a sua tarefa rotineira ele irá rodar aquilo que o invasor quiser.

Este tipo de técnica é usado pela maioria dos ataques de BufferOverflows. O Pax promete distinguir o código original do malicioso e não deixa que nenhum outro seja executado nela.

5.5-Classe 2

No ataque de classe 2 o invasor sobrescreve com novos dados o endereço de memória para controlar a forma em que os processos trabalham. Um processo seria o valor de retorno de um ponteiro usado por uma função dirigida ao programa, mas invés disso, irá retornar para o endereço que o invasor quer que retorne.

Na teoria existem várias formas de retorno para o ponteiro. No nosso estudo, podemos dizer que seria um Root ou administrador do sistema alvo.

5.6-Classe 3

Aqui está a classe 3. Esta classe é um tipo de técnica bastante rara, mas acontecem alguns casos em que elas são implementadas. Na classe 3, os dados importantes de um sistema são alterados. O que acontece é que para um sistema chegar a uma informação importante, ele seguirá um determinado caminho. O invasor pode simplesmente alterar esse caminho, determinando qual seguir.

Podemos usar um exemplo em Linux: Um sistema usa permissões de usuários, implementando que somente o usuário "A" poderá acessar um determinado diretório. Um usuário "Invasor" pode manipular os dados do sistema para que o faça entender que o usuário "invasor" é o usuário "A".

6-Conclusão

Bem, vimos um pouco sobre Shell e observamos detalhes dos Paxs para proteção de aplicações. Há muito que se discutir sobre esquemas de proteção da memória como os IDS (sistemas detectores de intrusos), que fazem varreduras no Kernel para tentar interromper execuções suspeitas ao sistema. Os Pax foram levados para discussão neste artigo por ser uma técnica nova para a área de segurança e sendo o mais seguro até o momento.

Abraços e até a próxima!

Jonas Monteiro Carreira.