

How Antivirus Software Works and How to Evade it

As hackers, we often are required to get past antivirus (AV) software or other security measures. To do so effectively, we need to have some understanding of how AV software works. In this tutorial, we will take a cursory view of how AV software works so that you can better strategize on how to [evade detection by it](#).

The antivirus software industry is huge and well-funded. As you would expect, they employ a large number of malware detection and identification personnel. (I will use the preferred term "malware" to mean all bad or malicious software including viruses, spyware, rootkits, worms, etc.)

Many of these people have a hacker or virus-making background, so the things we are doing here are not mysterious to them. (Eugene Kaspersky, the founder Kaspersky Labs, is rumored to have worked at one time as a hacker/cryptographer for Russia's KGB. Many others in the industry have a background in virus/worm creation.)

The AV industry attempts to collect as much information as they can about threats new and old and package that information into their software. The older a tool or technique is, the greater the chance they will have a mechanism to detect it.

Types of AV Detection Mechanisms:

Signature-Based

The key to making good AV software is to have a complete database of all malware signatures. These signatures are the essential part of the malware that distinguishes it from other software. That's why it's key for users to keep those databases updated daily. As new malware is introduced daily into the wild, an out-of-date database of signatures can become nearly useless.

For a better concept of what these signature databases are like, you might take a look at [Snort](#), an open-source IDS that detects malware and attacks on the wire. The beauty of working with Snort is that its signature database is open and viewable to anyone.

In the screenshot below, you can see some of the rules or signatures from the *web-attacks.rules* file from Snort. To better understand malware signatures, you may want to

take some time to study these "signatures." It's important to note that these rules don't look for the entire file, but simply the single element of the malware that is unique or its "signature." Obviously, this is much more efficient.

```
28
29 S $HTTP_PORTS (msg:"WEB-ATTACKS /bin/ps command attempt"; flow:to_server,established; uricontent:"/bin/ps"; nocase; classtype:web-appli
30 S $HTTP_PORTS (msg:"WEB-ATTACKS ps command attempt"; flow:to_server,established; uricontent:"ps%20"; nocase; classtype:web-appli
31 S $HTTP_PORTS (msg:"WEB-ATTACKS wget command attempt"; flow:to_server,established; content:"wget%20"; nocase; classtype:web-appli
32 S $HTTP_PORTS (msg:"WEB-ATTACKS uname -a command attempt"; flow:to_server,established; content:"uname%20-a"; nocase; classtype:v
33 S $HTTP_PORTS (msg:"WEB-ATTACKS /usr/bin/id command attempt"; flow:to_server,established; content:"/usr/bin/id"; nocase; classty
34 S $HTTP_PORTS (msg:"WEB-ATTACKS id command attempt"; flow:to_server,established; content:"|3B|id"; nocase; classtype:web-appl
35 S $HTTP_PORTS (msg:"WEB-ATTACKS echo command attempt"; flow:to_server,established; content:"/bin/echo"; nocase; classtype:web-ap
36 S $HTTP_PORTS (msg:"WEB-ATTACKS kill command attempt"; flow:to_server,established; content:"/bin/kill"; nocase; classtype:web-ap
37 S $HTTP_PORTS (msg:"WEB-ATTACKS chmod command attempt"; flow:to_server,established; content:"/bin/chmod"; nocase; classtype:web-
38 S $HTTP_PORTS (msg:"WEB-ATTACKS chgrp command attempt"; flow:to_server,established; content:"/chgrp"; nocase; classtype:web-appl
39 S $HTTP_PORTS (msg:"WEB-ATTACKS chown command attempt"; flow:to_server,established; content:"/chown"; nocase; classtype:web-appl
40 S $HTTP_PORTS (msg:"WEB-ATTACKS chsh command attempt"; flow:to_server,established; content:"/usr/bin/chsh"; nocase; classtype:we
41 S $HTTP_PORTS (msg:"WEB-ATTACKS tftp command attempt"; flow:to_server,established; content:"tftp%20"; nocase; classtype:web-appl
42 S $HTTP_PORTS (msg:"WEB-ATTACKS /usr/bin/gcc command attempt"; flow:to_server,established; content:"/usr/bin/gcc"; nocase; class
43 S $HTTP_PORTS (msg:"WEB-ATTACKS gcc command attempt"; flow:to_server,established; content:"gcc%20-o"; nocase; classtype:web-appl
44 S $HTTP_PORTS (msg:"WEB-ATTACKS /usr/bin/cc command attempt"; flow:to_server,established; content:"/usr/bin/cc"; nocase; classty
45 S $HTTP_PORTS (msg:"WEB-ATTACKS cc command attempt"; flow:to_server,established; content:"cc%20"; nocase; classtype:web-appl
46 S $HTTP_PORTS (msg:"WEB-ATTACKS /usr/bin/cpp command attempt"; flow:to_server,established; content:"/usr/bin/cpp"; nocase; class
47 S $HTTP_PORTS (msg:"WEB-ATTACKS cpp command attempt"; flow:to_server,established; content:"cpp%20"; nocase; classtype:web-appl
48 S $HTTP_PORTS (msg:"WEB-ATTACKS /usr/bin/g++ command attempt"; flow:to_server,established; content:"/usr/bin/g++"; nocase; class
49 S $HTTP_PORTS (msg:"WEB-ATTACKS g++ command attempt"; flow:to_server,established; content:"g++%20"; nocase; classtype:web-appl
50 S $HTTP_PORTS (msg:"WEB-ATTACKS bin/python access attempt"; flow:to_server,established; content:"bin/python"; nocase; classtype:
51 S $HTTP_PORTS (msg:"WEB-ATTACKS python access attempt"; flow:to_server,established; content:"python%20"; nocase; classtype:web-a
52 S $HTTP_PORTS (msg:"WEB-ATTACKS bin/tclsh execution attempt"; flow:to_server,established; content:"bin/tclsh"; nocase; classtype
53 S $HTTP_PORTS (msg:"WEB-ATTACKS tclsh execution attempt"; flow:to_server,established; content:"tclsh%20"; nocase; classtype:web
54 S $HTTP_PORTS (msg:"WEB-ATTACKS bin/nasm command attempt"; flow:to_server,established; content:"bin/nasm"; nocase; classtype:web
55 S $HTTP_PORTS (msg:"WEB-ATTACKS nasm command attempt"; flow:to_server,established; content:"nasm%20"; nocase; classtype:web-appl
56 S $HTTP_PORTS (msg:"WEB-ATTACKS /usr/bin/perl execution attempt"; flow:to_server,established; content:"/usr/bin/perl"; nocase; c
57 S $HTTP_PORTS (msg:"WEB-ATTACKS perl execution attempt"; flow:to_server,established; content:"perl%20"; nocase; classtype:web-ap
58 S $HTTP_PORTS (msg:"WEB-ATTACKS nt admin addition attempt"; flow:to_server,established; content:"net localgroup administrators /
59 S $HTTP_PORTS (msg:"WEB-ATTACKS traceroute command attempt"; flow:to_server,established; content:"traceroute%20"; nocase; classt
60 S $HTTP_PORTS (msg:"WEB-ATTACKS ping command attempt"; flow:to_server,established; content:"/bin/ping"; nocase; classtype:web-ap
61 S $HTTP_PORTS (msg:"WEB-ATTACKS netcat command attempt"; flow:to_server,established; content:"nc%20"; nocase; classtype:web-appl
62 S $HTTP_PORTS (msg:"WEB-ATTACKS nmap command attempt"; flow:to_server,established; content:"nmap%20"; nocase; classtype:web-appl
63 S $HTTP_PORTS (msg:"WEB-ATTACKS xterm command attempt"; flow:to_server,established; content:"/usr/X11R6/bin/xterm"; nocase; clas
64 S $HTTP_PORTS (msg:"WEB-ATTACKS X application to remote host attempt"; flow:to_server,established; content:"%20-display%20"; noc
```

The AV software companies maintain individuals and honeypots around the world that are constantly searching for new malware, and when they find it, they try to condense its signature down to the essentials. In addition, they have users all over the globe who send them suspicious malware they find.

These signatures, as I'm sure you guessed, are only effective on known malware. A zero-day attack would *not* have a signature and therefore would likely go undetected by the antivirus software.

One of the exceptions to this is that some malware production tools leave a signature on their output, so even a new piece of malware might be detected by AV if the tools used to create it have a signature that the AV developers have identified and coded for. A good example of this that many of you have found is the [msfvenom](#) module in [Metasploit](#). The template that creates the payloads has a signature, so no matter how we re-encode our payload, it still has a known signature. The key to defeat the AV with this module is to use a new template.

Heuristics

In some cases, it is not possible to have a signature for all malware, and in those cases, the AV developers attempt to deploy heuristic techniques to detect malware.

Wikipedia [defines](#) heuristic techniques as "...any approach to problem solving, learning, or discovery that employs a practical methodology not guaranteed to be optimal or perfect, but sufficient for the immediate goals. Where finding an optimal solution is impossible or impractical, heuristic methods can be used to speed up the process of finding a satisfactory solution."

James Whitcomb Riley once said, "When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck." That principle summarizes heuristics succinctly. If it looks like malware and behaves like malware, it's probably malware.

The AV software developers look for telltale signs that the software's structure or behavior is malicious and then treat it like malware, even if no signature exists. For instance, if a file begins to replace several system files, it's probably malware. If a piece of software is trying to make a TCP connection back to a known malicious IP address, it's probably malware. The beauty of this strategy is that it works with zero-day malware just as well as known malware. The drawbacks of this strategy are that it requires more compute cycles *and* the false positives that are often produced by innocuous software.

Behavior-Based

Some people create a separate category of behavior-based malware detection, but I consider it to be a subcategory of heuristics. Behavior is part of the "walks like a duck" mentioned above.

Sandbox

In some cases, suspicious software can be run in a virtual machine environment to see what it will do *before* it is installed on the system. This is referred to as "sandboxing." In this way, the AV software can study what it does or tries to do and then determine whether it is safe without endangering the entire system.



Image via [Lowe's](#)

Due to the time and resources necessary, this isn't a practical way of dealing with all software and files, but for particularly suspicious ones, it can be effective.

Evading AV Software:

First, let's state clearly that AV software is *not* perfect. Even with all their resources and skill, they still miss known malware. Depending what package you are using, detection of known malware can be as low as 40% and as high as 98 percent. Even in this best case scenario, that means that 1 out of 50 pieces of *known* malware gets past the AV software.


Msfvenom

As most of you know, [msfvenom](#) is a module in Metasploit that enables us to re-encode our payloads to evade AV software. Due to the ubiquity of Metasploit, as soon as a new encoding scheme is developed by Metasploit, the AV makers jump on it and produce a signature (they would be foolish if they didn't).

As I mentioned above, the AV makers don't really need to have a signature for every re-encoding by [msfvenom](#), they only need a signature for the template. This means that [msfvenom](#) has limited effectiveness *without an original template*.

Testing Your Malware with Virus Total

If you want to test whether your malware, rootkit, virus, or whatever will be detected by AV software, you can submit it to [VirusTotal](https://www.virustotal.com), which will use over 60 commercial software packages to determine whether your malware will be detected. You can keep re-encoding your malware until VirusTotal indicates it will not be detected.



The screenshot shows the VirusTotal website. At the top is a navigation bar with links: Home, Community, Statistics, Documentation, FAQ, About, English, Join our community, and Sign in. The main header features the VirusTotal logo and a description: "VirusTotal is a free service that analyzes suspicious files and URLs and facilitates the quick detection of viruses, worms, trojans, and all kinds of malware." Below this is a search bar with tabs for File, URL, and Search. A file upload area shows "No file selected" and a "Choose File" button, with a note "Maximum file size: 128MB". A consent message states: "By clicking 'Scan it!', you consent to our Terms of Service and allow VirusTotal to share this file with the security community. See our Privacy Policy for details." A large blue "Scan it!" button is centered below the message. At the bottom, a footer contains links for Blog, Twitter, contact@virustotal.com, Google groups, ToS, and Privacy policy.

Unfortunately, if you are successful in being undetected by any of these AV packages, VirusTotal will present your malware to the AV industry to develop a signature. Despite this, there will still be a window before the AV developers add the signature and the end users update their signature database.

Veil Evasion

[Veil-Evasion](#) is a software package for changing the signature of your payload executables so that they can bypass AV software. Veil is now in the Kali repository and can be installed simply by typing:

```
kali > apt-get install veil
```

```
root@kali:~# apt-get install veil
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  pure-ftpd-common
Use 'apt-get autoremove' to remove it.
The following extra packages will be installed:
```

I will be doing a tutorial on its use in the near future.

Web Mail Detection

Many of you have struggled sending malware via web-based mail, such as Gmail, Yahoo, Hotmail, etc. Of course, those companies maintain their legitimacy by deploying one of the major AV companies to scan any attachments you might try to send. In most cases, trying to send a malicious [PDF](#), [Word document](#), or other file will likely be detected by these services and stopped.

One of the beauties of using a Linux distribution is the ability to set up your own [SMTP server](#) and use it to send your email. There are numerous SMTP servers available for Linux that you can download and install including [atmail](#), [Exim](#), [qmail](#), [Postfix](#), and [sendmail](#). Look for my tutorial coming soon on installing the Exim SMTP server in my [Linux Basics series](#).

Having your own SMTP server would resolve any issues with *sending* malicious payloads but, of course, will not resolve issues on the receiving end.

Build Your Own

Ultimately, the best way to evade AV and other security devices is to build your own exploit/payload. That is our goal in my [Exploit Building series](#), where we are working to develop our own unique exploit that will be undetectable by any AV or security device.