CS 6262: Network Security - Assignment 3

Manish Choudhary

February 20, 2014

1. **Lab Setup**

   The lab setup was done as per the instructions. The image of the virtual machine was downloaded and three replicas were created. VMWare Workstation was used as the hypervisor to host these virtual machines. The details of the three instances are:

   - Instance 1 - Attacker

     The instance one was the attacker machine with metasploit running on it. Metasploit was already installed on it and the default configuration was used. The IP address of this machine was 192.168.139.152. The default reverse handler was used with following commands to launch the attack through msfconsole.

     ```
     use exploit/linux/ftp/proftp_telnet_iac
     set TARGET 4
     set RHOST 192.168.139.156
     exploit
     ```

   - Instance 2 - Monitor

     The second instance was the monitor with snort installed on it and with IP address 192.168.139.151. The snort.conf file was adapted to create my_snort.conf file. A sample copy of the configuration file has been submitted with the report. The major changes in the configuration file were following. Some variables were added which were used in the rules.

     ```
     ipvar FTP_SERVER 192.168.139.156
     ipvar EXTERNAL_NET !$FTP_SERVER
     portvar FTP_CONTROL 21
     portvar FTP_DATA 20
     portvar EXTERNAL_PORT any
     ```

     `include $RULE_PATH/my_ftp.rules` line was added to include the rule file in my_snort.conf. All the rules developed were maintained in my_ftp.rules rule file.

     `snort -d -l ./log -c /etc/snort/my_snort.conf` command was used to run snort directing it to use the specific configuration file and and to log the alerts under the log subdirectory.

   - Instance 3 - FTP Server

     The instance three was made the FTP server with proFTPD installed on it and with the IP address 192.168.139.156. The default FTP service running on the VM was vsFTPD. So, that service was stopped to run the vulnerable version of the proFTPD on it.

   The three virtual machines were put in a private network with the host. To achieve this, the host-only option of VMWare Workstation was used that connects the virtual machines and the host in a private network isolating this network from the internet. So, this ensured that the virtual machines were in an isolated network and didn't affect the external network in any way.

2. **Detecting All FTP Traffic**

   The rule developed for detecting the FTP traffic is:

   - Version 1:
     $alert\ tcp\ \$EXTERNAL\_NET\ \$EXTERNAL\_PORT\ ->\ \$FTP\_SERVER\ [\$FTP\_CONTROL,$
     $\$FTP\_DATA]\ (flow: to\_server, established; session: printable; msg: "FTP\ Traffic"; sid:$
     $1000001; rev: 1;)$

     The rule would alert and log in case it matches for some packet. FTP service runs over TCP and thus, it is basically a TCP flow.

     The rule has been designed to detect all the traffic arriving at port 21 and port 22 of the machine running the FTP server. Port 21 is the port generally used by the FTP service to establish the control channel with the client. The client requests the server to start a TCP session on this port and to use this channel to transfer the commands. So, all the requests coming to the server from the client would be destined to port 21. In addition, port 20 is used by the server to transfer/receive the data from the client. So, any data to be uploaded to the server would generally arrive at the port 20. This would hold only in the case of active FTP mode in which the server initiates the TCP connection for the data channel. In case of passive mode, the client sends PASV command via control channel to receive server's IP address and a port number to which the client initiates a TCP session for the data channel. In passive mode, server can choose any arbitrary port to establish data channel. So, we would have to replace 'FTP_DATA' with 'any' in the rule. But, it may generate false positives and thus, to find an intermediary solution it has been assumed that the session would be in active mode.

     If the requirement is to support FTP service on any port(can be configured by the administrator to run on ports other than 21 and 20), the FTP ports can be changed to 'any', but it may introduce a lot of false positives.

     The flow option has been used to make the rule applicable to certain direction of the TCP flow. Only the traffic to the FTP server in a client initiated and established session would be considered for the scrutinization.

     The session option has been used to extract the data from the packet. It would log all the printable strings present in the packet data. It can be removed in actual monitoring environment if it is not required.

     "FTP Traffic" would be the name of the corresponding alerts logged in the log file. The ID for this rule is 100001 and the revision version is 1.

     FALSE POSITIVES:

     The expected false positives would be the alerts in the case port 21 and 20 are used for some other purpose. Generally, these ports are used by the FTP service but it is possible to use them for some different operation. In that scenario, the rule would give a false alarm.

     FALSE NEGATIVES:

     There would not be any false negative if the IDS is able to process all the packets, matching the speed of the packet transfer.

   - Version 2

     In case the requirements are strict and the IDS is used in an environment where false positives may be high, following modified rule can be used to detect the FTP traffic.

     $alert\ tcp\ \$EXTERNAL\_NET\ \$EXTERNAL\_PORT\ ->\ \$FTP\_SERVER\ \$FTP\_CONTROL$
     $(flow: to\_server, established; pcre: "/(ABOR|ACCT|ADAT|ALLO|APPE|AUTH|$

$CCC|CDUP|CONF|CWD|DELE|ENC|EPRT|EPSV|FEAT|HELP|LANG|LIST|LPRT|$
$LPSV|MDTM|MIC|MKD|MLSD|MLST|MODE|NLST|NOOP|OPTS|PASS|PASV|PBSZ|$
$PORT|PROT|PWD|QUIT|REIN|REST|RETR|RMD|RNFR|RNTO|SITE|SIZE|SMNT|$
$STAT|STOR|STOU|STRU|SYST|TYPE|USER|XCUP|XMKD|XPWD|XRCP|XRMD|$
$XRSQ|XSEM|XSEN)/"; \ session : printable; msg : "FTP\ Traffic"; sid : 1000001; rev :$
$2;)$

All the fields are same except the regex pcre and that we are just monitoring port 21. All the known FTP commands have been included in the rule to check if any of these commands are present in the packet. This rule would result in a reduced performance because of the pattern matching but would provide more strict check. So, these different versions can be used as per the requirements. If the FTP service is running on an arbitrary port, the FTP_CONTROL can be assigned value 'any' so that traffic at any port would be monitored. But, it may result in a lot of false positives.

FALSE POSITIVES: In case same commands are used in a non-FTP communication either as commands or normal data.

FALSE NEGATIVES: No false negative until some new FTP command is introduced. Only the traffic to the port 21 would be monitored and thus, data arriving at port 20 or any other port wouldn't be considered for match.

3. **ProFTPD Attack**

The rule developed to detect an attempt to exploit the ProFTPD 1.3.2rc3 - 1.3.3b Telnet IAC Buffer Overflow vulnerability is:

$alert\ tcp\ \$EXTERNAL\_NET\ \$EXTERNAL\_PORT\ -> \ \$FTP\_SERVER\ \$FTP\_CONTROL$

$(flow : to\_server, established; \ content : "SITE"; \ nocase; \ within : 10; \ content : "|FF\ FF\ FF|";$

$offset : 125; \ session : all; \ msg : "ProFTP\ TELNET\ IAC\ Vulnerability"; \ sid :$
$100002; \ rev : 1; \ reference : cve, CVE - 2010 - 4221;$

$reference : url, bugs.proftpd.org/show\_bug.cgi?id = 3521)$

The logical reasoning behind this rule is:

- Alert: To alert and log the attempts detected
- tcp: FTP runs over TCP and thus, TCP sessions have to be monitored to detect the exploit.
- direction : The vulnerability can be exploited by sending large number of TELNET IAC Commands to the server. Thus, the traffic coming from the client to the server needs to be monitored only.
- The client can have any IP address and can try to connect via any port.
- The Server has specific IP address in this exercise. In general, "FTP_SERVER" should refer to the IP address or the list of IP addresses on which FTP service has to be monitored.
- The port number considered in this rule is 21 which is used generally by the FTP service to establish the control channel. This would be fine in the scenarios where the service is configured to run on port 21. It is also possible to run the service on some different available port and thus, "FTP_CONTROL" variable should have value "any" to cover those cases. We just need to monitor the control channel and not the data channel as the exploit sends request to server which is communicated via control channel always.

- flow: It has already been covered in the last section. We are trying to monitor the traffic from the client to the server on the established TCP sessions initiated by the client.
- SITE Search in Content: The FTP command ”SITE” is used for site specific commands and may vary from server to server. It has been observed in the exploit and strengthens the rule to avoid false positives. If there is a version of exploit that can work without this command, the SITE search can be removed as other parameters would take care of the detection logic.
- nocase: It directs the engine that the content search for ”SITE” should be case insensitive.
- within: SITE appears in the start of the data and thus, it is sufficient to look for it within the first 10 bytes only.
- Search for ”FF FF FF” in Content: This is the important part of the rule. The vulnerability description says that the buffer overflow can be exploited by large number of IAC commands. The number of IAC commands can vary across the different exploits. IAC(Interpret as Character) command is 255 (FF in hex) and is used to differentiate the TELNET commands from other data. So, any command or sequence of commands starts with IAC command. Looking just for ”FF” would be too specific and may result in lot of false positives. Such exploit itself won't be reliable and thus, is not a practical approach used by the malicious authors. In some cases, it is possible to use 255 or FF(even though not ASCII) as the optional 8-bit binary data if the client and the server have negotiated on it. So, looking for ”FF FF” may also give false positives in such scenarios. That is the reason for choosing ”FF FF FF” as the search pattern as it would be rare to have this in the genuine requests. The metasploit exploit uses a long sequence of IAC commands and therefore, we could have searched for more than three ”FF”. But,there may be some authors developing the exploit with lesser number of IAC commands taking the risk of failure(because of some miscalculation) to evade detection by any IDS in place.
- Offset 125: Offset 125 directs the engine to start the content search after the first 125 bytes in the payload. The reason for this is the actually vulnerability description. bugs.proftpd.org describes the bug in the following way. ”The flaw exists within the proftpd server component which listens by default on TCP port 21. When reading user input if a TELNET_IAC escape sequence is encountered the process miscalculates a buffer length counter value allowing a user controlled copy of data to a stack buffer.”

  So, interesting point is how and when it miscalculates the counter value.

  In the source code of proFTPD, FTP commands are read by function pr_cmd_read() present under src/main.c source file. At one point, the function pr_cmd_read() calls the function pr_netio_telnet_gets() which has following declaration(defined under src/netio.h).

  $char * pr\_netio\_telnet\_gets(char * buf, size\_tbuflen, pr\_netio\_stream\_t * in\_nstrm,$
  $pr\_netio\_stream\_t * out\_nstrm);$

  Here, the second parameter ”buflen” is the one that introduces the vulnerability. The value passed to this is one less than the size of a local buffer used by pr_cmd_read() function.

  $$char\ buf[PR\_DEFAULT\_CMD\_BUFSZ + 1]$$

  According to the bug log for Version 1.3.3, this parameter value is calculated as:

  $(PR\_DEFAULT\_CMD\_BUFSZ + 1) - 1\ =$
  $PR\_DEFAULT\_CMD\_BUFSZ\ =\ [definedinsrc/main.c]$
  $PR\_TUNABLE\_PATH\_MAX\ +\ 7\ =\ [definedininclude/options.h]$
  $MAXPATHLEN\ +\ 7\ =\ [on\ Linux,\ MAXPATHLEN == 4096]$

  The logic decrements the value of this parameter and reads each character until the the buflen becomes equal to zero or newline character is encountered. But, the function again decrements this value in the same iteration if a TELNET IAC command is found. So, buflen is decremented twice in an iteration effectively. Let's assume the buffer length is 50. It will decremented it from 50 to 49,48, 47 and so on. When the value is 1, if it encounters IAC command, it decrements buflen by 2 and thus, it becomes -1. So, it continues the loops from -1 to -2, -3, -4 and so on until a newline character is encountered. Therefore, the key observation is that the IAC command should be placed in the payload such that when the buflen is 1, it should encounter IAC command decrementing the value by 2 and allowing the buffer to overflow.

This parameter value is around 4K on Linux and same value is considered in the exploit for the the version is 1.3.3. So, the offset can be defined to be around 2000(as a sequence of IAC characters can decrement buflen faster).

But, the vulnerable version given in the exercise is 1.3.2 and we have following under include/option.h:

# Maximum path length. GNU HURD (and some others) do not define MAXPATHLEN. POSIX' #PATH_MAX is mandated to be at least 256 (according to some), so 1K, in the absense of #MAXPATHLEN, should be a reasonable default

$\#ifndefPR\_TUNABLE\_PATH\_MAX$

$\#ifdefMAXPATHLEN$

$\#definePR\_TUNABLE\_PATH\_MAXMAXPATHLEN$

$\#else$

$\#definePR\_TUNABLE\_PATH\_MAX1024$

$\#endif$

$\#endif$

So, this option has been commented which can be used by administrator to tune the size. pr_cmd_read() function has following buffer:

char buf[PR_TUNABLE_BUFFER_SIZE]

The PR_TUNABLE_BUFFER_SIZE value is taken automatically from the system. This value may vary from platform to platform and also depends upon the configuration. On Linux, this is 4096.

After all this research about the vulnerability and correct offset value, the rule has been tested initially with the offset value 1000 and found to be working fine, detecting all the attempts to exploit. The exploit also considers this value to be around 1000 for version 1.3.2 and ubuntu platform. But, to generalize the rule in an attempt to cover different platforms, the offset value has been changed to 125 as 256 is the minimum requirement for POSIX PATH_MAX and if a sequence of IAC characters is used, each IAC character would decrement the buflen by 2. Therefore, 125 characters can decrement the value by 250. Thus, it is the value closer to minimum offset that should be considered.

This can provide a great performance boost as we can skip many initial bytes. This value can be fine-tuned on the basis of the proFTPD version and the platform hosting the FTP service(1000 if we just want to cover the environment set up for the exercise). It can also help to reduce false positives as we are not checking some initial bytes.

- session: To log all the data in the payload. This can be used as per the requirement and can be removed if the payload is not required.
- msg: It is sort of a name for the rule which would be printed in the logs.
- sid and rev: These have already been discussed above.

- reference: It is used to provide references where addition information can be found and plays no role in the detection logic.

FALSE POSITIVES: It may generate false positives in a scenario when ”FF FF FF” and ”SITE” are used in some genuine request with size greater than 125 bytes.

FALSE NEGATIVES: It may generate false negatives if there is a way to exploit without ”SITE” command. To counter this, the search for ”SITE” can be removed from the rule. False negatives would also be there if the buffer size is less than 125 as we are skipping the first 125 bytes. It can be tuned as per the environment to be monitored. As we are monitoring the traffic on port 21, it would not detect the exploit in case FTP service uses some other port. But, generalizing the rule by doing all this would increment the chances of generating false positives.

This rule provides a fair balance between false positives and false negatives.

NOTE: The rule has been developed targeting the vulnerability. Another approach would be to search for a sequence of bytes used in the exploit payload. But, it makes the rule specific to the exploit. As various versions of the exploits can be developed using different techniques (return to LibC, ROP and others), it may not be a good idea to develop the detection logic on the basis of the particular technique used by the exploit. So, it has been tried to make it applicable to different versions and environments.

4. **Revised ProFTPD Signature**

The revised signature to detect the exploit only for vulnerable versions is:

$alert\ tcp\ \$FTP\_SERVER\ \$FTP\_CONTROL -> \$EXTERNAL\_NET\ \$EXTERNAL\_PORT$

$(flow : established;\ msg : "Candidate\ ProFTPD\ TELNET\ IAC\ Vulnerability";\ pcre : "/ProFTPD\ 1.3.(2|3)?(r|a|b|$
$s)?/";\ flowbits : set, exploit;\ sid : 100003;\ rev : 1;\ flowbits : noalert;)$

$alert\ tcp\ \$EXTERNAL\_NET\ \$EXTERNAL\_PORT -> \$FTP\_SERVER\ \$FTP\_CONTROL$

$(msg : "Specific\ ProFTPD\ TELNET\ IAC\ Vulnerability";\ flowbits : isset, exploit;\ content : "SITE";\ nocase;\ within : 10;\ content : "|FF\ FF\ FF|";\ offset : 125;\ session : all;\ sid : 100004;\ rev : 1;\ reference : cve, CVE - 2010 - 4221;)$

In the revised version, 2 rules have been developed to monitor the flow across the packets.

Rule 1 has been created to check the version of the proFTPD service running on the server so that the our detection mechanism works only for the vulnerable versions. The FTP server sends the version information to the client in a packet. We monitor the traffic from the server to the client and search for the vulnerable versions in the packet. There are three vulnerable versions 1.3.2rc3, 1.3.3a and 1.3.3b which are searched in the packet. pcre option has been used to define the regex. In the exercise environment, ”220 ProFTPD 1.3.2 Server (ProFTPD Default Installation)” goes in the packet. So, ”220 ProFTPD 1.3.2 ” is the candidate to be matched. If the match is found, the flowbit variable ”exploit” is set and the alert gets logged with the name ”Candidate ProFTPD TELNET IAC Vulnerability” and sid value 100003(in case we don't use flowbits noalert option). But, we would not prefer it to be logged for each match for the version in actual monitoring environment as it may happen very frequently even in genuine cases. So, we can use flowbits noalert option to avoid this. On version match, it would set the flowbit ”exploit” but won't log it.

Rule 2 is almost same as the one discussed in the last section except the flowbits option. It checks if the bit ”exploit” is set and if yes, then further options are checked for. If the packet with the attack payload is found, it gets logged with the name ”Specific ProFTPD TELNET Vulnerability” and sid value 100004.

FALSE POSITIVES: No new false positives would occur because of the revised version. As the search for the version is very specific, it is not expected to produce false positives. The rule 2 would generate the false positives in case ”FF FF FF” and ”SITE” are present in some genuine request.

FALSE NEGATIVES: As per the manual, flowbits is used to remember the state across the packets for a TCP session. So, if the packets with version information and the payloads are part of different sessions, rule would not trigger. The rule has been tested and found to be generating alerts(without noalert option to check that both are working fine) as shown below:

```
[**] [1:100003:1] Candidate ProFTPD TELNET IAC Vulnerability [**]
[Priority: 0]
02/20-20:39:58.142793 192.168.139.156:21 -> 192.168.139.152:41260
TCP TTL:64 TOS:0x0 ID:30050 IpLen:20 DgmLen:127 DF
***AP*** Seq: 0xB017DF5A Ack: 0x7DBF70D7 Win: 0x72 TcpLen: 32
TCP Options (3) => NOP NOP TS: 32683009 90671802
[**] [1:100004:1] Specific ProFTPD TELNET IAC Vulnerability [**]
[Priority: 0]
02/20-20:39:58.143736 192.168.139.152:41260 -> 192.168.139.156:21
TCP TTL:64 TOS:0x0 ID:64101 IpLen:20 DgmLen:1310 DF
***AP*** Seq: 0x7DBF70D7 Ack: 0xB017DFA5 Win: 0x73 TcpLen: 32
TCP Options (3) => NOP NOP TS: 90674308 32683009
```

If we use flowbits noalert option, there won’t be any log for the rule with sid 100003. Following gets logged in such scenario:

```
[**] [1:100004:1] Specific ProFTPD TELNET IAC Vulnerability [**]
[Priority: 0]
02/21-11:15:40.663132 192.168.139.152:32881 -> 192.168.139.156:21
TCP TTL:64 TOS:0x0 ID:28017 IpLen:20 DgmLen:1310DF
***AP*** Seq: 0xBFDA7AE Ack: 0xB7A955A8 Win:0x73 TcpLen:32
TCP Options (3) => NOP NOP TS: 101635214 43643911
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4221]
```