DISCLAIMER

This is a little Disclaimer for if you haven't read the one on our site. The tools and tutorials KD-Team develops and publishes are only meant for educational purpose. WE DO NOT encourage the use of these tools and tutorials for malicious purpose. We learned a lot during the development of them so we hope you also learn and don't just use it without any brains. We take completely NO responsibility for any damage caused by them nor are we or our isp responsible for what you do with them.
Greetz: KD-Team

http://www.kd-team.com

Timing Rootkits

Using RDTSC and RDPMC to defeat rootkits

Now that you feel safe…

Let's defeat this technique by using WRMSR

Index

A word of thanks

First of all I think the best begin of this paper is to start with a word of thanks to all the people that helped me in a way in finding if this new approach to detect a rootkit was plausible or not.

I want to thank them all for their time and effort in answering my questions.
In no particular order:

n0limit
parad0x
Animal
ScriptGod
Bob K (lovepump)
tweakz20

I would also like to thank the following people specially.

wiNGCom for being kind enough to provide a special purpose rootkit
tibbar for defeating what seemed to be a full proof idea.

I also want to thank all the testers that helped me out with running the POC on theire computers.

Sorry if I forgot any one. If so thanks to who ever I might of have forgotten. Without every one named and unnamed it would of have been impossible to bring this along.

Introduction

Welcome all to another paper from KD-Team. This time the paper is about a new way of looking at how you can detect the presence of a rootkit on a system mostly targeted at the windows operating system. Like you have probably noticed on our site we have posted alternate ways of detecting a rootkit. Both of the ways where just a simple brute force on a function that hadn't been hooked by the rootkit. You can find them both in the tool section of our site named "Detect Con" and "Detect Proc". Like you have already guessed, these methods are not full proof.
So we started to think about a totally different approach to detect a rootkit. We started with answering the following two questions:

1) Does a rootkit have impact on system performance?
2) Does a rootkit slow down a computer?

When trying to find the answers to those questions we where thinking in a user way. Would a user notice if his computer got slower? If so how much slower should it get? Would a user notice if his computer suddenly had bad performance? If so how bad should the performance decrease?
So how do you measure that? The answer was using the instructions and possibilities provided by the x86 instruction set. The Intel manuals where very helpful along the road. The only bad thing about all this is, it started as a unbeatable method cause the instructions provided by intel are not hookable unless you remake the instruction set on the chip. So what we need to do is start the measurement before function exection then execute the function and stop the measuring when the function ends. The registers used though are subtle to manipulation like tibbar nicely pointed out, more about this in the WRMSR part.
Although tibbar has found a workaround for this , it does not mean it renders the method useless. This method combined with the work of rootkit revealers and other detectors, can be a very though pain in the ass for most rootkit developers.

So now that we have had the introduction and the basics of the detection explained lets see into it in more detail. We will discuss the time option for rootkit detection , the performance option and finally how to defeat both of these methods.

I know what you all think now, isn't it odd to release a paper with a new detection method and at the same time explain how to defeat it? Well since the discovery of how to defeat this method surged while investigating the method we think it's best to publish them both. At least now if some one implements this method in a rootkit detector he will not have a vain sense of security.

For the performed tests we used a modified version of the code which link is in the references part of this paper. Reason we haven't coded a program for this is simply cause we don't have got the time.

RDTSC for detection

What is RDTSC?

Lets quote the intel manual: “RDTSC—Read Time-Stamp Counter”. So this is a instruction to retrieve the value of the time stamp counter. Now what exactly is the time stamp counter? Lets quote intel again “The processor monotonically increments the time-stamp counter MSR every clock cycle and resets it to 0 whenever the processor is reset”. So put in other words it counts the cpu cycles. Now this sounds interesting if we could measure the clock cycles a function needs to execute right?

How can we use it for detection?

As stated before you must start the measurement before function execution and stop it afterwards. There are a few things to take into account when coding this. The windows operating system is a pre-emptive system. Meaning that it shares the time between processes and threads. So you got to make sure the function you are measuring won’t be interrupted by windows. Luckily windows provides a excellent way to do that named SetPriorityClass(). With that function and using as second parameter REALTIME_PRIORITY_CLASS. You can make sure the execution of your measuring doesn’t get interrupted.
A practical point to take into consideration is the following. The execution time (clock cycle) a function needs depends on the work a function must perform. So you must make sure that you use hooked functions that are not to dependant of the environment. This means avoid functions interacting with your hard drive, avoid functions that depend heavily on the amount of data they have to go through. For example we could use the following function to perform the tests OpenProcess(). Why? Cause this function is not disk dependant, and the amount of data it has to go through is reasonable constant if you perform the tests with the same amount of processes running. Just don’t forget the paging of memory by windows so it could in a rare occasion happen that the memory is paged out, usually you should be alright though.
Although a light alteration in the process would not severely influence on detecting a rootkit it would influence on the results displayed. Here are some results we tested with it’s always displayed in a before and after rootkit way.

INTEL BEFORE Clock	INTEL AFTER Clock	AMD BEFORE Clock	AMD AFTER Clock
50216	63824	28904	58835
9516	12724	3546	6333
7440	9516	2501	3946
7108	9052	2358	3344
7204	9032	2415	3168
7108	9040	2294	3312
7112	9024	2272	3168
7192	9020	2272	3140
7204	9012	2399	3244
7200	9036	2272	3168

7176	9056	2272	3140
7168	9028	2415	3140
7168	9044	2272	3282
7168	9016	2272	3168
7168	9032	2272	3140
7188	9032	2349	3229
7172	9024	2272	3168
7176	9052	2272	3140
7148	9016	2325	3140
7160	9060	2272	3209

The above two tests where performed on different machines, with different processor and different amount of processes. As you can see the difference between a machine without a rootkit and a machine with a rootkit is obvious. We have performed more tests and the amount of extra cycle with a hooked function of OpenProcess() stays +/- around the 1000. This is of course hook specific. So this clearly shows that the hooking of a function can be detected. A important thing to remember is that dkom is not detectable with this measuring.

RDPMC for detection

What is RDPMC?

Lets quote the intel manual: “RDPMC—Read Performance-Monitoring Counters”. So this is a instruction to retrieve the value of the performance counters. Now what exactly are the performance counters? Lets quote intel again “The performance-monitoring counters are event counters that can be programmed to count events such as the number of instructions decoded, number of interrupts received, or number of cache loads”. So this means we can actually measure almost everything the processor does when executing a function. This is highly interesting cause it can gives us extra guarantee of the presence of a rootkit.

How can we use it for detection

Most off the conditions explained in the same section on the RDTSC command apply to this command to. It is important to remind you that the RDPMC is a help to detect a rootkit we do not recommend to use it stand alone, instead use it in combination with the RDSTC detection method. This caused by the minimalistic performance changes that occur when using a rootkit. Below you can see the result of one of the multiple tests we conducted. Like you can see the changes are minimalistic to none.

BEFORE			AFTER		
Uops	CodeMiss	DataMiss	Uops	CodeMiss	DataMiss
662	21	10	662	23	11
547	4	1	547	5	1
547	4	1	547	5	1
547	4	1	547	5	1
547	4	1	547	5	1
547	4	1	547	5	1
547	3	1	547	5	1
547	4	1	547	5	1
547	3	1	547	5	1
547	3	1	547	5	1
547	3	1	547	5	1
547	4	1	547	5	1
547	3	1	547	5	1
547	3	1	547	5	1
547	4	1	547	5	1
547	3	1	547	5	1
547	3	1	547	5	1
547	3	1	547	5	1
547	4	1	547	5	1
547	3	1	547	5	1

Like you can see it’s a difference but not to big. So it’s best to use this together with RDSTC.

WRMSR makes sure that rootkits continue to reign

Well now we are at the part that totally destroyed the idea of being full proof cause of the idea that processor instructions can not be hooked. We will explain the main idea on how to defeat it. A more technical view of the defeating method will be probably be published by tibbar himself.

Lets start with that it indeed is not possible to hook a processor instruction as far as we know. We DID however forget that we are after all measuring a hooked function. So everything what that function performs will be performed anyways.

So this consists that the rootkit all it needs to do is:

- measure the number of cycles used by original function
- calculate how much extra cycles the hook it is going to take
- use RDMSR to find the current timestamp
- decrement that value with the extra cycles used by the rootkit and the following WRMSR call
- use WRMSR to write that value to the correct registers

After performing the above operations when the measuring program reads the registers to know how long it took the function to execute it will have the same cycles a non-hooked function does.
This of course requires a great amount off effort for the rootkit writer to implement it correct and make sure he is exactly correct or at least in a reasonable range.

Like almost always the defeat of security is easier and shorter then the security measure itself.

Conclusion

We can safely conclude that detecting a rootkit with time and performance monitoring is possible. We can also conclude that this method can not really be used as a rootkit detection in the real sense of the word. Since the differences between processors is to big to make a baseline. So this should be used as a check of the system virginity. So when a system is clean you can take a snapshot of the system and save the results or print them out. Then whenever the need arises to be sure the system has not been compromised you can perform a measurement and compare the results.
So this method is good to make sure system integrity is harder to circumvent, seeing the fact that a rootkit needs to fake a lot of data.

Off course we must not forget that it indeed IS possible to fake the data. So when implementing this method you must make sure you implement the most amount of data as possible. This way making it a real pain in the ass for most rootkit writers. Also make sure this method is not standalone but uses other methods like registry hive reading etc.

Well we have come to the end of this paper. Hope you all enjoyed it and that it was a new fresh way to look at how you can detect rootkits.

References

Websites:

http://www.rootkit.com – excellent reference for every one wanting to know more about rootkits.
http://www.msdn.microsoft.com – always nice to have around when playing with windows
http://www.agner.org/assem/#testp – reference code for any one needing a example on how to work with RDTSC and RDPMC. We used a slightly modified version of this to perform the tests. To busy at the moment to release a special purpose application for this kind of detection. Maybe we will in the future.
http://www.intel.com – off course needed to retrieve the manuals.

Books:

- Rootkits, *subverting the windows kernel* by Greg Hoglund and James Butler
- Microsoft Windows Internals by Mark E. Russinovich and David A. Solomon