RESEARCH GROUP

“Ataques Polimórficos”

Rodrigo Rubira Branco

rodrigo@kernelhacking.com
rodrigo@risesecurity.org

RESEARCH GROUP

A idéia

- Detectores de intrusos utilizam-se de assinaturas de ataques para identificação dos mesmos

- Sistemas de anti-vírus fazem uso das mesmas técnicas para identificar códigos viróticos

- Os escritores de vírus adicionaram diversas técnicas de polimorfismo aos vírus, evitando-se assim sua detecção e iniciando um ciclo de combate onde novas técnicas de segurança surgem e novas formas de burlarem as mesmas vem logo a seguir

- Por quê não fazer o mesmo contra detectores de intrusos?

RESEARCH GROUP

O que veremos?

- Esta apresentação não enfocará em outras técnicas de evasão de detectores de intrusos, tais como fragmentação de códigos de ataques e inversões em camadas de aplicações, apenas no polimorfismo

- Quando aqui for citado detector de intrusos, refere-se a qualquer tipo de sistema que detecte a intrusão, seja ele um IPS (detector de intrusos inline) ou um IDS, ou até mesmo um Firewall com recursos de bloqueio de ataques (tais como os existentes no Checkpoint)

RESEARCH GROUP

Assinaturas

- Chama-se assinatura de um ataque qualquer tipo de característica que este ataque possua que permita que o mesmo seja detectado

- Tais assinaturas podem cobrir diversos aspectos (como característica do tráfego de rede gerado, padrões de protocolo, tamanho das estruturas de dados e análise do conteúdo para localização de código malicioso)

- Cada uma destas características pode ser modificada por técnicas diferentes e normalmente são complementares para os diversos tipos de ataques (já que apenas uma delas muitas vezes não mostra-se suficiente para determinar um ataque)

RESEARCH GROUP

Quem usa assinaturas?

RESEARCH GROUP

Polimorfismo

- Capacidade de se existir em múltiplas formas

- Consiste em ter-se um código capaz de se auto-modificar quando executado

- Devido a esta auto-modificação na execução, torna-se impossível detectar o código malicioso quando passa pela rede (evasão de detectores de intrusos) ou quando está armazenado em um arquivo no computador (evasão de softwares anti-vírus)

RESEARCH GROUP

Nivelando conhecimentos

A abordagem mostrará a arquitetura Intel-x86 onde todos os registradores são de 32 bits e podem ser divididos em sub-secões de 16 ou 8 bits

32 bits	16 bits	8 bits	8 bits
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL

RESEARCH GROUP

Nivelando conhecimentos (cont...)

- "Inline Assembly" - AT&T
- Nome dos registradores é precedido por "%"

MOV – Copiar valor para um registrador
mov %0x8, %al
PUSH – Colocar dados na Stack
push $0x0
POP – Carregar dados da Stack para um registrador
pop %edx
INT – Interrupção via software – 0x80 passa o controle para o kernel
int $0x80

RESEARCH GROUP

Como um código polimórfico se parece?

---

call decoder

---

shellcode

---

decoder

---

jmp shellcode

RESEARCH GROUP

Como funciona

O decoder é responsável por realizar o processo inverso utilizado para codificar o seu shellcode.

Este processo pode ser:

- ADD
- SUB
- XOR
- SHIFT
- Criptografia (ex: DES)

RESEARCH GROUP

“Encriptando o shellcode”

- A criptografia do shellcode não precisa utilizar um algoritmo real de criptografia e sim um método simples qualquer, pois a análise do detector de intrusos não tem como ser profunda (imagine ele realizando diversos testes de XOR com todos os valores possíveis, depois todos de ADD, etc...)

- Pega-se o shellcode original e passa-se ele por uma rotina em C ou ASM básica que gera o shellcode criptografado

- Após isso, concatena-se o shellcode criptografado ao respectivo decoder e têm-se o shellcode final (agora polimórfico)

RESEARCH GROUP

Como funciona

```
call decoder
shellcode:
.string shellcode_encriptado

decoder:
xor %ecx, %ecx
mov sizeof(shellcode_encriptado), %cl
pop %reg

looplab:
mov (%reg), %al
- manipula AL conforme a criptografia -
mov %al, (%reg)

loop looplab

jmp shellcode
```

Como localizar o shellcode na memória?

- Quando uma chamada CALL é realizada, o próximo endereço (no caso da estrutura apresentada, o endereço do shellcode criptografado, é armazenado na stack (para permitir o retorno da chamada para o próximo código a ser executado)).

- Neste caso, o decoder pode facilmente pegar o endereço do shellcode fazendo um pop para um registrador qualquer.

- Então basta o decoder manipular os bytes do shellcode e fazer um jmp para o endereço do mesmo.

Mudando a estrutura mostrada

- Após entender o funcionamento do código polimórfico e do decoder, basta concatenar o shellcode criptografado ao fim do decoder.

- Modificar o byte relativo ao mov sizeof(shellcode_encriptado), %cl.

- Tomar o cuidado de evitar qualquer zero-byte (0x00) no opcode do decoder e do shellcode encriptado, porque o sistema entenderá como fim da string.

RESEARCH GROUP

Dificuldades

- Evitar badchars (\r \t \0)

- Gerar shellcodes alphanuméricos (filtros isalpha())

- Eliminar constantes em decoders (podem ser detectados)

- Manter decoders para várias plataformas e sistemas

- Inserir instruções “do nothing” no meio e em partes do código final gerado

- Otimizar o tamanho do decoder

RESEARCH GROUP

SCMorphism

- A intenção não é apenas encriptar o shellcode original com um valor randômico e inserir o decoder. A ferramenta também possui diversos tipos de decoders

- Pode gerar decoders XOR/ADD/SUB/ByteShitf para qualquer lado/Inversão de bytes/INC/DEC/Alphanuméricos.

- Os decoders são divididos em peças (idéia de zillion@safemode.org).

- Através destas peças um mesmo decoder pode se parecer de diversas formas e também facilita a inserção de instruções “do nothing” no decoder.

SCMorphism

```
Usage:
-f <shellcode file> <var name> <shellcode lenght>
-t <type>
-n <number>. Value used in single decoder operations.
-x <number>.  Value used when xor numbers.
-a <number>.  Value used when add numbers.
-u <number>.  Value used when sub numbers.
-r <number>.  Value used into rotate operations
-e --> Execute the generated shellcode
-s <filename> --> Stdout form: to a file, if not gived, use console
-E <varname> --> Stdout form: ENV var: varname
-b <badchar1,badchar2,badcharN> --> Deletee this chars from shellcode
-c <archfile>,<osfile>,<typefile>
-A <nopsize in bytes> --> Gen. random alphanumeric junks only
-j <nopsize> --> Size of NOP include in the shellcode
-T <type> --> Type of NOPs:
-L <location> --> Location of NOPs:
-v --> Version and Greetz information
-h --> Help stuff
```

SCMorphism - Benefícios

- As instruções do nothing utilizadas e o decoder randomicamente gerado tornam impossível de se escrever assinaturas para a ferramenta (ou o admin teria milhões de falsos positivos).

- Quando a função de codificação é chamada, ela gera um número randômico para ser usada com a peça do decoder escolhida.

- O número escolhido será utilizado para performar as operações sobre os shellcodes

- Caracteres definidos como ruins (BADCHARS) serão removidos do código gerado, permitindo assim passar por outros filtros de caracteres

RESEARCH GROUP

Badchars e “Do nothing”

- Caso o usuário não escolha nenhum badchar, o sistema usará os famosos (\r \t \0 – obrigado para Ramon de Carvalho – Rise Security).

- Caracteres definidos como ruins (BADCHARS) serão removidos do código gerado, permitindo assim passar por outros filtros de caracteres.

- A ferramenta possui um array de instruções “do nothing” que não atrapalham na execução do shellcode (obrigado novamente ao Ramon pela ajuda na população deste array). Este array é usado para gerar randomicamente as instruções para o meio do decoder, tornando as possibilidades de combinações gigantescas.

Detecção

- Algumas técnicas vem sendo desenvolvidas para se detectar este tipo de ataque

- A maioria delas conta com emulação de código (difícil de se implementar, excessivamente lenta e ainda com muitos problemas de plataforma) ou então verificação em busca de instruções de máquina válidas (para tentar detectar o decoder)

- Testes com decoders metamórficos (onde até mesmo o decoder se auto modifica) mostram-se eficazes contra a segunda classe destes sistemas e loopings infinitos (pulados pela execução normal do código) acabam dificultando a primeira classe

RESEARCH GROUP

Praticando

- Bind shellcode simples
- Testando o mesmo shellcode polimórfico

- Exploit sendo detectado via checagem de instruções
- Exploit sendo detectado via emulação de código

RESEARCH GROUP

Referências

http://cs.southwesternadventist.edu/
http://rosec.org
http://m00.void.ru
http://www.infosecwriters.com/text_resources/pdf/basics_of_shellcoding.pdf
http://www.securityfocus.com/infocus/1768
http://www.priv8security.com
http://www.intel.com/design/Pentium4/documentation.htm#pap
http://www.phrack.org
http://www.kernelhacking.com/rodrigo

RESEARCH GROUP

Fim! Será mesmo?

Dúvidas ?

Rodrigo Rubira Branco

rodrigo@kernelhacking.com
rodrigo@risesecurity.org