Module 5

Windows security features

What defense mechanisms are included in Windows?

This module deals briefly with defense mechanisms found in the most popular `Microsoft Windows` systems.

	Windows XP	Windows 7 and 8.x
ASLR	No	Yes
DEP	From SP2	Yes
PatchGuard	In x64	In x64
Signed drivers	No	In x64
UAC	No	Yes
System firewall	Yes	Yes

`ASLR`: provides randomization for the base address space where programs and `dlls` are loaded. The randomization process makes use of `relocations` found in a program. Relocations allow programs to load at many different image bases. You can then modify the specific addresses to make them point to selected memory spaces. Even though `Windows XP` programs did have relocations, the system would always load a program at the address placed in the `PE` header of an executable (typically, `0x00400000`). The libraries that load with all programs, `kernel32` and `ntdll`, would always load at the same space. All in all, this enabled programmers to write `shellcode` with 'fixed' function addresses. The introduction of `ASLR` made exploiting vulnerabilities (for example using `buffer overflow`) harder. Still, there are some techniques that bypass this security means.

Starting with `Windows Vista`, libraries like `kernel32` or `ntdll` load in a different order every time the system boots up. `Windows 8` also sees the implementation of many ASLR enhancements, such as increasing randomized spaces and increasing entropy, which hampers or virtually thwarts brute-forcing.

`DEP`: this solution prevents the execution of code in a non-executable memory page. The non-executable areas typically include the stack and heap. Data Execution Prevention is also designed to obstruct exploitation.

`PatchGuard`: the main reason we're not discussing kernel-mode `rootkits`. This mechanism prevents modifications to the `Windows` kernel. If an alteration takes place (for example by modifying an address in the `SSDT` table), the `Blue Screen of Death` shows up and the system restarts. As it turns out however, the mechanism can be bypassed. Anyone interested in this topic can learn all about it on the Internet.

`Signed drivers`: this mechanism prevents unsigned drivers from being loaded. `Windows XP` lacks this security feature, and users must agree to installing a specific driver. In `Windows 7` and `8.x` users cannot load unsigned drivers even if they want to. This defense mechanism is enabled by default, which means that loading drivers to the kernel is much more difficult. It would require signing the driver, while every code modification entails getting another digital signature.

`UAC`: user account control. This feature was introduced in `Windows Vista`. User account control is designed to restraint and adjust programs' privileges to run a variety of actions (for example, allowing Windows files to be edited). If a program lacks adequate privileges, the system will prompt you with an elevation request. An administrator password is needed to elevate a program. `Windows Vista` users memorably detested `UAC`, making it one of the major reasons `Vista` was a flop. The mechanism displayed many slow-loading prompts, temporarily freezing the system before the elevation request appeared. `Windows 7` improved on the system, making elevation prompts much rarer and quicker to load. `UAC` itself is not a new idea (many antiviruses offered the feature long time before `Vista` did) and has been very well-developed in theory. However, practice shows it's often faulty and may be successfully bypassed (simply type `UAC bypass` into a search engine to find out more). Most `malware` today has the `UAC bypass` field in configuration options.

The crux of the problem is that a user needs to agree to elevating the privilege level of a swarm of harmless applications that need administrative access to run. `Malware` can bypass `UAC` in the background without needing a user to click anything.

`System firewall`: from `Windows XP` onwards the systems are equipped with the `Security Center`, a feature that includes an inbuilt system `firewall`. Unfortunately, the `firewall` is not a hundred percent attack-proof solution. The next chapters present ways to bypass `Windows Firewall` and many other similar products. Firewalls on the whole are based on trusted application rules. The snag is that it’s easy for an attacker to discover which applications are trusted, and most malware can do it to inject malicious code and connect to the Internet.

Another downside is that the system `firewall` rules don’t include a specific rule for the port number with which applications connect. For browsers, ports `80` and `443` should be open. If they were, the `firewall` could alert you if code was injected to the browser, detecting traffic on a non-standard port. In addition, the `firewall` fails to recognize if a trusted file has been modified. An attacker could easily replace a browser file with malicious code. Again, the problem is similar to the one `UAC` had. The user puts in a lot of work to click harmless application in. Unfortunately, at the same time the real threats can bypass defenses easily.

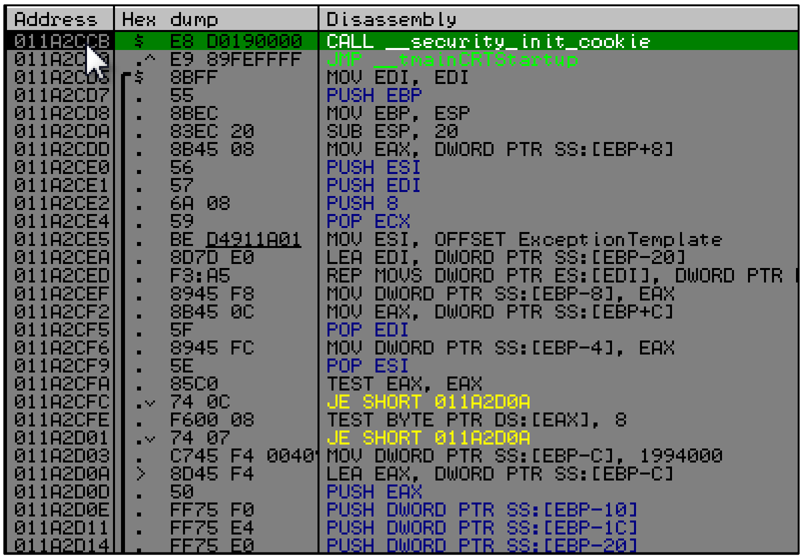
Practice: video module transcript

Welcome to the fifth module of the training. In this module we'll discuss the Windows security features. One of the flag Windows security features is ASLR, which stands for Address Space Layout Randomization. This technology wasn't implemented in the standard version of Windows XP, but it's present in Windows Vista and, of course Windows 7 and Windows 8.

Another security feature which we'll discuss is DEP, the Data Execution Prevention technology, which is present in Windows XP from Service Pack 2 on, and exists in Windows 7 and Windows 8 from the far the beginning. When talking about security features, we can't omit the Patchguard technology, which is responsible for kernel protection and is available in all 64-bit versions of Windows. The requirement of digitally signed drivers is no longer valid when it comes to Windows XP. In Windows 7 and Windows 8 it's present only in the 64-bit version.

Another Microsoft technology, User Account Control, or UAC for short, is responsible for restricting user permissions until explicitly confirmed and was introduced with the Windows Vista system. Apart from the aforementioned mechanisms, there is also a standard, built-in system firewall, which is present in all versions of Windows starting from XP. Now let's briefly discuss the listed mechanisms. What is ASLR?

It's a mechanism which loads programs and dll libraries to a somewhat new random memory address. Due to this, we can't use "fixed" function addresses in systems with ASLR implementation, but have to get them dynamically. Let's check what the ASLR mechanism looks like in practice.



```
Address   Hex dump          Disassembly
011A2CCB   $  E8 D0190000   CALL __security_init_cookie
011A2CD0  .^  E9 89FEFFFF   JMP __tmainCRTStartup
011A2CD5  ┌$  8BFF          MOV EDI, EDI
011A2CD7  │.  55            PUSH EBP
011A2CD8  │.  8BEC          MOV EBP, ESP
011A2CDA  │.  83EC 20       SUB ESP, 20
011A2CDD  │.  8B45 08       MOV EAX, DWORD PTR SS:[EBP+8]
011A2CE0  │.  56            PUSH ESI
011A2CE1  │.  57            PUSH EDI
011A2CE2  │.  6A 08         PUSH 8
011A2CE4  │.  59            POP ECX
011A2CE5  │.  BE D4911A01   MOV ESI, OFFSET ExceptionTemplate
011A2CEA  │.  8D7D E0       LEA EDI, DWORD PTR SS:[EBP-20]
011A2CED  │.  F3:A5         REP MOVS DWORD PTR ES:[EDI], DWORD PTR
011A2CEF  │.  8945 F8       MOV DWORD PTR SS:[EBP-8], EAX
011A2CF2  │.  8B45 0C       MOV EAX, DWORD PTR SS:[EBP+C]
011A2CF5  │.  5F            POP EDI
011A2CF6  │.  8945 FC       MOV DWORD PTR SS:[EBP-4], EAX
011A2CF9  │.  5E            POP ESI
011A2CFA  │.  85C0          TEST EAX, EAX
011A2CFC  │.v 74 0C         JE SHORT 011A2D0A
011A2CFE  │.  F600 08       TEST BYTE PTR DS:[EAX], 8
011A2D01  │.v 74 07         JE SHORT 011A2D0A
011A2D03  │.  C745 F4 0040  MOV DWORD PTR SS:[EBP-C], 1994000
011A2D0A  │>  8D45 F4       LEA EAX, DWORD PTR SS:[EBP-C]
011A2D0D  │.  50            PUSH EAX
011A2D0E  │.  FF75 F0       PUSH DWORD PTR SS:[EBP-10]
011A2D11  │.  FF75 E4       PUSH DWORD PTR SS:[EBP-1C]
011A2D14  │.  FF75 E0       PUSH DWORD PTR SS:[EBP-20]
```

Here we have the rcmd program, which we remember from the previous module. Let's open it in Olly debugger. We can see that the code starts in 011A2CCB. Let's have a look at the loaded modules. Rcmd was loaded to the address 011A0000. Let's close the program and check where it will load at the next launch. As we can see, the address is different. The module list address is also different. Previously it was 011A0000, now it's 013B0000. As we can see, the ASLR mechanism is active. Now let's move on to discuss the remaining security features.

Another mechanism is DEP, a mechanism which forbids the execution of code which has no right to be executed. Windows XP didn't worry whether the given memory page has execution rights – it simply executed the code. The task of this mechanism is, first of all, to protect against exploits, which take advantage of the buffer overrun, because in this case the code is usually present on the stack and, as we know, stacks have no execution rights. We have to allocate memory using the PAGE_EXECUTE_READWRITE flag, so that the injected code can be executed. More about this mechanism and using various attack techniques in Windows systems can be found in the training on the Security of Windows Applications. Surely, it's good to get to know this item if we're interested, for instance, in the subject of bypassing DEP or ASLR. But as for now, let's move on to discuss the next technology.

Patchguard is one of the mechanisms present only in 64-bit systems. It guarantees that neither the kernel code nor any of the kernel structures are modified. If such a situation happens, we immediately see the blue screen known from the previous versions of Windows and the system is halted. In Windows XP the user could agree or disagree to load a driver which has no digital signature. In Windows 7 and 8 the user has no such possibility. If there are drivers without a valid digital signature, the system prevents them from being loaded.

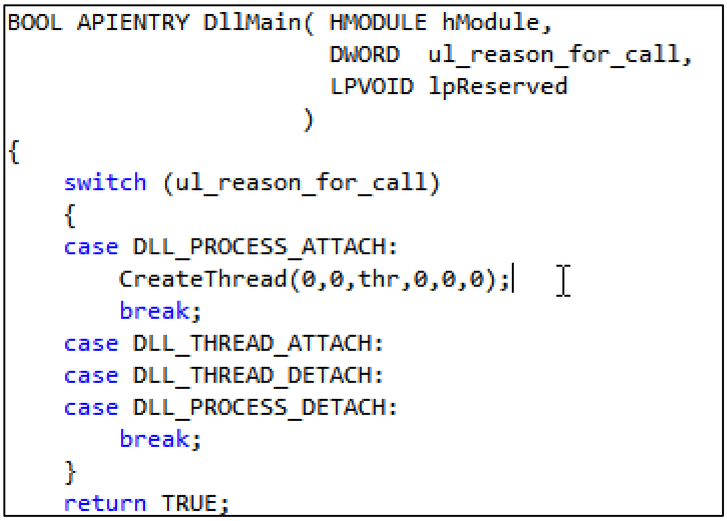
A digital signature is a certain value which is a hash function, such as for instance, md5, which joins the publisher identity with the document contents, in this case with the driver code. Any code change results in the fact that the digital signature is no longer correct. User Account Control is a mechanism which, in a way, limits the user freedom, but provides greater security by limiting access to certain system areas. We can't modify these areas without admin rights.

Crucially, if you want to create a program which has to work without admin rights, don't place registry keys in the LOCAL_MACHINE tree because adding

a key there requires the elevation of privileges. We can place an entry in the registry without the admin's approval, for instance in the CURRENT_USER tree. Similarly, never copy files to the C:\Windows directory. In order to place anything there, we need the admin's approval.
Instead, we can freely copy and modify files within the user profile. Thanks to that, our applications will be more reliable and won't raise any suspicions.

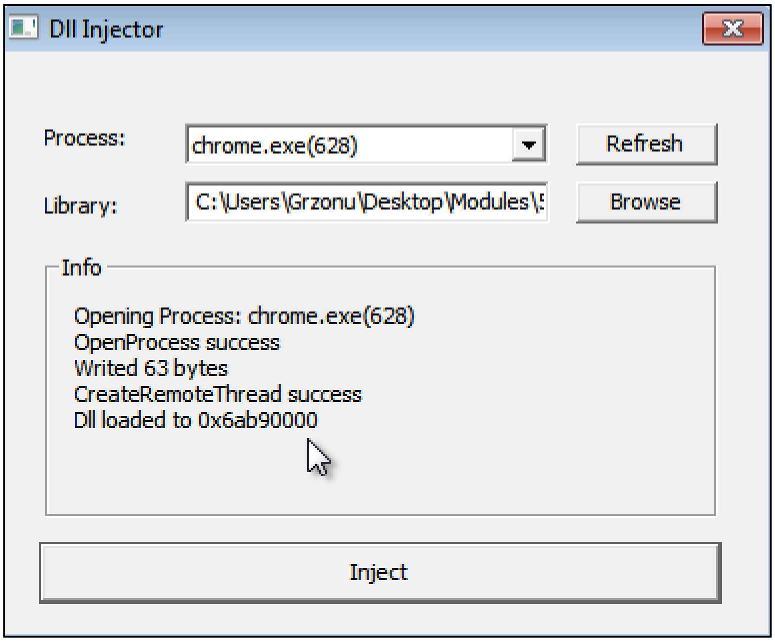
A built-in firewall is a mechanism present in the system since Windows XP Service Pack 2. It's a typical network firewall the operation of which is based on a set of rules. Based on these rules, the firewall decides whether a given application can communicate with the network. However, as we'll demonstrate, we can cheat it in a very simple way.

Recall our rcmd program from one of the previous modules. The firewall will probably block it. We enter the dir command. After 10 seconds at most, we should receive an answer. If the firewall blocks the communication, the program can neither get a command, nor send an answer. As we can see, we constantly refresh the page, but nothing happens. Now we'll close our program and use a simple trick. We place the code of our program in a dll library which we inject, for instance, to the browser process, which, understandably, has permission for network communication.



```
BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
        CreateThread(0,0,thr,0,0,0);
        break;
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
```

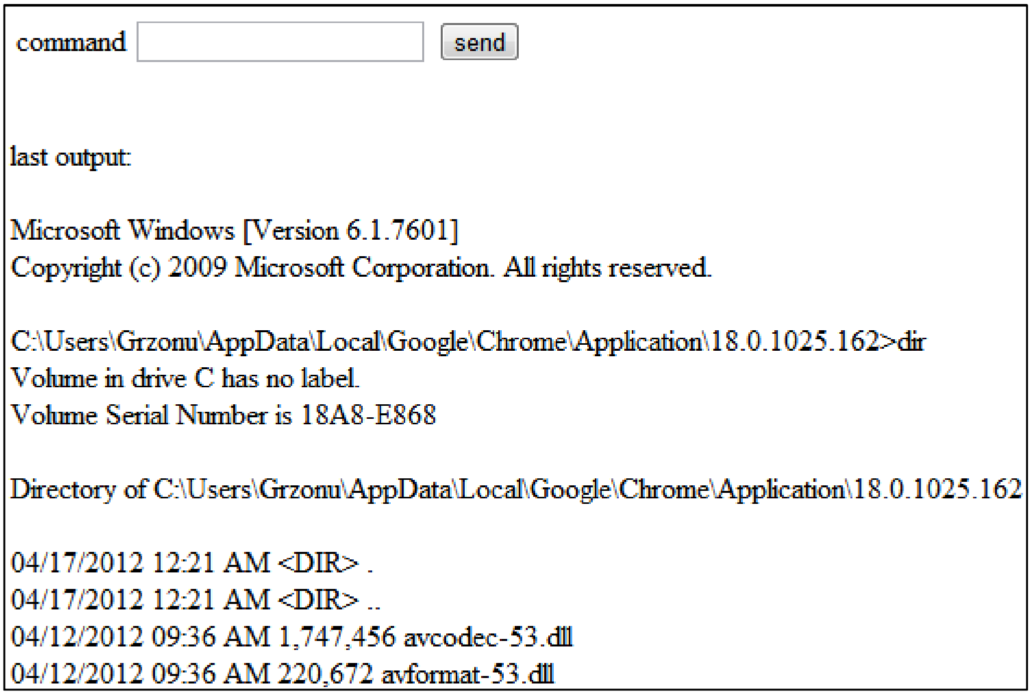
Here we have exactly the same code we saw in the earlier module, the difference being that the entire code which was previously present in the main function is now present in the thread function. However, in the main function we create only the thread we've mentioned. The moment we add the dll library to the process, the thread is started. We already have a compiled version of this library. We can see it here.



```
chrome.exe(628)
C:\Users\Grzonu\Desktop\Modules\
```

```
Opening Process: chrome.exe(628)
OpenProcess success
Writed 63 bytes
CreateRemoteThread success
Dll loaded to 0x6ab90000
```

Now we inject it to the Chrome process, that being the Internet browser which is most likely to have access to the Internet. We refresh the page and as we can see, the program included in the dll library managed to download and execute the command. We can also see that the default program directory is the Chrome browser directory.



```
last output:

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Grzonu\AppData\Local\Google\Chrome\Application\18.0.1025.162>dir
Volume in drive C has no label.
Volume Serial Number is 18A8-E868

Directory of C:\Users\Grzonu\AppData\Local\Google\Chrome\Application\18.0.1025.162

04/17/2012 12:21 AM <DIR> .
04/17/2012 12:21 AM <DIR> ..
04/12/2012 09:36 AM 1,747,456 avcodec-53.dll
04/12/2012 09:36 AM 220,672 avformat-53.dll
```

Now we can try to execute a command, for instance to ping Google.com. We have to wait a while for the response. Unfortunately, we've received the message General failure, the ping process was most probably blocked by the firewall because it's not present on the white list. Again, we perform the ping command and wait a while. We've received an answer, accompanied by the remaining part of the previous result. However, the dir command executed successfully. We can see all files and directories from the Chrome directory.

The built-in system firewall works because it blocked the rcmd application. But, as we've just seen, we can very easily bypass this mechanism - putting the code in a dll library takes up no more than 10 minutes. Then, we can inject it to any browser which probably has rights to access the Internet. In the next modules we'll learn how to get a path to the default browser, how to create its new process and how to inject a code into it.

Let's summarise what we've learnt so far. From versions XP to 8, Windows has more and more security features. Some of them are only a nuisance for a potential aggressor, such as ASLR or DEP and it's possible to bypass them (we can learn more about it from the training on Windows Applications Security). In practice, performing an attack is still possible.

However, let's not forget that thanks to the PatchGuard technology it's impossible to create a kernel-mode rootkit. The progress in the branch of security is definitely noticeable. Another thing which improved a lot is the system stability, because only trusted (that is digitally signed) drivers are loaded to the system. It's not possible to load drivers of unknown origin which could damage the system.

The UAC technology also contributed to our safety. Similarly to the unix systems, the user doesn't work with admin rights as a default. In order to perform an action which is potentially harmful to the system, we require the admin's approval. Thanks for your attention and please go to the next module. See you there.