

The Threat of Shellcode Obfuscation

By Don Parker

hydra291@hotmail.com

Introduction

Most large corporations, government and military computer networks have a central focal point from which they can manage and administer such things as patches, operating system updates and vulnerability fixes. Most of the aforementioned will push out the fixes for whatever vendor vulnerabilities have been discovered via a baseline rollout. That is assuming as well that these very same networks employ a software baseline through which they can regulate the workstation's software configuration. Should these networks be configured this way, then performing a baseline rollout is by far the most effective means of managing operating system patches and vulnerability fixes. It has been noted, though, by some system administrators that some of these patches actually have broken functionality on some of their network's workstations and servers.

This has sometimes led to a large gap between the time that a software vendor releases an advisory or fix based upon a security researcher's vulnerability and the time that the system administrator actually applies the fix. The fear that a fix will actually break a system is a well-founded one, but it must be mitigated as best possible. Normally one should never apply such remedies to a production system without first having tested the fixes out in a controlled lab environment. By doing testing on a lab machine it is of no importance if the vendor fix crashes or locks up the lab system. It is of importance, though, that the test machine or server mirror exactly the software configuration of your production counterpart. By doing patch testing in this way, you will find out definitively if there will be any software conflicts or other oddball problems as a result of applying these fixes.

Speed is of the essence when protecting your network assets against the never-ending stream of vulnerabilities. You must endeavor to have the fix downloaded and tested quickly so you can roll out as fast as possible the vendor solution. Simply saying that you want to ensure functionality and taking weeks before applying the fix is no longer acceptable. No longer do you have the option of waiting to see what the community has to say about the fix before you actually install the fix. Well why the importance to patch so quickly, you ask? Many of the buffer overflows and format string attacks chronicled on full disclosure sites are easily used by people who have no idea how to actually program. These are the group of people that many refer to as script kiddies. What they do know how to do, though, is download fully functional exploit code and attack computers at large. Compiling code is not only in the realm of the programmer and savvy user. Using gcc is an easy affair, as evidenced by the huge amount of exploit code being picked up by application-level firewalls and intrusion detection systems.

IDS Signatures

While it may be difficult for large networks to get remedies pushed out to the workstations and servers in a timely fashion, getting updated IDS

signatures loaded is quicker. Normally on a critical threat an IDS vendor will have a signature developed and ready for download within a couple of days. Of note is that vendors may have patches quickly available, too, but the impact to business operations is different in that a hastily applied patch can break functionality and more directly impact revenues, reputation and other undesirable consequences. However, an IDS signature will impact security operations, which can affect profits through higher maintenance costs, though it typically takes a back seat to business operations.

You should also tweak IDS signatures if possible for your specific needs and environment. Fine-tuning your IDS to your actual needs and services being offered by your network are crucial. There are several reasons as to why your IDS should be tailored for your network. A mistake commonly made is that all signatures are on; that list could probably be pruned by half or less. All this white noise generated by your IDS will only serve to wear down and desensitize the analyst checking its alarms. Human nature being what it is, soon enough the system will largely be ignored, and possible critical alarms will not be checked and will be thought of as just another false positive.

With that in mind, now is the time to draw up a battle plan to fine-tune that expensive IDS and make the most of its capabilities. What should be done first is an audit of the services being offered by your network. This will in turn allow you to go through your signature library and turn off the ones that are not needed. There is no sense in having FTP signatures if you are not offering FTP on your network. Now that you have a baseline of services offered and only those signatures required turned on, you may start further fine-tuning the IDS. Once the alarms trigger, take a careful look at each and every one of them. Most vendors will supply a library of what each signature trips on as well as related false positive criteria, which may set it off. IDS signature criteria can be varied from ASCII content in the packet to a specific hex signature. Matches between the aforementioned and some other TCP/IP metric like port number are also used to write IDS signatures. Please note as well that ideally an organization's application/server deployment strategy should include not only security audits, but a review of the IDS signatures to ensure that attacks against the new application/server are detected. This can be done by contracting out a specialized security firm, which will ensure your security posture is indeed as it should be.

At this point, you explore exactly why the signature fires and what part of the packet makes the IDS signature trip. By extension you will also see what makes a seemingly innocuous packet fire off a signature. Once you have seen the cause of false positives and understand what is happening, you may wish to take this up with your IDS vendor in an effort to reduce the bane of the IDS—the false positive. If possible, attempt to rid yourself of any conditions which are resulting in the IDS false positives. Each and every alarm will need to be proven unequivocally a false positive. By doing so, you will gain a deeper understanding of packet analysis, so all is not in vain. Packet analysis is a very important skill to learn for the network security

analyst. If your in-house staff does not possess it, I highly suggest you have them trained in it.

Exploit Code

We can now see that although vulnerabilities and normal operating system updates may not be rolled out immediately, some of the dangers posed by this can be mitigated. Some of the reasons mentioned for not rolling out a fix posthaste are valid ones like lab testing of the patch and system update first, size and complexity of networks, as well as other possible considerations. Well at least you have a finely tuned IDS with some brand new signatures to keep you safe from that newly published exploit code. Is it really protecting you though? There have been some recent "improvements" to the world of exploit code that make it far more difficult to detect. First, though, I will attempt to explain what I mean by exploit code and the ASM-based component.

All of the sexy exploit stuff seems to be buffer overflows. This is a truism for a simple reason. The buffer overflow can result in privileged access and or some other type of high-threat danger. Through a buffer overflow, an attacker can execute code of their choosing, whether it is a reverse command shell, port scanner or TFTP server. This code is sent in what is referred to as an egg or sometimes, egg shell, written in the machine language of the victim computer. Most overflow eggs are written using an Assembler instead of a high-level compiler and handcrafted to minimize their size. Writing code in Assembler is not an overly user-friendly task and requires a high level of programming skill.

NOOP Sleds

Almost all of these exploits follow the same format. You have the "injector" written in either C or C++ whose purpose is to modify the overflow egg for the particular operating system running on the victim computer and then deliver and sometimes interact with the overflow egg. Many overflow exploits will use an egg with a NOOP sled. This NOOP provides a "fudge factor" so the attacker does not have to predict the exact location of the overflow egg in the victim computer's memory. Therefore, when the CPU jumps to the overflow egg, if it misses by a few bytes, it will land on the NOOP sled and harmlessly slide down to the egg. The NOOP sled itself is composed of a sequence of one-byte instructions literally meaning "no operation." In essence this instruction tells the computer's central processing unit (CPU) to do nothing. It is a harmless instruction that will not affect the state of the computer itself. Were you to put in a different instruction, it is possible that you would arbitrarily affect the state of the computer and cause it to crash.

The people who build and maintain signatures for IDSs quickly realized this. What they did in turn to try and counteract this high-level threat was write signatures that look for large amounts of NOOPs in the hex portion of the packet, as it is being compared against the IDS signature list. The IDS vendors also added such signatures as looking for /bin/sh in the ASCII content of a packet, which is a string used in UNIX command shell eggs. Adding such signatures has proven very effective indeed. Having shellcode signatures turned on, they are prone to the normal false positive. The IDS may take a performance hit as well, for with this rule, you are now checking the entire packet contents normally.

Circumventing Signatures


Exploit coders, being very clever to begin with, realized this and began to devise ways of circumventing these signatures. In quick succession

there were a variety of new ways to try and slip past undetected by the IDS. There was ASCII shellcode, polymorphic shellcode, and also polymorphic ASCII shellcode, to name some of the publicly known ones. In turn there have been tools released by some of these talented coders that have simplified the process of camouflaging shellcode. These are tools such ADMmutate and Phiral, which automate the obfuscation process. Since the use of such tools is beyond the skill level of the average script kiddie, their usage is more in the realm of the novice programmer, who at least makes an attempt to learn and has some existing knowledge of C and a low-level language such as Assembler. Prior to the release of such tools, one would have had to simply swap out the NOOP sled and find suitable idempotent substitutions. By being idempotent, the instruction will not adversely affect the state of the computer or program and avoid a crash. The idempotent substitution also must be one byte on length in case the CPU starts executing on the off-byte, changing the interpretation of the instruction. Multi-byte idempotent instructions may be possible though every cut of the instruction sequence must in itself yield an idempotent instruction. Finding such instructions took time and testing to make sure that the exploit still indeed worked after substituting the well-known IDS-triggering value of NOOP.

Conclusion

There are ways of making shellcode far more difficult if not impossible to detect on some networks, depending on their configuration. Most people who also deal with IDS on a daily basis also realize that having the shellcode signature in place also results in a fair amount of false positives. The only real way to try and minimize this problem is by having the person who actually looks at the IDS output fully trained. This would entail having this person understand what shellcode looks like and what some of the various hex values equate to. Only through educating the human interface to the IDS will we be able to deal as effectively as possible with this very dangerous threat. Computers will almost always function as advertised or programmed. Not enough attention is paid to properly training the individual themselves. The cost of doing so is minimal once you compare it to the possible consequences of a network breach.

The beauty of computer communications is that it always comes down to the packet. Because of this, it is impossible to hide anything within that packet, for it has to be sent to your computer. You can obfuscate it, and do your best to hide the packet's intent, but the malicious content will still be there. All of the information is there to be had, as long as the person takes the time to look at and research the contents of the packet if required. Shellcode obfuscation presents a serious threat indeed, but the threat can be managed through training and having a dedicated network security staff. There are also other solutions to this, such as application-layer firewalls.

Hopefully having now seen that obfuscated shellcode can be a serious problem, this article will have enlightened you to the importance of rolling out your patches and other system fixes as quickly as possible. Make sure also that you update your IDS signatures on a regular basis. Most importantly, make sure you have your network security staff properly trained. 



Don Parker is an intrusion detection specialist working for Rigel Kent Security and Advisory Services Inc. Don Parker remains active in the security community by authoring articles and giving briefings to select audiences.