

Rootkits

Rootkits are trojan backdoor tools that modify existing operating system software so that an attacker can keep access to and hide on a machine.

User-mode rootkits

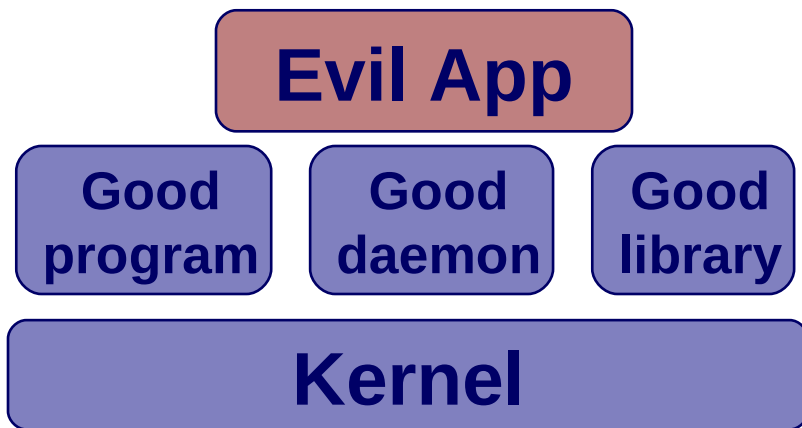
Modifications to the system at the user-level that hide the attacker and/or provide backdoor access

- Replace or modify normal binary executables and libraries
- Direct process tampering

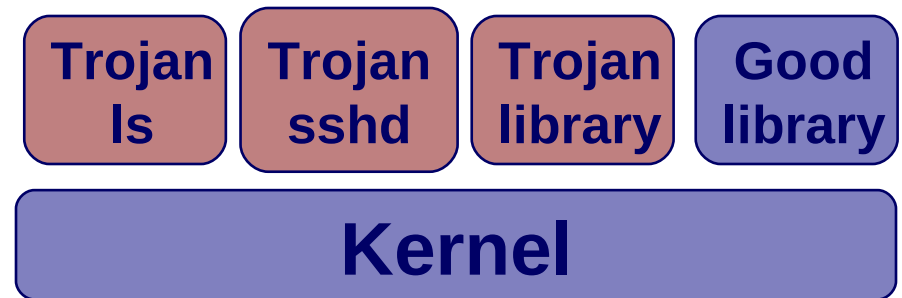
User-mode rootkits

Differentiating between a trojan backdoor and a root kit

Application-level Trojan Backdoor



User-Mode Rootkit

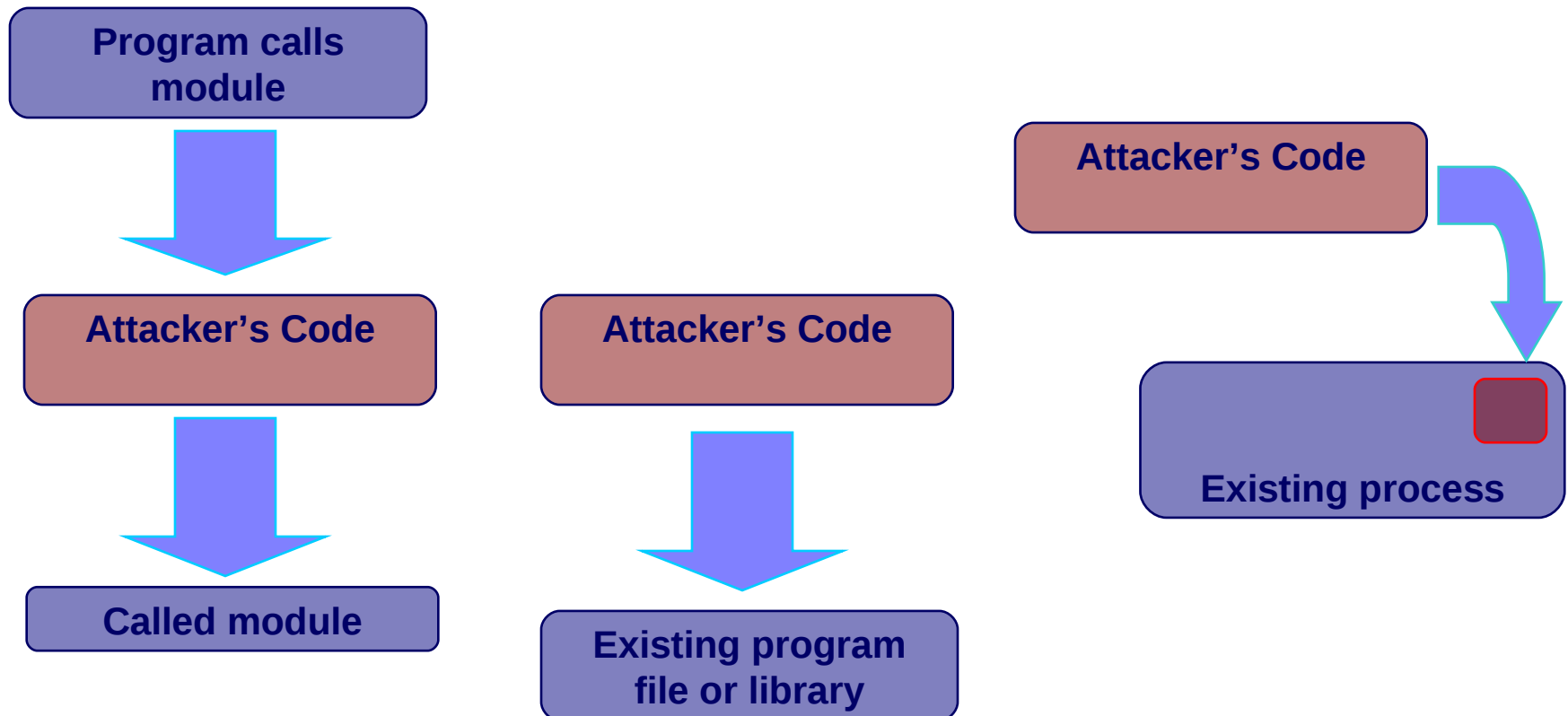


Three Techniques

Interpose

Replace or modify binary/library

Inject code into memory image of process



Executable replacement

Binary executable

Modify or replace program executables, libraries, or kernel modules in the file system to perform rootkit functions

- **Replace executable/library and reimplement function**
- **Replace executable/library and have it forward call to real version**

DLL companion attack

Place trojan copy of DLL in process' working directory

- Will favor local copy over system copy
- Wraps real system dll
- Can find dependencies with standard tools

Via the Registry

- AppInit_DLL key
- Add a DLL that hooks IAT, kernel32.dll or ntdll.dll
 - e.g. inject code for network backdoor in Notepad.exe.

Hot patching in memory

Hot patches

Modify executable code in memory as program executes to get it to do what you want

- “Direct code-byte patch”
- Hard to detect
- Must be able to read/write memory where functions reside

Example

- Replace function calls (like running a security routine) with NOPs

DLL (Library) injection

DLL injection into running process

Force a running EXE process to accept an unrequested DLL forced into its memory space.

Steps

- Allocate space in the victim process for code to inject DLL
- Write DLL injection code into the memory space of the victim
- Create or hijack a thread in the victim to run/load the DLL
- Clean up tracks

Preserving original functionality

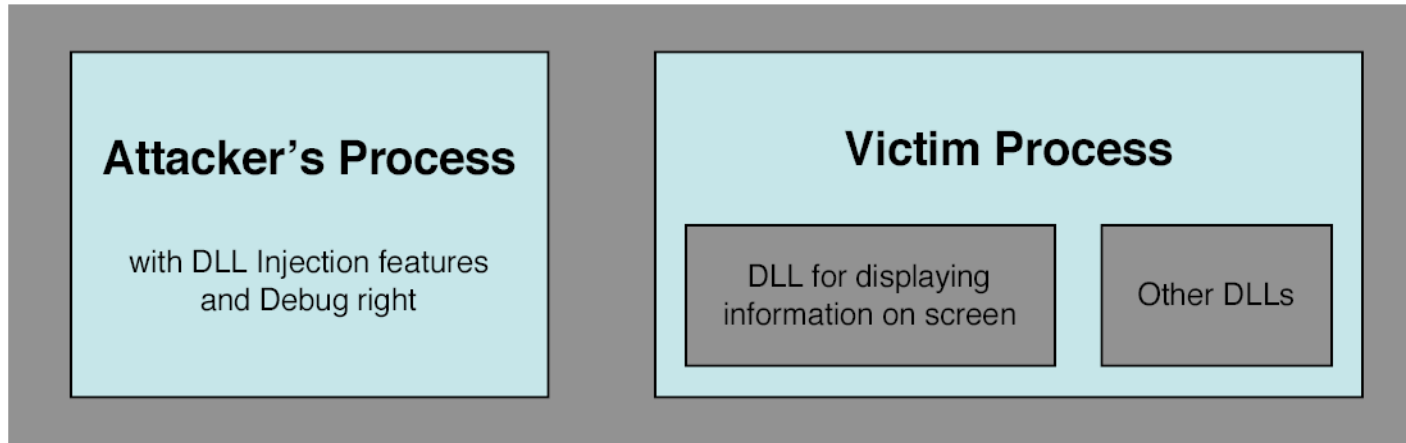
- Still need original functions to work correctly
- Injected DLL often set up to call original DLL to support desired functionality
- Interposed between application and real DLL

Example tool

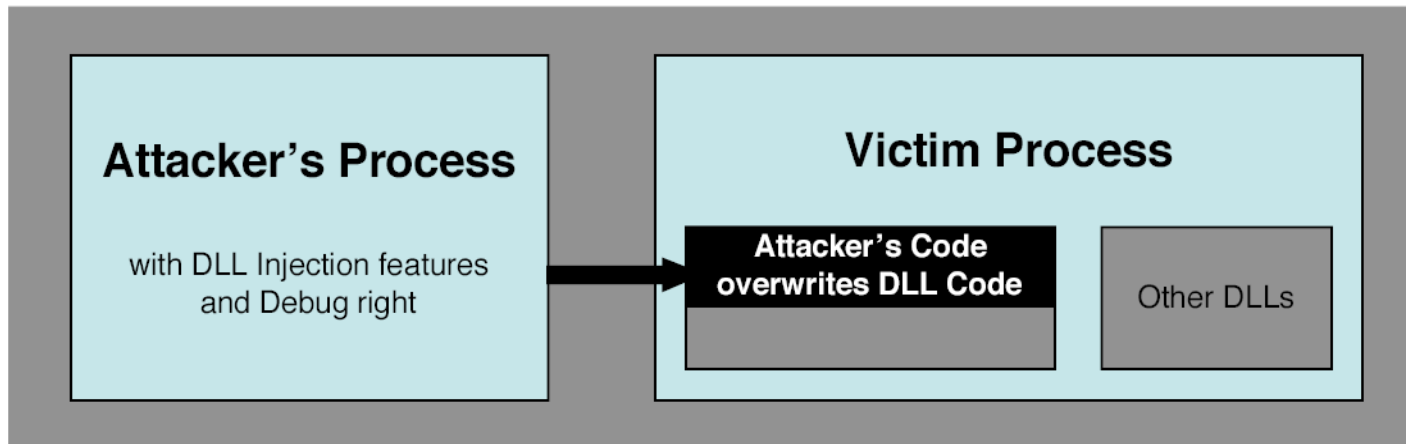
- Inject.exe (Aphex)
- C:\> inject.exe winlogon "myrootkit.dll"

DLL injection into running process

Before



After



DLL injection into running process

Method #1: Debug interface

Attacker's process uses various Windows Debug API function calls to inject code AND make victim process run it.

Attacker must have Debug programs rights on system

Get debugger attached to process and run

- Break when you want to inject
- Obtain code to inject/load a DLL into memory space
 - Analyze PE header to find a usable, writable part of memory for code
 - ReadProcessMemory to save what is there
 - WriteProcessMemory to write injection code
 - » Include INT 3 at end of injection code for debugger to stop
 - » Debugger will break on this instruction
 - Set EIP to start of code to inject a DLL and continue
 - Breaks when DLL loaded, restore original state of memory (i.e. remove code to inject DLL)

DLL injection into running process

Method #2: Remote threads

With proper privileges, Windows allows one to create a thread on a remote process

- **CreateRemoteThread**

Often used in DLL injection attacks

- **Steps**
 - **Create or hijack thread**
 - **Have thread jump to code that performs LoadLibrary**
 - **Load rogue DLL**
 - **Destroy thread**
- **Allows one to avoid detection**

Function hooking

Function hooking

Mechanism used to redirect function calls to injected attack code

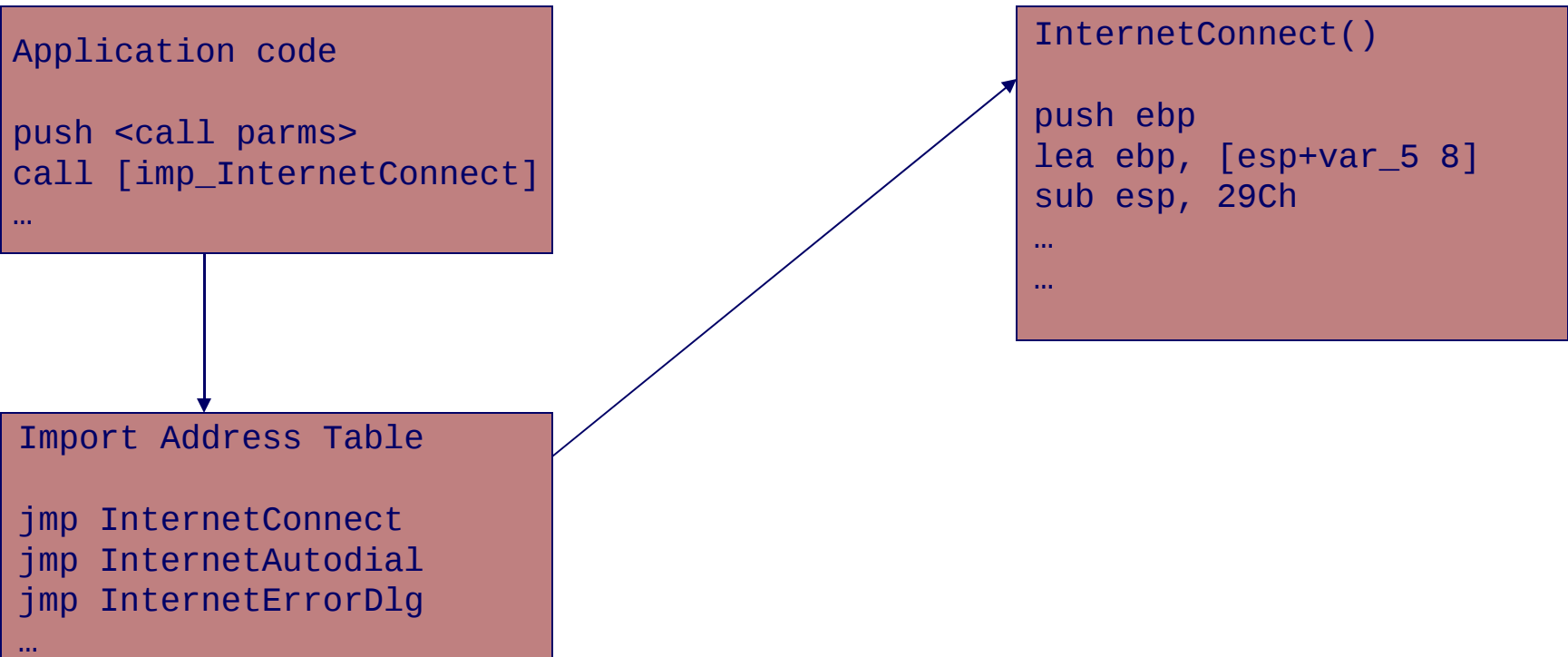
- Replaces legitimate function with alternative one

Two general methods

- Hooking function tables
 - Run-time data structures that contain function pointers that are invoked during program execution
- Hot patching function invocation
 - Modify JMP/CALL targets
 - Modify function prologs to add detour to trampoline

Library (IAT) hooks

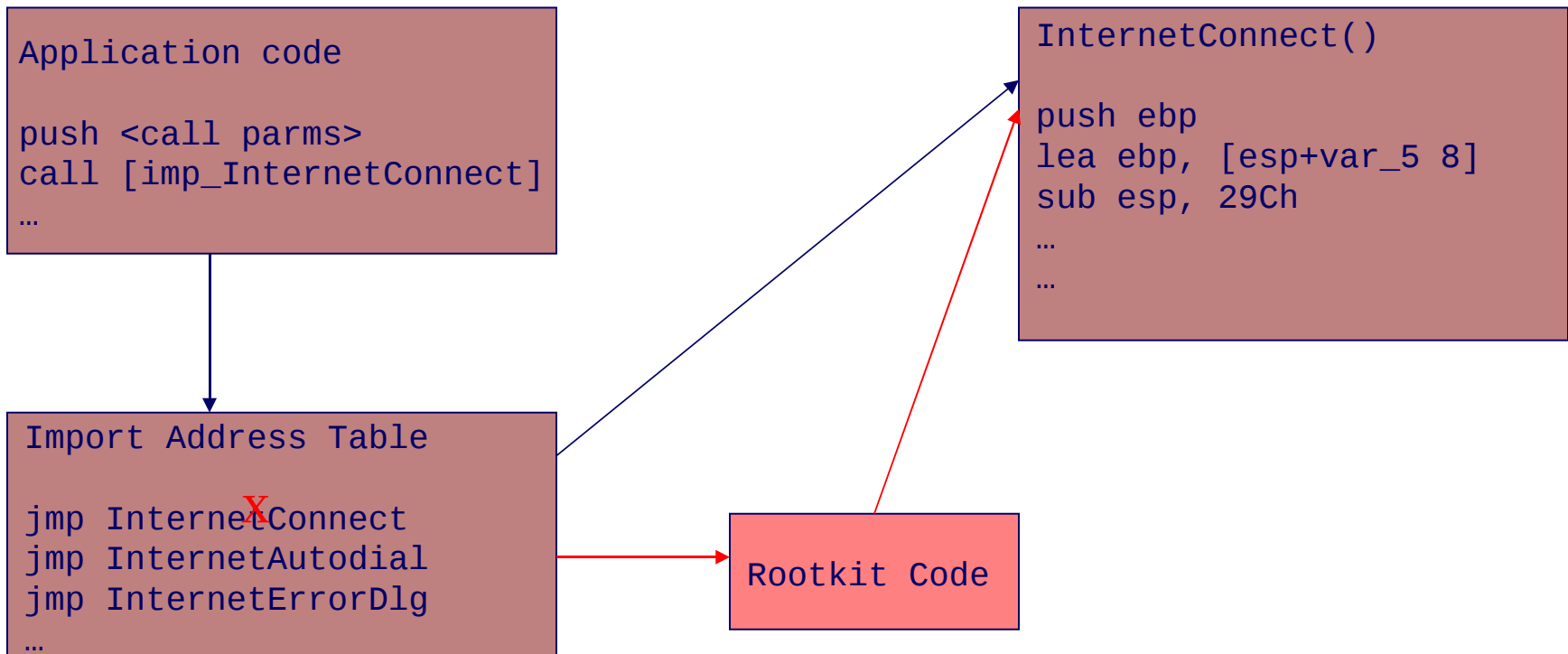
Import Address Table (IAT) used to call functions in libraries



Library (IAT) hooks

Modify IAT to hijack a DLL call

- Makes a hack 'portable' to other applications
- Load rootkit hook function into memory
- Replace target function's address in the IAT with address of hook function
- Integrity check must verify contents that IAT entries point to



Library (IAT) hooks

Powerful and simple

- Read and parse PE header of library
- Locate import section from IAT
- Find IMAGE_IMPORT_DESCRIPTOR chunk of DLL that exports that function
- Locate IMAGE_THUNK_DATA which holds original address of imported function
- Replace address in IAT to point to your function and have your function eventually call the original

Detection problems

- Legitimate hooking common
 - Methods such as DLL forwarding makes benign vs. malicious hooks hard to discern
- Late binding
 - Applications do late-demand binding where function addresses are not resolved until called
 - Reduces amount of memory used
 - But, won't know what the legitimate values should be!

Library (IAT) hooks

Dynamic IAT hooks

- Dynamic (in program) DLL loads have no IAT entries initially
- Hook LoadLibrary()/GetProcAddress() API to return interceptor function address instead of real one

Callback hooks

Functions that are called when particular events happen (event handlers)

- Rewrite original function pointer to your function
- Useful in executing automated code for particular events
- Example: Windows callback routines that run when processes and threads are loaded
 - `PCREATE_PROCESS_NOTIFY_ROUTINE`
 - `PCREATE_THREAD_NOTIFY_ROUTINE`
 - `PLOAD_IMAGE_NOTIFY_ROUTINE`

Hooking a hook chain

Windows processes some events via a chain of functions

- Window messages and events of another process
- More common in the kernel (next lecture)
 - I/O events with chain of handlers

Examples: system libraries

Three OS subsystems processes depend on

- Win32
- POSIX
- OS/2

Processes rely on APIs provided by above

- DLLs loaded at runtime into process address space
 - Kernel32.dll, User32.dll, Gui32.dll, Advapi.dll
 - Kernel32 loaded into private address space between 0x00010000 and 0x7FFE0000
 - Addresses of functions placed in Import Address Table (IAT)
- Hook code after it is loaded or modify IAT to point elsewhere
- Example: Hiding files in a directory
 - Replace FindFirstFile(), FindNextFile() in Kernel32 to skip rootkit files

Examples: system libraries

Hook keyboard/DirectInput APIs to obtain keyboard/mouse events

- `GetKeyboardState()`, `GetKeyState()`, `GetDeviceState()`, etc.

```
SHORT WINAPI FakeGetAsyncKeyState(int vKey)
{
    SHORT nResult = 0;
    if (g_bNeedMP) {
        if (vKey == VK_M) {
            nResult |= 0x8000; // M pressed
            g_bNeedMP = FALSE;
        }
    }
    else
        nResult = RealGetAsyncKeyState(vKey);
    //...
    return nResult;
}
```

Examples: system libraries

Window message processing

- Inject or obtain messages we want to process
- Hook WndProc calling chain of process by window subclassing
- Via RegisterClass() API
- SetWindowsHookEx
 - Change function hooks in chain for handling window messages
 - API call specifies the process/thread to hook
 - Set to 0 and the system hooks all threads in the current Windows desktop!

```
DWORD nOldWndProc = 0;
DWORD nNewWndProc = 0xbaadcode;
ATOM WINAPI NewRegisterClass(CONST WNDCLASS *lpWndClass) {
    nOldWndProc = (DWORD)lpWndClass->lpfnWndProc;
    lpWndClass->lpfnWndProc = (WNDPROC)nNewWndProc;
    return OldRegisterClass(lpWndClass);}
```

```
HWND hWnd = NULL;
DWORD nOldWndProc = 0;
DWORD nNewWndProc = 0xbaadcode;
hWnd = FindWindow(_T("EasyGameClient"), _T("EasyGameClient"));
if (hWnd) { nOldWndProc = GetWindowLong(hWnd, GWL_WNDPROC);
    SetWindowLong(hWnd, GWL_WNDPROC, nNewWndProc);}
```

Examples: system libraries

DirectX/OpenGL APIs and time functions

- Typically hooked to implement cheating in on-line games

Winsock API

- Hooked to monitor network traffic

Example utilities

MadCodeHook (Madshi)

EliRT (EliCZ)

APIHijack (Wade Brainerd)

AFX Window Rootkit

Finding code to hook

- **Dependency Walker tool**
- **dumpbin for Windows**

Hot patching function calls

Redirect function calls in-line

Good targets

- Within authentication functions
- Kernel functions
 - Integrity-checking functions
 - Loader program that loads the kernel itself
 - Network functions
 - Firmware and BIOS

Detours

Microsoft's programmer-friendly approach for function patching

- G. Hunt, D. Brubaker, “Detours: Binary Interception of Win32 Functions”, 3rd USENIX Windows NT Symposium, July 1999.
- A “feature” of Windows

Detour patching

- Call hooks modify tables and can be detected by anti-virus/anti-rootkit technology
- Insert jump instruction (Detour) into function directly
 - Save original bytes of function elsewhere in a “trampoline”
 - Detour calls trampoline
- Trampoline
 - Implements 5 replaced bytes of original function
 - Implements the function you want to execute
 - jmps back to original target function plus 5

Detour details

Replace function preamble with a 5-byte unconditional jmp to trampoline

■ Before XP

- 55 push ebp
- 8bec mov ebp, esp
- Hard to hook since you must disassemble user code

■ After XP

- 8bff mov edi, edi
- 55 push ebp
- 8bec mov ebp, esp
- Easy to hook, exactly 5 bytes
- MSFT intentionally did this to make hot patches easy

Implement replaced instructions in trampoline code

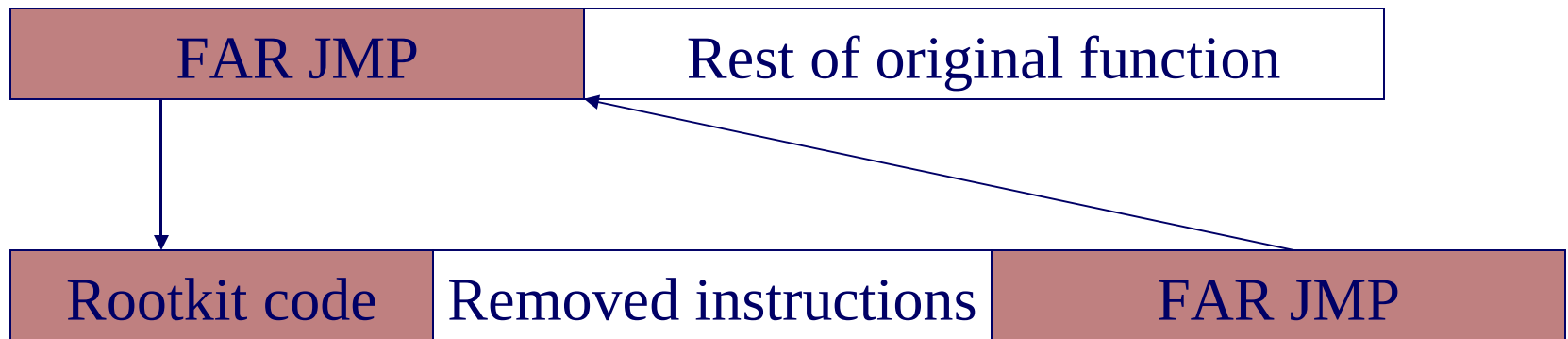
- Copy initial instructions into trampoline
- Run initial instructions then interceptor function in trampoline

Jump back to original function address at end of trampoline

Detours

Implement patch via **JMP, PUSH, RET**

- Must ensure no one else has patched the function already
- Patching addresses
 - Relative FAR JMP instruction target calculated at run-time



Detour details

More powerful than IAT hooking

- Do not have problems with binding time
- No matter how the function is called, your code will run
- Functions appearing in multiple tables are handled in one step
- Can be used for both kernel and user functions

Detours

Example: MigBot

- Detours two kernel functions: `NtDeviceIoControlFile` and `SeAccessCheck`
- Both are exported and have entries in the PE header

Hot patching function calls

Problem: Patching done at entry point

- Easy to detect if placed in well-known place
- Rootkit detection software often checks first 20 bytes of a function only
- Solution: patch deep into function

Code Caves

Where do you place trampoline?

Common way is with code caves

- Find unused code space in program's memory
- Add hack code into this space
- Inject detour into program code that jumps to the code cave
- Duplicate the overwritten code in the cave
- Return to location

Code Caves continued

The screenshot shows the OllyDbg interface for BF2.exe. The assembly window displays the following code:

```
007D6688 . 3345 F0 XOR EAX,DWORD PTR SS:[EBP-10]
007D6689 . 33F0 XOR ESI,EAX
007D668A . 8935 20238F00 MOV DWORD PTR DS:[8F2320],ESI
007D668B . 75 0A JNZ SHORT BF2.007D6697
007D668C . C705 20238F00 MOV DWORD PTR DS:[8F2320],BB40E64E
007D668D > 5E POP ESI
007D668E > C9 LEAVE
007D668F . C3 RETN
007D6690 $-FF25 00068100 JMP DWORD PTR DS:[<&MSUCR71.__security_ MSUCR71.__security_er
007D6691 $-FF25 0C068100 JMP DWORD PTR DS:[<&MSUCR71.?terminate@ MSUCR71.?terminate@y
007D6692 $-FF25 10068100 JMP DWORD PTR DS:[<&MSUCR71._controlfp MSUCR71._controlfp
007D6693 $-FF25 98078100 JMP DWORD PTR DS:[<&dbghelp.MiniDumpWri dbghelp.MiniDumpWrite
007D6694 CC INT3
007D6695 CC INT3
007D6696 CC INT3
007D6697 CC INT3
007D6698 CC INT3
007D6699 CC INT3
007D669A CC INT3
007D669B CC INT3
007D669C CC INT3
007D669D CC INT3
007D669E CC INT3
007D669F CC INT3
007D66A0 CC INT3
007D66A1 CC INT3
007D66A2 CC INT3
007D66A3 CC INT3
007D66A4 CC INT3
007D66A5 CC INT3
007D66A6 CC INT3
007D66A7 CC INT3
007D66A8 CC INT3
007D66A9 CC INT3
007D66AA CC INT3
007D66AB CC INT3
007D66AC CC INT3
007D66AD CC INT3
007D66AE CC INT3
007D66AF CC INT3
007D66B0 $ B8 F4678C00 MOV EAX,BF2.008C67F4
007D66B1 ^E9 32F8FFFF JMP <JMP.&MSUCR71._CxxFrameHandler>
```

A blue circle highlights the sequence of INT3 instructions from address 007D6694 to 007D66B0, with a label "Code cave" pointing to it.

The Registers (FPU) window shows the following state:

```
EAX 00000000
ECX 0012FFB0
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFDB000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 007D5F08 BF2.<ModuleEntryPoint>
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_MOD_NOT_FOUND (0000)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty -UNORM BDEC 01050104 00000000
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.000000000000000000000000
ST7 empty 1.000000000000000000000000
3 2 1 0 E S P U O
```

The Exception List window shows the following entry:

```
0012FFC4 7C81604F RETURN to kernel32.7C8160
0012FFC8 7C910738 ntdll.7C910738
0012FFCC FFFFFFFF
0012FFD0 7FFDB000
0012FFD4 8054A6ED
0012FFD8 0012FFC8
0012FFDC 83CF5B58
0012FFE0 FFFFFFFF End of SEH chain
0012FFE4 7C8399F3 SE handler
```

The Address List window shows the following data:

Address	Hex dump	ASCII
008DE000	00 00 00 00 34 66 7D 00	...4f).
008DE008	80 99 7D 00 70 98 7D 00	@).pc).
008DE010	00 9B 7D 00 00 9B 7D 00	ac).#c).
008DE018	00 9C 7D 00 80 9D 7D 00	.c).#d).
008DE020	C5 9D 7D 00 81 9E 7D 00	+%).u.f).
008DE028	83 9F 7D 00 B7 9F 7D 00	u.f).n.f).
008DE030	CD 9F 7D 00 E3 9F 7D 00	=f).n.f).
008DE038	E9 9F 7D 00 0F 00 7D 00	.f).#a).

The status bar at the bottom indicates: "Analysing BF2: 19905 heuristical procedures, 1721 calls to known, 28953 calls to guessed functions" and the application is "Paused".

Other rootkit functions

Disable or modify anti-virus process

Disable software updates

Disable periodic “rehooking” code

Modify network operations and services

Modify boot loader

- Have boot loader apply patches to kernel before loading

Modify on-disk kernel

- Modify boot loader to allow new kernel to pass integrity check

Registering as a driver or boot service

- Load on boot via run key in registry
- Must hide key from anti-virus after being loaded

Other rootkit functions

Using a trojan or infected file

- Replace a different .sys or executable that is run at boot-time
- Modify search path to change DLL being used
- Initialization files that specify executables to run and DLLs to load (win.ini)

Trap invocation of anti-virus

- CreateProcess with “SUSPENDED” flag
- Attach debugger and patch/inject code

Case studies

Popular rootkits:

- ILRK, URK, t0rnkit, Illogick, SK, ZK, Aquatica, Universal Root Kit (URK)

Linux RootKit (LRK)

First released in '96, constantly updated

Binary replacements with backdoor

- login – hidden “rewt” account
- rshd
- chfn – edits who has backdoor privs
- chsh
- passwd
- inetd – backdoor port 5002
- tcpd – backdoor port 5002
- sshd
- su

Linux RootKit (LRK)

Hides /dev/ptyq

Binary replacements that hide attacker

- netstat – hides ports in /dev/ptyq
- ps - /dev/ptyp
- top
- ls – /dev/ptyr, “-/" option
- du
- ifconfig
- syslogd - /dev/ptys, events and addresses
- killall
- crontab – loads processes in /dev/hda02
- pidof
- find

Linux RootKit (LRK)

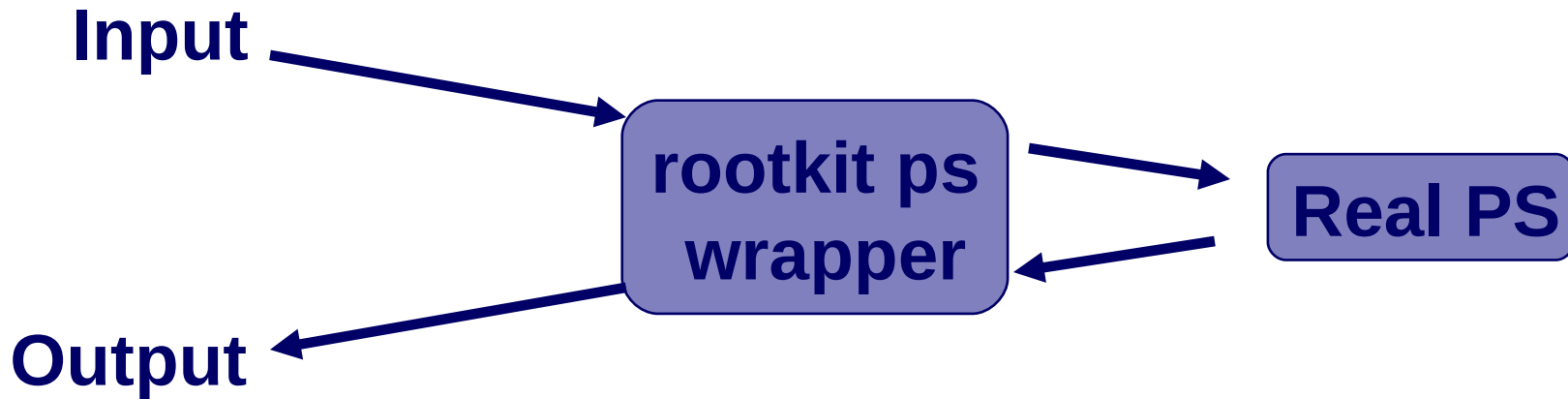
Other tools

- zap2 – fixing a_time, c_time, m_time, CRC padding
- wted – utmp, wtmp, btmp, lastlog patching
- sniffit, linsniffer – ftp and telnet
- sniffchk – see if sniffer is running
- bindshell – remote shell

Universal Rootkit (URK)

General purpose including BSD, OpenBSD, FreeBSD, Solairs, SunOS, HP-UX, AIX, IRIX...

- **Similar replacement programs**
 - **Backdoors**
 - » login, sshd, su, pidentd
 - **Privilege Escalations**
 - » ping, passwd
 - **Hiding**
 - » ps, top, find
- **Configured using urk.h and urk.conf**



Windows User-Mode RootKits

Graphical Identification and Authentication (GINA)

Manipulating Windows Login

- Winlogon process started
- Winlogon invokes GINA library code (msgina.dll)
- GINA requests credentials

FakeGINA

Sits between Winlogon and msgina.dll

Logs username/password

Exploits mechanism intended to allow other means of authentication

Configured by setting a registry key

Winlogon process

- winlogon executes
- fakegina.dll requests credentials
- fakegina.dll passes credentials to msgina.dll

Rootkit Defenses

Harden and patch system

File integrity tools to check executables and libraries

- AIDE, Osiris, Samhain, Tripwire, WFP

Anti-rootkit tools

- Identification tools – chkrootkit
- Filesystem forensics tools – The Coroner's Toolkit, Sleuth

chkrootkit

Signature based detection with kernel checks for Unix

- Shell script checks system binaries for modification.
- ifpromisc, chklastlog, chkwtmp, check_wtmp, chkproc, chkdirs, strings
- Over 40 root kit signatures

So the system is 0wnd. Now what?

- Trust nothing!
- Rebuild