Creating an Anti-AV scanner …and blocking AVs

```
C:\Scanner>"Black Scanner.exe" "Anti-Bootkit Tools Signatures.bin" C:\Temp\AV-Zoo

  Black Scanner
  Engine version 1  Jan 23 2012

aswMBR.exe: Image Name infected
aswMBR.exe: Icon infected
aswMBR.exe: Icon infected
aswMBR.exe: Icon infected
aswMBR.exe: Icon infected
aswMBR.exe: Icon infected
aswMBR.exe: Icon infected
aswMBR.exe: Version Info -> Company infected
aswMBR.exe: Version Info -> Original File Name infected
TDSSKiller.exe: Image Name infected
TDSSKiller.exe: Section Name infected
TDSSKiller.exe: Icon infected
TDSSKiller.exe: Icon infected
TDSSKiller.exe: Icon infected
TDSSKiller.exe: Icon infected
TDSSKiller.exe: Icon infected
TDSSKiller.exe: Icon infected
TDSSKiller.exe: Icon infected
TDSSKiller.exe: Version Info -> Company infected
TDSSKiller.exe: Version Info -> Original File Name infected
TDSSKiller.exe: Certificate -> O infected
TDSSKiller.exe: Certificate -> CN infected
gmer 1.0.15.14972.exe: Icon infected
GMER 1.0.15.1508.exe: Icon infected
GMER 1.0.15.15641.exe: Icon infected
Gmer.exe: Image Name infected
Gmer.exe: Icon infected
gmer2u.exe: Icon infected
```

This article appeared in a magazine published at http://vx.netlux.org/spth/ on March 15, 2012 and may not be distributed in Austria.

Creating an Anti-AV scanner

There is malware in the wild that blocks anti-viruses from execution. They usually use a blacklist of hard coded file or process names. While that turned out to be working quite good, a more decent solution would be making a full anti-AV scanner based on real signatures.

Prior to this article I was watching a very interesting movement: A bootkit was blocking TDSSKillers execution by patching the entry point, and they (lame Kaspersky) responded by adding a TLS callback to their executable (that gets executed before the patched entry point).

TDSSKiller as anti-rootkit tool is a very good example, as it is the most aggressive (evil) one of its kind. It is packed like a virus, uses rootkit-like features and has pseudo heuristic signatures for MBRs such as *Rootkit.Win32.BackBoot.gen*. By making a signature over 440 bytes false positives are confirmed by oath.

In a discussion about this (prior to this paper) a very good developer came up with the "rootkit paradox":

There is what we call the "rootkit paradox," namely, the more you try to hide the easier you are to find. The reason is because everything you do to hide produces artifacts by which you can be detected. So you have to hide not only the original rootkit, but also all of the hooks that you use to hide. Once you have hidden your hooks you then have to hide everything you did to hide the hooks, and so on. Eventually the rootkit collapses under its own weight.

Obviously the solution to this rootkit paradox is either not doing any hooks (and being exposed to anti-viruses), or to block the execution of AVs in advance.

For some simple solutions in the wild that rely on the file name of the AV it is easy enough to rename the AV to bypass that. But what is about a full featured anti-AV scanner?

Important: This is research here and should be considered as proof of concept.

Peter Kleissner, Software Development G.

Table of Contents

The objective

We need a scanner that checks every file that is executed and stops its execution in case it is detected as an anti-virus. This also includes parts of anti-virus programs like libraries or drivers. We totally want to block them, including special removers and rootkit detection tools.

The technical requirements are: We only need to scan PE files loaded in memory. The code has to be portable, working both in user-mode and kernel-mode. Since this not an AV but an anti-AV we can rely on one important fact: AV files are easily identifiable and verifiable through meta-data such as file names, descriptions and certificates.

Meta-data in PE files that can be used for AV file identification are:

- Image name
- Icon
- Version Info -> Original File Name
- Version Info -> Company
- Section Name
- Certificate -> Organization Name (O)
- Certificate -> Common Name (CN)
- Certificate -> Fingerprint
- Debug data

Some remover tools are packed (some by UPX, some by custom ones); therefore some information could be encrypted. This is usually the case for the string table (as part of the resources) or the debug data.

The basic concept of AVs is "take the file and scan all signatures over it". This is obviously slow (but necessary). In our case, however, we can do it reverse: Take the meta-data, hash it and look it up in a (blacklist) tree. The time to scan a file is then nearly always the same: Doing the hashes, comparing them to the list (tree).

The signatures are stored in "trees" (which are none, technically just lists but it sounds better) of the hashes calculated from the meta-data. Candidates for the hashing algorithm would be CRC32, MD5 and SHA-1, as they are widely used and easy to implement. In my solution I use CRC32 as it just uses a dword for every hash (which makes the lookup pretty fast then).

Each hash tree identifies some meta-data type:

1	Image Name	`TDSSKiller.exe, Gmer.exe, aswMBR.exe`
2	Icon	
3	Company Name	`Kaspersky Lab ZAO, Kaspersky Lab`
4	PE Section Names	`.pklav`

Then there should be another tree for the certificate fingerprints, to explicitly blacklist them as well.

These hash trees do not equal the signature types. Rather, there are different types for the specific info that are checked:

```
#define BLACK_SIG_TYPE_IMAGENAME                        1
#define BLACK_SIG_TYPE_RESOURCE_ICON                    2
#define BLACK_SIG_TYPE_VERSION_INFO_ORIGINAL_FILENAME   3
#define BLACK_SIG_TYPE_VERSION_INFO_COMPANY             4
#define BLACK_SIG_TYPE_SECTION_NAME                     5
#define BLACK_SIG_TYPE_CERTIFICATE_CN                   6
#define BLACK_SIG_TYPE_CERTIFICATE_O                    7
```

The “image name” and the “version info -> original file name” signature types for example use the same hash tree (Image Name).

Signature Database

A scanner engine can only be as good (or in Aviras case as worse) as its signature database. The example used and shown at the first page of this article is "Anti-Bootkit Tools Signatures". Its source looks like this:

```
# blacklist TDSSKiller and Kaspersky Lab
1   TDSSKiller.exe
2   CRC32   1561770547
2   CRC32   1148540081
2   CRC32   616636565
2   CRC32   1009560204
2   CRC32   649440059
2   CRC32   1449080003
2   CRC32   -1989445225
3   "Kaspersky Lab"
3   "Kaspersky Lab ZAO"
4   .pklav

# blacklist Gmer
1   Gmer.exe
2   CRC32   -1651801256

# blacklist aswMBR (uses same engine as Gmer)
1   aswMBR.exe
2   CRC32   2008367110
2   CRC32   1770432001
2   CRC32   1969924194
2   CRC32   2103422372
2   CRC32   -1560200948
2   CRC32   -1770691826
3   "AVAST Software"
```

This text file is compiled to a binary using a custom parser and results in a 104 bytes binary file. The (binary) file format is very simple: It has a small header followed by the 4 signature trees.

```
typedef struct
{
    DWORD SizeOfHeader;
    DWORD CountSig1;
    DWORD CountSig2;
    DWORD CountSig3;
    DWORD CountSig4;
} SigHeader;
```

Gmer

Gmer deserves to be mentioned here in a single chapter. It has a history with malware trying to blocking it. If you want to download it, the website suggests already to use a button that generates a random file name, because some viruses used to block (hard-coded) Gmer.exe [1]:

It's recommended to download randomly named EXE (click button above) because some malware won't let gmer.exe launch.

The developer was very good with removing meta-data and identifiable information from the image – though he missed one point, the icon. No matter what version, the icon remains always the same, therefore making every Gmer executable identifiable.

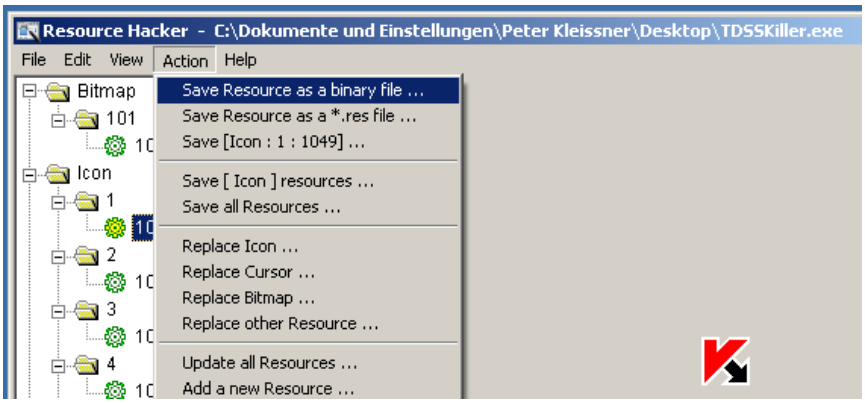
The Gmer icon is shown below, extracted using the program "Resource Hacker" [2].

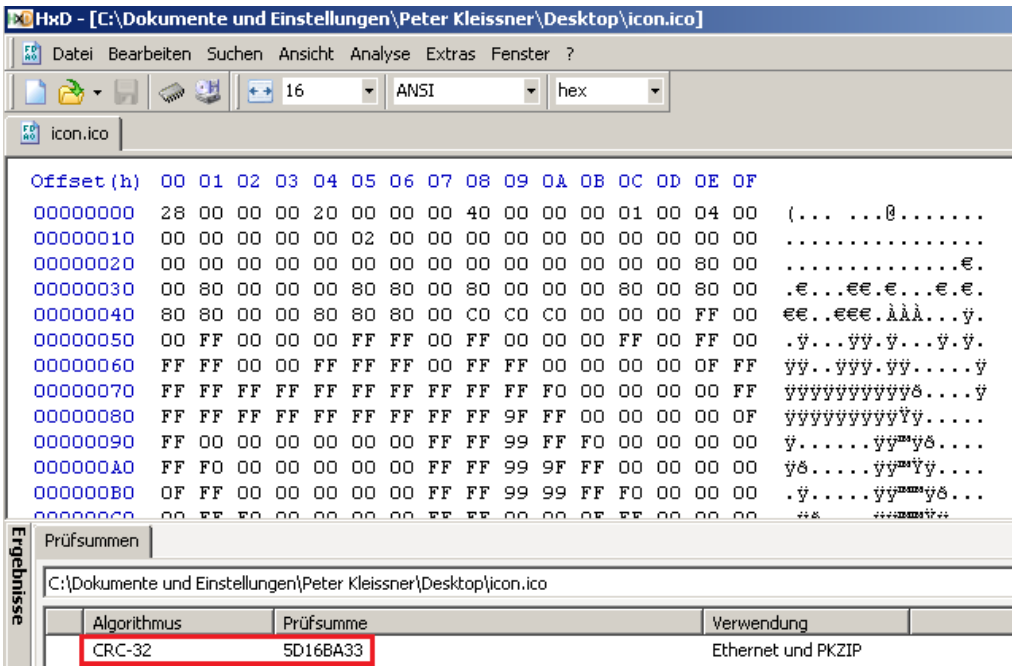
TDSSKiller

While Gmer is king in having sparse identifiable information, TDSSKiller is the opposite. The Black Scanner can be used to generate signatures (or at least output that helps in generating signatures) and this is the result for TDSSKiller:

```
TDSSKiller.exe: Image Name CRC -984132845
TDSSKiller.exe: Section Name CRC -395631440
TDSSKiller.exe: Section Name CRC -1620306906
TDSSKiller.exe: Section Name CRC -421465104
TDSSKiller.exe: Section Name CRC 501016963
TDSSKiller.exe: Icon CRC 1561770547
TDSSKiller.exe: Icon CRC 1148540081
TDSSKiller.exe: Icon CRC 616636565
TDSSKiller.exe: Icon CRC 1009560204
TDSSKiller.exe: Icon CRC 649440059
TDSSKiller.exe: Icon CRC 1449080003
TDSSKiller.exe: Icon CRC -1989445225
TDSSKiller.exe: Version Info -> Company CRC 1627962736
TDSSKiller.exe: Version Info -> Original File Name CRC -984132845
TDSSKiller.exe: Certificate -> O CRC 1231392781
TDSSKiller.exe: Certificate -> CN CRC 1231392781
```

This output shows the CRCs (in decimal notation) of the info. You can directly compare the CRCs of the icons with the sample signature database in the chapter "Signature Database". Prior to this generation functionality of the scanner I had to manually extract the icon, and then generate the CRC32 using 3rd party tools:





This acquired CRC32 can be added to the signature database by adding a single line. The 2 means a CRC32 signature of a RT_ICON resource:

```
2 CRC32 1561770547
```

Preventing files from being executed

PE image files can be scanned in the PsSetLoadImageNotifyRoutine driver callback function. That is, after the image is loaded into memory, but before it is being executed. Previous blocking solutions in-the-wild have been implemented as:

1. Injecting an APC that calls ExitProcess() – done by ZeroAccess [3]

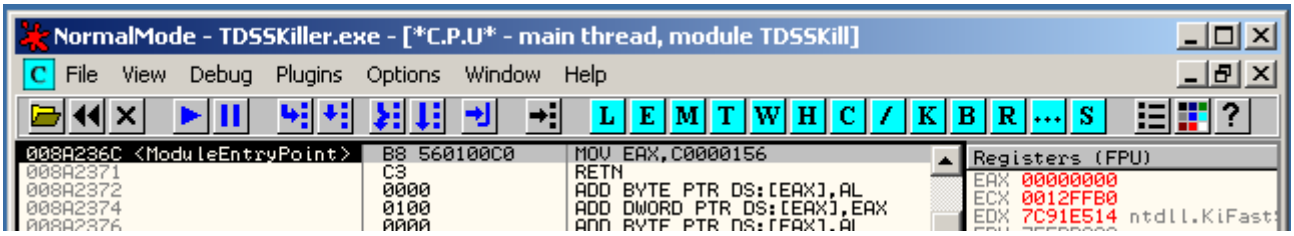
When a typical security scanner tries to analyze the rootkit-created svchost.exe file, the rootkit queues an initialized APC into the scanner's own process, then calls the ExitProcess() function — essentially forcing the scanner to kill itself.

2. Patching the entry point with a ret instruction – done by BlackEnergy [4]

Instead of defence, this malware attacks AV by preventing AV processes from loading. It registers notification callbacks using functions such as PsSetLoadImageNotifyRoutine.

When a process is loading, the malware gets a callback by the OS where it first checks if the loading process is on its blacklist of known AV processes. If it isn't, then the malware allows the process to load normally. But if it is an AV process, the malware writes a return instruction (0xc3) at the entry point in the memory of the process and then allows it to load. The AV process typically terminates immediately due to this patch by the malware. This technique was first reported to be used in the wild by Nuwar back in 2007.

The simplest (and most practical) solution is to overwrite the entry point with a return code. Below the entry point is patched to return the infamous STATUS_TOO_MANY_SECRETS error code, from a bootkit (which hashes the image name and compares it to a blacklist):



This error code is also used by ZeroAccess [3]:

```
call    Kill_Process_and_File
mov     eax, 0C0000156h ; STATUS_TOO_MANY_SECRETS
jmp     _end
```

And also by TDL4 [5]:

When a program or driver attempts to open one of the \Device\HarddiskVolumeX devices, the rootkit will return error code 0xC0000156 (STATUS_TOO_MANY_SECRETS - the author had a sense of humor) to block

opening. Because most antirootkits try to open this (for raw-disk scanning), most of them will have issues.

Obviously also the TLS callbacks have to be patched, as they are executed before the entry point.

A more "funny" solution would be to wipe the entire hard disk (and maybe flash the boot firmware), to force the vendor to withdraw the tool. Of course I am not supporting that, but installing Kaspersky on your machine makes it to a brick already anyway.

Preventing files from being executed

Parsing Certificates

Even though there are Windows (user-mode) functions to handle certificates (and extract information) they cannot be used here, because one requirement of the scanner is to run within kernel mode.

Unfortunately, to make things even more complicated, the certificate table has to be loaded explicitly into memory by the scanner. The PE documentation (pecoff_v8.docx) says:

These certificates are not loaded into memory as part of the image. As such, the first field of this entry, which is normally an RVA, is a file pointer instead.

For the execution of user mode programs this makes perfectly sense, because the certificate is not checked there. I verified that the pointer remains a file pointer of the loaded image in memory (no cheating by Microsofts PE loader). Also scanning the memory for the certificate was without luck. In kernel mode, digital signatures of images are verified by ImgpValidateImageHash, an important function for bootkits (explained in "The Art of Bootkit Development").

In fact the certificate is stored immediately after the Certificate Table in PCKS7 DER encoded format. Microsoft calls it "Authenticode". OpenSSL supports that format, you can launch following command on the dumped raw certificate:

```
openssl.exe pkcs7 -inform DER -text -print_certs -in rawcert.pkcs7

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            11:a3:0b:cf:b2:e8:2a:d7:1f:54:1d:11:27:ab:d1:f6
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, O=VeriSign, Inc., OU=VeriSign Trust Network, OU=Terms
of use at https://www.verisign.com/rpa (c)10, CN=VeriSign Class 3 Code
Signing 2010 CA
        Validity
            Not Before: Feb 21 00:00:00 2011 GMT
            Not After : Mar  7 23:59:59 2012 GMT
        Subject: C=RU, ST=Moscow, L=Moscow, O=Kaspersky Lab, OU=Digital ID
Class 3 - Microsoft Software Validation v2, OU=Technical dept, CN=Kaspersky
Lab
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:bd:95:60:5a:15:bb:1d:de:23:a6:3b:ab:d2:73:
                    ea:44:38:1e:ee:29:91:7f:f3:22:b6:39:ce:40:1f:
                    43:45:24:7d:b4:b0:df:6e:bf:5d:3a:19:f9:c8:e1:
...
```

One interesting thing about Authenticode is that Microsoft still allows the use of MD5 hashes, whereof collisions can be easily generated. SHA-1 is used by Microsofts code signing tool by default though. [6] [7]

The OpenSSL library, however, turns out to be unusable in the scanner. It is bloated up, (the official distribution) requires MinGW/Gnu LibGwC on Windows, contains as good as no comments in its source (badly documented, the spare html website does not help there either) is unreadable and the PKCS7 code is not abstracted from things like the data input at all. I would not call this interoperability.

However, there is a very good lightweight ASN.1 dumping code available from Peter Gutmann. PKCS7 uses ASN.1 representation. The interesting fields there are the ones with the object identifier "2 5 4 10" = Organization Name (O) and "2 5 4 3" = Common Name (CN).

AV Zoo

Opening an AV zoo is important for the signature generation to stay up to date. Like AVs download the virus samples directly from the source (from the bad guys C&Cs), the AVs (or remover tools) can be downloaded automatically.

TDSSKiller is a good example here, on every run it checks if there is an updated version on the Kaspersky server. For that reason it downloads an xml file:

```
4        0.064434      10.0.0.2      195.27.181.39        HTTP    GET
/downloads/utils/tdsskiller.xml HTTP/1.1
```

That file can be downloaded directly from http://195.27.181.39/downloads/utils/tdsskiller.xml:

```
<tdsskiller>
      <release>
            <version>2.7.0.0</version>

<downloadzip>http://support.kaspersky.com/downloads/utils/tdsskiller.zip</d
ownloadzip>

<description>http://support.kaspersky.com/faq/?qid=208283363</description>
      </release>
</tdsskiller>
```

This simple xml format is ideal for automating the download process, and automatically adding the latest TDSSKiller version to the zoo.

Ideally, an automated system checks every half hour for a new version and automatically generates a new signature in case it is not detected. For a botnet, this means the bad guys could always stay ahead of anti-viruses. The process:

1. Dowload anti-virus executable
2. Check if it is a new file (SHA-1 hash), if not -> exit
3. Add it to the AV database (zoo)
4. Check if it is detected by current signatures, if so -> exit
5. Generate a new signature and alarm the developer
    a. Compare the signature over legitimate files
    b. Generate a new signature database
6. Deploy the new signature database to the botnet

Behind the scenes – technical implementation

The project is split into the scanner and the actual scanner engine. The engine is coded in C/C++ and uses a structure to save its internal state:

```
struct Scanner
{
    // Input File (all parameters are required)
    char * ImageName;
    wchar_t * FullFilePath;
    void * ImageBase;

    // Scanning Options
    BOOL StopOnFirstFound;
    BOOL SignatureGenerationMode;
    void (* Callback)(const Scanner * Instance);

    // Results
    unsigned SignaturesFound;
    unsigned LastSigType;
    DWORD LastBadCRC;

    // Internal Data
    PIMAGE_NT_HEADERS NtHeader;
    DWORD ImageSize;
};
```

The callback is called on every found infection. The scanner engine was developed to check files that are loaded into memory (on the image notification callback). To check files on hard disk, the scanner uses LoadLibraryEx with the DONT_RESOLVE_DLL_REFERENCES flag (that prevents the entry point of being called).

The most complicated code part was the resource parsing. Someone at Microsoft should be fired for making these unwell defined “flying around” structures.

The core function is ScanCRCInDatabase which looks up a generated CRC in the tree corresponding to the specific signature type. If it finds it, it will invoke the callback.

The source is published under the European Union Public Licence.

Conclusion

What I have shown here is the reverse operation of an anti-virus, relying on meta-data and the fact that AV files are easy identifiable. Obviously this is only of advantage if the virus is *already installed*. What I have described is a way to protect from anti-viruses, not how to bypass them.

At the end of the day virus developers always have a small advantage though. They can use services like scan4you or KIMS to check if they are being detected and doing appropriate changes. AV vendors, however, cannot simply change their file name, product name etc.

Conficker is a famous example of a virus that was blocking AVs as hell. With the result however, that is was simple enough visiting a website to check if someone's infected (Conficker blocks AV websites so embedded pictures linked to them would not show up).

Peter Kleissner

References

[1] Gmer
http://www.gmer.net/

[2] Resource Hacker
http://www.angusj.com/resourcehacker/

[3] Webroot: ZeroAccess Rootkit Guards Itself with a Tripwire
http://blog.webroot.com/2011/07/08/zeroaccess-rootkit-guards-itself-with-a-tripwire/

[4] McAfee: Predicting the future of stealth attacks
http://www.mcafee.com/us/resources/reports/rp-predicting-stealth-attacks.pdf

[5] Forum entry at Sysinternals "Interesting new malware"
http://forum.sysinternals.com/interesting-new-malware_topic15413.html

[6] Playing With Authenticode and MD5 Collisions
http://blog.didierstevens.com/2009/01/17/playing-with-authenticode-and-md5-collisions/

[7] MD5 Collision Demo
http://www.mscs.dal.ca/~selinger/md5collision/

[8] Format of a Signed File and ASN.1 dumping code
http://www.cs.auckland.ac.nz/~pgut001/pubs/authenticode.txt
http://www.cs.auckland.ac.nz/~pgut001/dumpasn1.c