

System Virginitv Verifier

**Defining the Roadmap for Malware Detection
on Windows System**

Joanna Rutkowska

`http://invisiblethings.org`

*Hack In The Box Security Conference,
September 28th -29th 2005,
Kuala Lumpur, Malaysia.*

Menu for today...

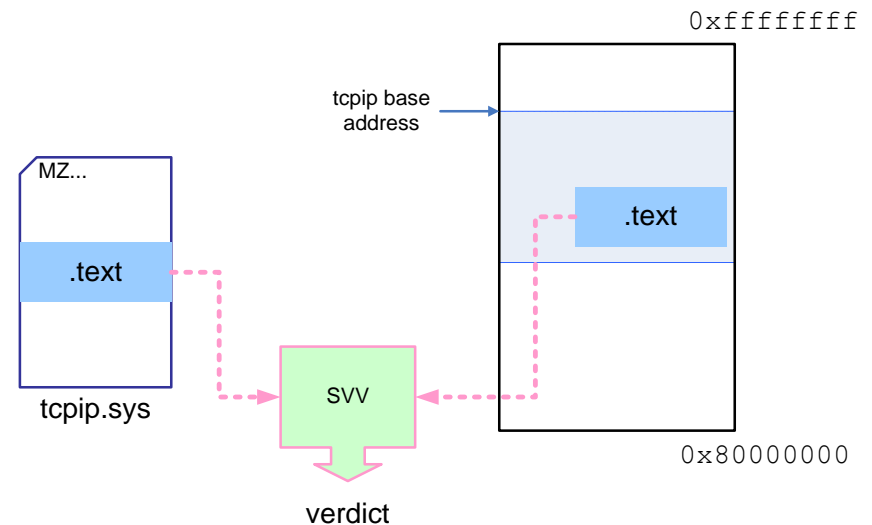
- ⊕ System Virginty Verifier (SVV)
 - ⊕ New tool for Windows 2000/XP/2003
 - ⊕ Verifies in-memory code sections integrity
 - ⊕ Allows for disinfection (malware removal)
 - ⊕ Care taken not to generate false positives
- ⊕ Open Methodology for Compromise Detection (OMCD)
 - ⊕ Why do we need OMCD?
 - ⊕ OMCD outline
- ⊕ Few words about implementation specific attacks

System Compromise Detection

- ⊕ Check Integrity of Important OS elements
 - ⊕ System files integrity
 - ⊕ List of autorun programs
 - ⊕ Code sections integrity
 - ⊕ IAT/EAT, SDT, IRP tables
 - ⊕ NDIS pointers
 - ⊕ ...
- ⊕ Cross view based approaches
 - ⊕ Detect hidden files, registry keys, processes
- ⊕ Signature based approaches
 - ⊕ Scan for known rootkit/backdoor engines

System Virginity Verifier Idea

- ⊕ Code sections are read-only in all modern OSes
- ⊕ Program *should not* modify their code!
- ⊕ Idea: check if code sections of important system DLLs and system drivers (kernel modules) are the same in memory and in the corresponding PE files on disk
 - ⊕ Don't forget about relocations!
 - ⊕ Skip .idata
 - ⊕ etc...



False Positives

- ⊕ Important in commercial implementations (AV products)
- ⊕ Rootkit technology” is widely used in many innocent tools ;)
 - ⊕ Personal Firewalls/Behavior blocking systems
 - ⊕ Monitoring tools like Virus Monitors
 - ⊕ System “tracing” tools (like Filemon, DbgView, etc...)
 - ⊕ ...
- ⊕ How to deal with those false positives?
- ⊕ But first... the self –modifying kernel problem...

Self-modifying kernel code

- ⊕ `.text` of two core kernel components does not match the corresponding files on the disk!
- ⊕ Reason: code self modifications to replace some hardware related functions so they match the actual hardware (like sync functions in SMP vs non SMP systems)
- ⊕ Those two components are:
 - ⊕ `ntoskrnl.exe` ← kernel core
 - ⊕ `HAL.DLL` ← Hardware access functions
- ⊕ How to deal with those discrepancies?

Filtering kernel self-modifications

- ⊕ Observation: there are only a bunch of functions which are altered (10-20 in `ntoskrnl.exe` and `hal.dll`).
- ⊕ In HAL: all modifications are `00` → `XX` (i.e. original function body is NULL in the file)
- ⊕ In NTOSKRNL:
 - ⊕ Most are one-byte modifications (`NOP` → `XX`)
 - ⊕ The rest (5 or so) are pretty stable among different kernels (from 2000, through XP, to 2003)


DEMO

- ⊕ Looking at the kernel self-modifications using the Kernel Debugger:
 - ⊕ Inside NTOSKRNL.EXE
 - ⊕ Inside HAL.DLL

False Positives

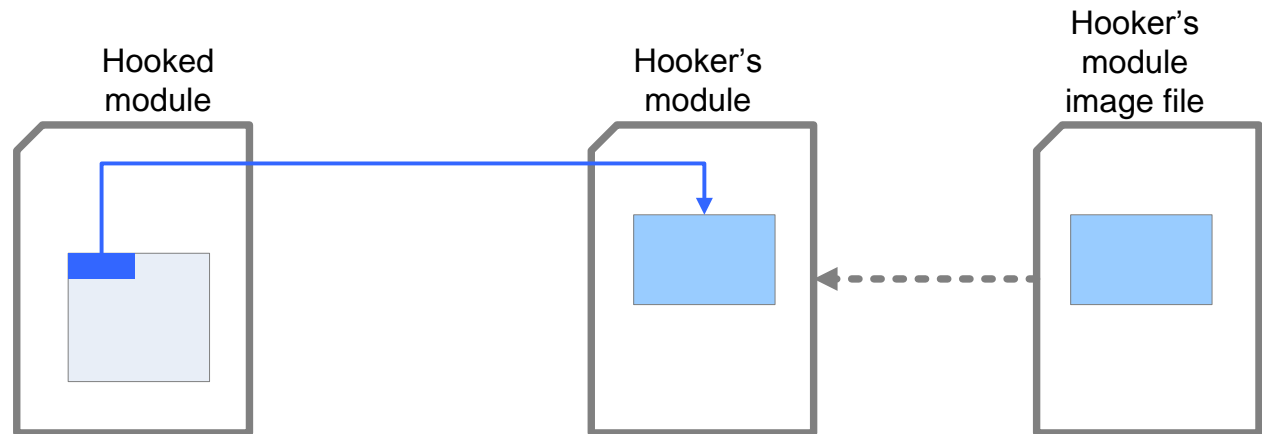
- ⊕ Now how to deal with “innocent” hooking?
- ⊕ To solve this problem we should describe the several types of hooking which we can observe in the nature...

Missing vs. hidden files

- ⊕ Some definitions first:
- ⊕ File is *missing* when `CreateFile()` fails,
- ⊕ File or directory is *hidden* when `FindFirstFile()`/
`FindNextFile()` do not return that file in directory listing,
- ⊕ If a directory is *hidden* all its files and subdirectories are also *hidden*.
- ⊕ Note:
 - ⊕ missing files don't need to be hidden (e.g. `REGSYS.SYS`)
 - ⊕ hidden files usually are *not* missing
(e.g. `c:\r00t3d\hxdef.sys`)
 Dir is hidden, but present

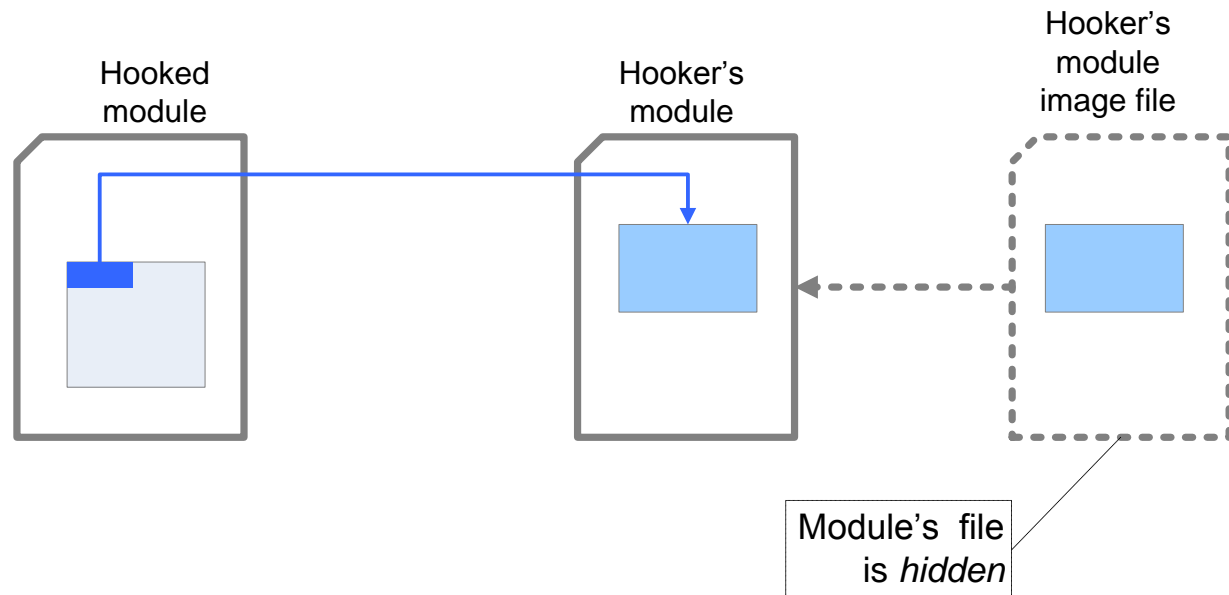
Typical Innocent hooking

- ⊕ New code looks like a “JMP” (not obfuscated)
- ⊕ Redirection to code section inside valid module
- ⊕ Hooker's module file present and not hidden (rootkit/backdoor needs to hide its module)
- ⊕ Verdict: probably personal firewall or so...



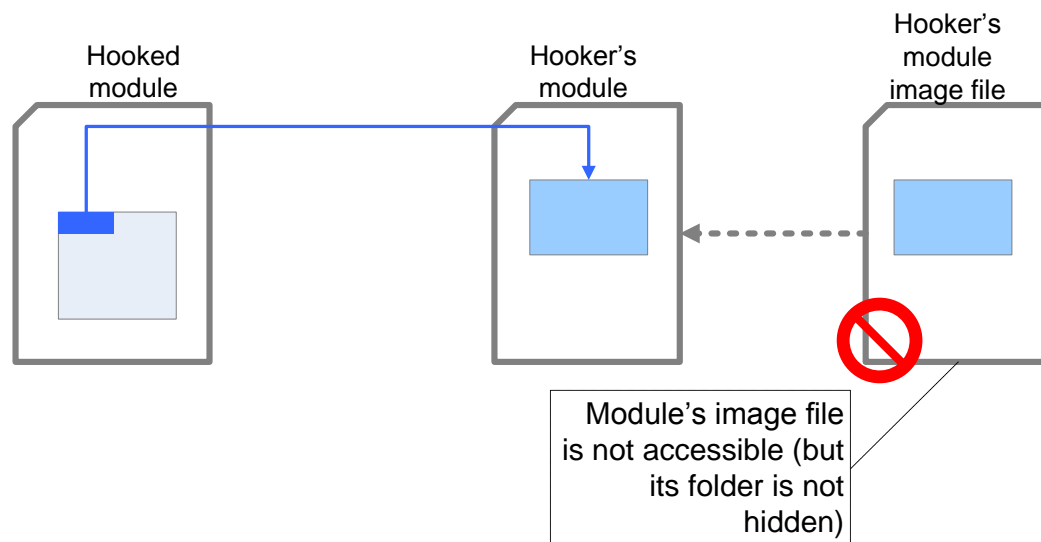
Malicious hooking I

- ⊕ Hooker's module file is *hidden*. It may be openable by *CreateFile()* but still it is probably a rootkit. Innocent programs do not hide their module files.



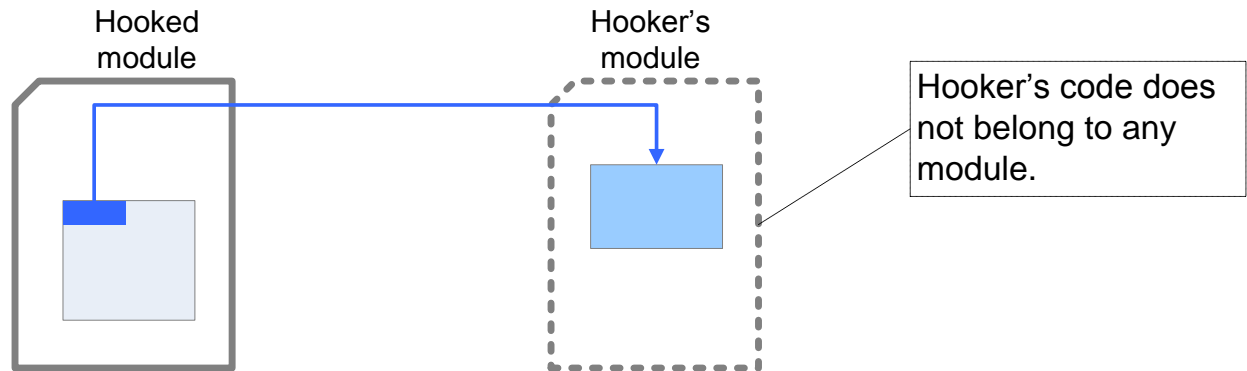
“Debugger” – like hooking

- ⊕ Module image file is missing (but is not located in a hidden directory)
- ⊕ Typical for tools which extract their driver from resource section and place it *temporarily* in the system driver directory (DbgView, RegMon, etc...)



Malicious hooking II

- ⊕ Redirection to code which does not belong to any module.
- ⊕ Typically this means that either:
 - ⊕ Code was placed in a manually allocated block of memory
 - ⊕ Owning module's descriptor was intentionally tampered
- ⊕ This is very suspicious. Innocent program should not do this.



Malicious hooking III

- ⊕ Rootkit uses obfuscated code to redirect execution → SVV cannot determine the target address.
- ⊕ It's not a problem, because we expect such behavior (i.e. using obfuscated JMP-ing code) in malicious software only
- ⊕ So, if altered code cannot be decoded into a JMP then report it as an infection.

Detecting SDT modifications

- ⊕ `KiServiceTable` is part of `.text`
- ⊕ If modifications in inside `KiServiceTable` then it is the target address itself (not JMP-ing code)
- ⊕ The actual SDT can be relocated (like when Kaspersky AV is used)
- ⊕ Thus `KiServiceTable` is not always pointed by `[KeServiceDescriptorTbl[0]+0]`
- ⊕ Thus a robust method for finding the address of `KiServiceTable` is needed – see 90210's article on rootkit.com for the smart solution

SVV Verdicts



100% virgin (not expected to occur in the wild)



Seems ok



Innocent hooking detected



“Debugger” like hooking detected



Very suspected but may be a false positive



Compromised!

DEMO

- ⊕ SVV on a clean system
- ⊕ SVV vs. some popular rootkits
- ⊕ SVV disinfection demo
- ⊕ SVV vs. system tracing tools

SVV 1.0 Public Release

- ⊕ SVV 1.0 check only `.text` sections
- ⊕ Should catch all code-alternating rootkits
- ⊕ Some backdoors (like EEYE BootRoot) can modify other code sections as well (like `PAGE*` inside `NDIS.SYS`)
- ✗ Checking of other sections will be added soon (next week or so)
- ✗ Currently one known bug (discovered few days ago): BSOD on terminal services when more then one user are logged in (should be patched soon too)

SVV future

- ⊕ SVV 1.0 takes care only about CODE VIRGINITY
- ⊕ It is only the very first step in building integrity-based rootkit detector
- ⊕ Next versions should include:
 - ⊕ EAT/IAT, SSDT, IDT and IRP tables verification
 - ⊕ NDIS important data pointers verification
 - ⊕ More ...
- ⊕ So, we see the need to **define all the vital OS components** which should be verified...
- ⊕ ... this is what OMCD project (see later) is about.

Related work (integrity checkers)

- ⊕ VICE by Jamie Butler
 - ⊕ Tries to detect hook, not code modifications
 - ⊕ Relatively easy to bypass when rootkit uses non standard (polymorphic) JMP-ing code
 - ⊕ Checks also EAT, SDT and IRP (good!)
 - ⊕ Uses API to read other processes memory (bad!)

Related work (integrity checkers)

- ⊕ SDT restore by SK Chong
 - ⊕ SDT integrity checking
 - ⊕ Relocated SDT detection (good!)
 - ⊕ BTW, would be nice if it also checked IDT table

Related work (integrity checkers)

- ⊕ CaptainHook by Chris Carr (bugcheck.org)
 - ⊕ Not public, sent to me by Chris last week
 - ⊕ Same idea as SVV
 - ⊕ However:
 - ◆ Does not take care about kernel self-modifications
 - ◆ Does not detect innocent hooking

Related work (integrity checkers)

- ⊕ IceSword by pjf USTC
 - ⊕ SSDT
 - ⊕ Win32 Message hooks
 - ⊕ ASEPS (Some auto-starting programs, BHO, SPI)
 - ⊕ Hidden Processes and Services enumeration (hidden not marked – not like in X-DIFF)
 - ⊕ File system and registry explorer (including *hidden objects*)
- ⊕ It is only enumeration tool – does not give any verdict if the system is compromised or not.

System Integrity Verifiers

	SVV 1.0	SDT Restore 0.2	VICE 2.0	IceSword 1.12	Kernel Debugger (+ scripts)
Code verification	YES		Only “JMP” detection		YES
SDT		YES	YES	YES	YES
IDT				?	YES
IAT/EAT			YES (lots of false positives)		YES
IRP dispatch Tables			YES		YES

Open Methodology for Compromise Detection

OMCD

<http://www.isecom.org/omcd/>



OMCD

- ⊕ New project just started by the author with the help of the community
- ⊕ Currently focuses only Windows systems
- ⊕ Hosted at ISECOM:
`http://www.isecom.org/omcd/`

OMCD goals

- ⊕ Create list of all important OS elements which should be verified to find system compromise
- ⊕ Not only a list but also a road map (methodology) of how those parts should be verified
- ⊕ Intended mainly for developers of compromise detection tools
- ⊕ Could be useful as a reference for comparing different tools on the market (“*our tool implements OMCD sections A, B, C & D!*”)
- ⊕ Author *believes* that there are only a **finite number of ways that the rootkits and network backdoors can be implemented** in the OS...

OMCD outline

- A. Preliminaries
- B. File system verification
- C. Registry Verification
- D. Application Integrity Verification
- E. Usermode-level memory verification**
- F. Kernel-mode level memory verification**
- G. Network activity analysis

Section E (Usermode-level memory) overview

- ⊕ Discovering suspected (not hidden) processes and threads
- ⊕ Code sections verification
- ⊕ IAT/EAT tables verification
- ⊕ Considerations for memory access techniques
 - ⊕ Using API for accessing other processes memory
 - ⊕ Direct virtual memory access (DVMA)
- ⊕ ??? [your suggestions here]

Section F (kernel memory) overview

- ⊕ Considerations for memory access technique
- ⊕ Code sections verification
- ⊕ Dispatch tables and function pointers verification
 - ⊕ Service Dispatch Table (SDT) verification
 - ⊕ Interrupt Dispatch Table (IDT) verification
 - ⊕ IRP dispatch tables hook discovery
 - ⊕ NDIS function pointers hook discovery (Network DKOM rootkits)
 - ⊕ Configuration Manager function pointers hook discovery (Registry DKOM rootkits)
- ⊕ System Registers verification
 - ⊕ IA-32 processors (SYSENTER, Debug registers)
 - ⊕ Other processors?
- ⊕ Discovering suspected attached devices (filer based rootkits)
- ⊕ Discovering suspected kernel modules
- ⊕ Discovering hidden threads and processes

OMCD example: Shadow Walker detection

- ⊕ SW modifies IDT (INT 0xE hooking) → OMCD.F (IDT Verification)
 - ⊕ SW can modify Page Fault Handler code instead (Inline code hook) → OMCD.F (Code verification)
 - ⊕ SW can use Debug registers to hide IDT or Page fault handler modifications → OMCD.F (CPU Registers Verification)
-
- ⊕ No demo this time – I have no SW code :(
 - ⊕ As it is theoretical only I can be wrong here... anyone?

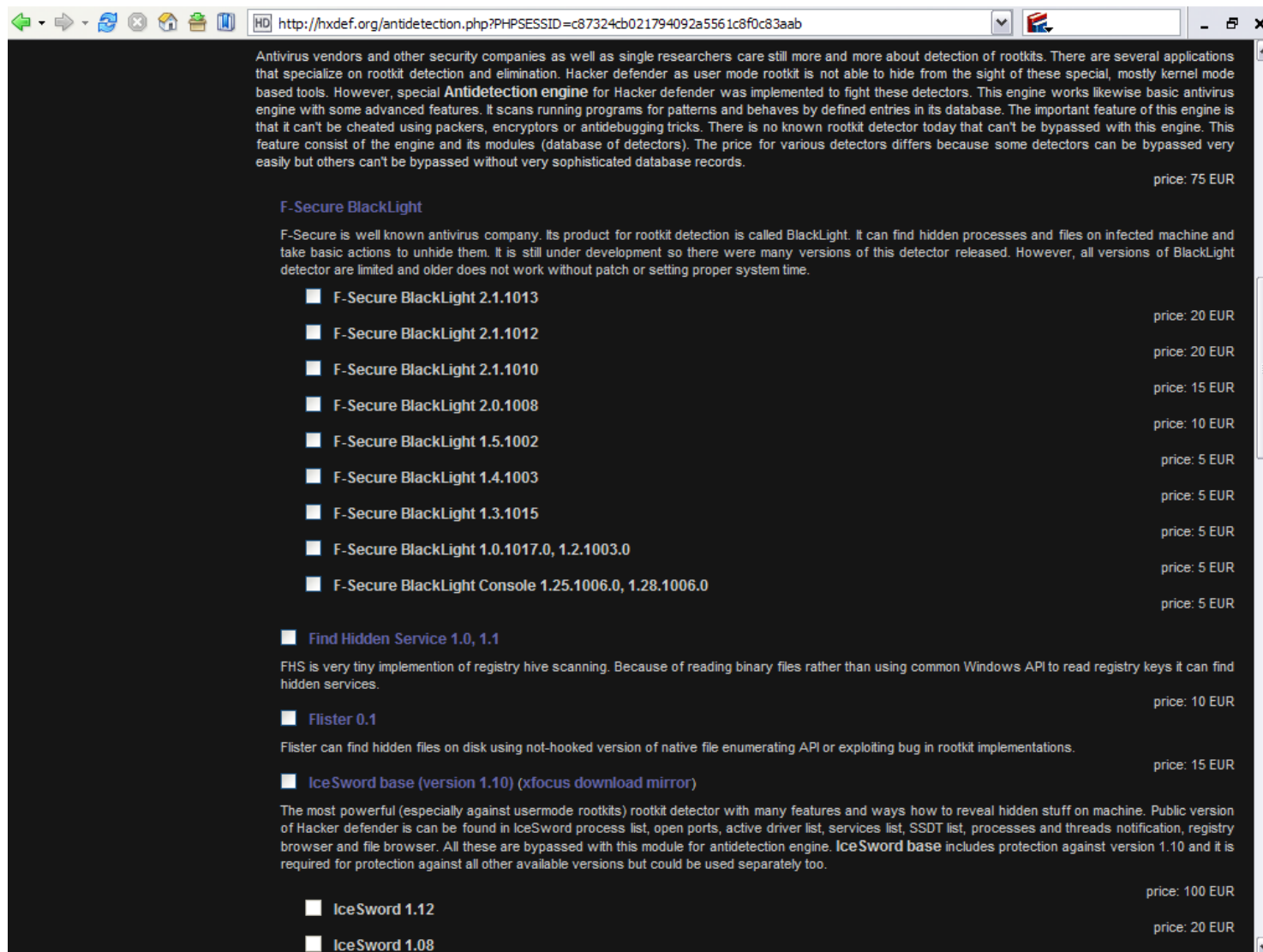
OMCD development

1. Release some early OMCD version (lots of gaps)
 2. Feedback from the community how rookits/backdoors can be implemented so that they are not detected by the latest OMCD version
 3. Update OMCD and goto #2
- ⊕ Thesis: The above algorithm is not infinite :)
 - ⊕ We are leaving less and less space for the malware to survive...
 - ⊕ If OMCD worked then in the future we would be observing only *implementation specific attacks*.

Implementation specific attacks

- ⊕ Malware author decides to cheat *particular* detector
- ⊕ It can for example:
 - ⊕ Hook IRP communication between detector and its kernel agent.
 - ⊕ “Exploit” design bug in the detector
 - ⊕ Cheat by hooking UI functions! (Win32 message hooking)
 - ⊕ Detect particular process (by signature scanning) and replace it with own version – which looks the same but reports clear system
 - ⊕ etc...

Implementation specific attacks



Antivirus vendors and other security companies as well as single researchers care still more and more about detection of rootkits. There are several applications that specialize on rootkit detection and elimination. Hacker defender as user mode rootkit is not able to hide from the sight of these special, mostly kernel mode based tools. However, special **Antidetection engine** for Hacker defender was implemented to fight these detectors. This engine works likewise basic antivirus engine with some advanced features. It scans running programs for patterns and behaves by defined entries in its database. The important feature of this engine is that it can't be cheated using packers, encryptors or antidebugging tricks. There is no known rootkit detector today that can't be bypassed with this engine. This feature consist of the engine and its modules (database of detectors). The price for various detectors differs because some detectors can be bypassed very easily but others can't be bypassed without very sophisticated database records.

price: 75 EUR

F-Secure BlackLight

F-Secure is well known antivirus company. Its product for rootkit detection is called BlackLight. It can find hidden processes and files on infected machine and take basic actions to unhide them. It is still under development so there were many versions of this detector released. However, all versions of BlackLight detector are limited and older does not work without patch or setting proper system time.

- F-Secure BlackLight 2.1.1013 price: 20 EUR
- F-Secure BlackLight 2.1.1012 price: 20 EUR
- F-Secure BlackLight 2.1.1010 price: 15 EUR
- F-Secure BlackLight 2.0.1008 price: 10 EUR
- F-Secure BlackLight 1.5.1002 price: 5 EUR
- F-Secure BlackLight 1.4.1003 price: 5 EUR
- F-Secure BlackLight 1.3.1015 price: 5 EUR
- F-Secure BlackLight 1.0.1017.0, 1.2.1003.0 price: 5 EUR
- F-Secure BlackLight Console 1.25.1006.0, 1.28.1006.0 price: 5 EUR

Find Hidden Service 1.0, 1.1

FHS is very tiny implementation of registry hive scanning. Because of reading binary files rather than using common Windows API to read registry keys it can find hidden services.

price: 10 EUR

Flister 0.1

Flister can find hidden files on disk using not-hooked version of native file enumerating API or exploiting bug in rootkit implementations.

price: 15 EUR

IceSword base (version 1.10) (xfocus download mirror)

The most powerful (especially against usermode rootkits) rootkit detector with many features and ways how to reveal hidden stuff on machine. Public version of Hacker defender is can be found in IceSword process list, open ports, active driver list, services list, SSDT list, processes and threads notification, registry browser and file browser. All these are bypassed with this module for antidetection engine. **IceSword base** includes protection against version 1.10 and it is required for protection against all other available versions but could be used separately too.

price: 100 EUR

- IceSword 1.12 price: 20 EUR
- IceSword 1.08

Implementation specific attacks

- ⊕ Let's say it aloud: they are ugly and stupid!
- ⊕ They cause people start asking existential questions (what's the sense of all of this?!)
- ⊕ Such attacks are *always* possible against a *particular* program (even if the detector has no bugs) – so don't confuse with bugs exploitation (Buffer Overflows and so)

Implementation Specific attacks: possible solutions

- ⊕ The more (various) detectors exist on the market the less profitable such attacks are for the malware authors
 - ⊕ Now the attacker starts acting just like his fellows from AV companies ;)
 - ⊕ OMCD aims to stimulate more various tools to be created
- ⊕ For commercial tools: make use of the update feature to constantly introduce small changes into detector (communication interface, UI, exec signatures, etc...). This could be automated, but the program for doing this should be kept private by the company.

Acknowledgements

- ⊕ Greg Wroblewski / Microsoft (for interesting discussions about kernel self modifications)
- ⊕ Sherri Sparks & Jamie Butler (for exploring new ideas for rootkit creation)