Polymorphic Shellcodes vs. Application IDSs

NEXT GENERATION SECURITY TECHNOLOGIES
http://www.ngsec.com

1. Introduction

This document focuses on how IDS, under certain circumstances, can detect Polymorphic shellcodes.

We will go through the three main parts of a polymorphic shellcode and analyze IDS common problems to detect them.

2. Shellcode types and recognition techniques

We will discuss shellcodes referring to IA32 platforms.

Please note that all these techniques can be implemented on other platforms such as SPARC, HPPA, MIPS, etc.

Before polymorphic appeared, **regular shellcodes** had two well-defined sections:

- **NOP section**: This is the section where the program will jump when a common buffer overflow is successfully exploited. It is just a huge amount of NOP instructions.
- **Shellcode payload**: This is the section where gets executed /bin/sh, binds a shell to a TCP port, etc.

These shellcodes are very easy to detect. On the past years it was very common to use 0x90 instructions (nop) on IA32 platforms in the NOP section. So IDS just searched for an X amount of 0x90s and triggered a shellcode alarm. Also, some IDS have signatures (such as /bin/sh) to detect the shellcode payload.

When **polymorphic shellcodes** appeared these techniques became obsolete.

Polymorphic shellcodes have three well-defined sections:

- **NOP section**: This section now is a random mix of no-effect instructions such as inc %eax, inc %ebx, pop %eax, nop, dec %eax, …
- **Decrypter engine**: This section contains an engine to decrypt the shellcode payload. This engine is not the same from one shellcode to another, it varies randomly using some virii polymorphic techniques. The cipher used normally is a xor mechanism or a double xor mechanism, it could be implemented a better ciphering mechanism such as RJINDAEL, but shellcode would grow in some thousands of bytes and would not be useful anymore in many buffer overflow exploitations.
- **Encrypted shellcode payload**: This section has the original shellcode encrypted.

This kind of shellcode is harder, though not impossible to detect.

There has been some discussion on how to detect such shellcodes:

- **Shellcode payload decrypt and detection with old shellcode payload signatures**: This technique, used by antivirus to detect viral code, can be an approached to detect these shellcodes.

But it has some open issues:

- How do you detect it is an encrypted shellcode payload?
- Which cipher mechanism uses?
- Which key(s) are used in the cipher mechanism?
- Can it be brute-forced in low time?

- **Signatures to detect the decrypter engine**: This technique could be a better approach, but has some problems too:
  - Since decrypter engine mutates and too many instructions are involved, IDS would have to check too many signatures (mask).
  - Lots of false positives.
  - Too many CPU cycles and time needed.

- **Decrypter engine emulation**: IDS is emulating the code so if it finds code that seems to decrypt something in memory it raises a shellcode alarm. This technique raises low number of false positives but has a strong weakness:
  - Too many CPU cycles and time needed.

- **NOPS section detection**: IMHO this is the best technique, it just tries to detect a NOP_NUMBER number of no-effect instructions.

It has some issues such as many false positive, but if you set NOP_NUMBER to a range between 50-60, recognizes almost every shellcode. Weaknesses:

- Lots of CPU cycles.
- False positive when NOP_NUMBER is low.
- Since some non-effect instructions have ascii representation, some character strings such as AAAAA…60times…A, would be recognized as shellcodes.

It is very important to **set NOP_NUMBER to a reasonable value** in order to avoid too many false positives.

Throughout **NGSEC's benchmark test** we found that a number ranging 8**0-90** was a good value to detect shellcodes without too many false positives.

3. Intrusion Detection Systems

There are mainly three types of Intrusion Detection Systems:

- **Network IDS**: This type of IDS grabs datagrams from a network interface and looks for attack patterns in them (such as port scanning, cgi exploitation, etc).
- **Host IDS**: This type of IDS looks for patterns in local user actions; e.g. if a user is trying to view such file as /root/.rhosts, this action could clearly be identified as an attack pattern.
- **Application IDS**: This type of IDS has recently appeared. It just looks for attack patterns to all the input data that comes to the application.

Implementing the **NOPS section detection** on these types of IDS has, as usual, open issues:

- **Network IDS**: Since it has to grab as many datagrams as it can, it can't waste too much time looking at every packet, because it could drop too many datagrams while looking for NOPS. This would cause a big decrease on IDS performance.
- **Host IDS**: This technique can be implemented with these type of IDS's, but there are better ways of "buffer overflow exploitation" detection, since you can watch the flow of the program and see when return pointer is changed.
- **Application IDS**: This is the best kind of IDS to implement this technique. Since all input data is checked, no drop of data is possible, and normally not many data is checked (low CPU and time to check) you can recognize almost all polymorphic shellcodes at input data. You have to set the NOP_NUMBER to a value that fits better with your protocol, and normal data input.

4. Proof of concept: NIDSfindshellcode

As proof of concept NGSEC has developed a free Network IDS (easier to implement for PoC rather than an Application IDS) called NIDSFindShellcode, able to detect Polymorphic Shellcodes using the NOP section detection technique.

Download it at:

http://www.ngsec.com/ngresearch/ngtools/

Here is a sample output, when someone tries to exploit a vulnerable pop3 server with a polymorphic shellcode.

```
piscis:~# nidsfindshellcode -d eth0 -v
NIDS_shellcode 1.0 by Fermín J. Serna <fjserna@ngsec.com>
Next Generation Security Technologies
http://www.ngsec.com

IA32 shellcode found: Protocol TCP piscis:1547 -> blackpill:110
Dumping data:
USER ]B@Z..N7[K....]F..D....T7..RF.EGU@R
.EZ_....F7.KXLWP.V..G..@_JOZSJ..MD@X/..E
`^_SE.I]F..H_..ZO^.ZL'H.D.7^TGLL.F..YFWK
..?.PU.Q/J_MQJR.]DYVC'/E./DZ..^M.QM.^...
L...A.?[MCKKE^....Y.^L^V.[.^?`JCUY..EPC_
...N.T.R.J.KIF..C.7DWWPTN.O.JBD.`MNI^L/M
.].?/.U].^?XK.K.I.O7.

piscis:~#
```

NGSEC has also implemented the NOP section detection technique on our Application IDS and Firewall for Web Servers: NGSecureWeb®.

5. References

[1] NGSecureWeb® at http://www.ngsec.com/ngproducts/ngsw/
[2] NGTools at http://www.ngsec.com/ngresearch/ngtools/
[3] ADM mutate at http://www.ktwo.ca/security.html
[4] Focus-IDS at http://www.securityfocus.com/

6. Credits

This document was brought to you by:

Fermín J. Serna < **fjserna@ngsec.com** >
Chief Technology Officer / NGSEC
Next Generation Security Technologies
http://www.ngsec.com

labs@NGSEC < **labs@ngsec.com** >