Student's name: Matheus Henrique Schaly

Professor's name: Carlisle Adams

Course: CSI 4139 Design of Secure Computer Systems

Date: 23rd September, 2019

Report #1

First Part

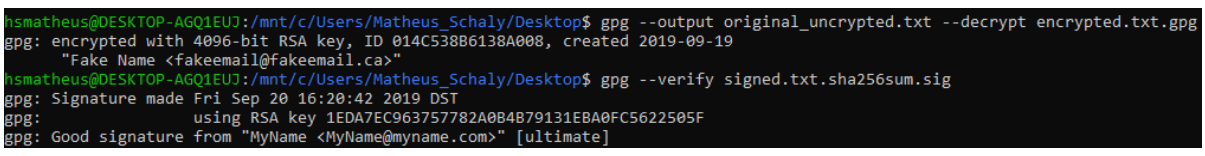
The file protection program used was GnuPG (GPG). Command line itself was used to generate the keys, sign, encrypt, verify and decrypt. The GPG has already had some vulnerabilities. The most recent was around June 2018, when the SigSpoof attackes were announced. These allowed an attacker to convincingly spoof digital signatures. Nonetheless, GPG is still greatly reliable, thus, is a good tool to be used [1].

These were the steps [2]:

1) Generate a GPG keypair: The command *gpg –full-generate-key* was used to create the GPG keypair. The algorithm used, due to its reliability and widely usage, was the RSA (Rivest-Shamir-Adleman), which is an asymmetric cryptographic algorithm [3]. In RSA, this asymmetry is based on the practical difficulty of the factorization of the product of two large prime numbers, the "factoring problem" [4]. The keysize used may vary from 1024 to 4096 bits long. RSA claims that 1024-bit keys are likely to become crackable some time between 2006 and 2010 and that 2048-bit keys are sufficient until 2030. The NIST (National Institute of Standards and Technology) recommends 2048-bit keys for RSA [5]. Therefore, to be on the safe side, 3072 (which is the default) keysize was used.

2) Make your GPG public key available to the other party: After generating the keys, the person that will receive the message must have access to our public key, so that he can verify that we were the actual sender. Thus, we use *gpg --armor --output mypubkey.gpg --export matheusmhs@hotmail.com* to export our public key to a file.
3) Retrieve the message recipient's public key: In order to encrypt the file, we need to have the recipient's public key. So, we use *gpg --import recipient_public_key.gpg* to add it to our keyring.
4) Encrypt the message: Then we encrypt the message using the sender's public key *gpg –output encryted.txt.gpg –-encrypt –-recipient your.friend@domain.ca unencrypted.txt*.
5) Sign the message: Instead of signing the message, a checksum of the message was generated and singed. To do so, it was used *shasum -a 256 uncrypted.txt | awk '{print $1}' >checksum.txt.sha256sum*. And also *gpg --output signed.txt.sha256sum.sig --sign checksum..txt.sha256sum*.
6) Decrypt the message: In order to the receiver decrypt the message, it was used the *gpg –output original_uncrypted.txt –decrypt encrypted.txt.gpg* command.
7) Verify the signature of the message: Finally, to verify the authenticity of the sender, the code *gpg –verify signed.txt.sha256.sig* was used.

The following image shows the final of the process, after creating the keys, encrypting, and signing. Actually, by mistake, a RSA keysize of 4096 bits was used. This is not actually necessary (in our situation) as mentioned at the beginning. Moreover, the e-mails and names used were fake ones.



References

[1] Wikipedia. (2019). *GNU Privacy Guard*. [online] Available at: https://en.wikipedia.org/wiki/GNU_Privacy_Guard [Accessed 20 Sep. 2019].

[2] Han, P. (2019). *How to use GPG to encrypt stuff - Pang Yan Han's blog*. [online] Pang Yan Han's blog. Available at: https://yanhan.github.io/posts/2017-09-27-how-to-use-gpg-to-encrypt-stuff.html [Accessed 20 Sep. 2019].

[3] Wikipedia. (2019). *RSA algorithm*. [online] Available at: https://simple.wikipedia.org/wiki/RSA_algorithm [Accessed 20 Sep. 2019].

[4] Wikipedia. (2019). *RSA (cryptosystem)*. [online] Available at: https://en.wikipedia.org/wiki/RSA_(cryptosystem) [Accessed 20 Sep. 2019].

[5] Wikipedia. (2019). *Key size*. [online] Available at: https://en.wikipedia.org/wiki/Key_size [Accessed 20 Sep. 2019].

Second Part

In order to run both the virtual machines we used the Oracle VM Virtual box. By using that software we could create two OS. The first one, Ubuntu, was the victim, whereas the second one, Kali, was the attacker. Then, to get the ip address of both the Ubunto and Kali we used *ipconfig* in the Ubunto OS (Figure 1).

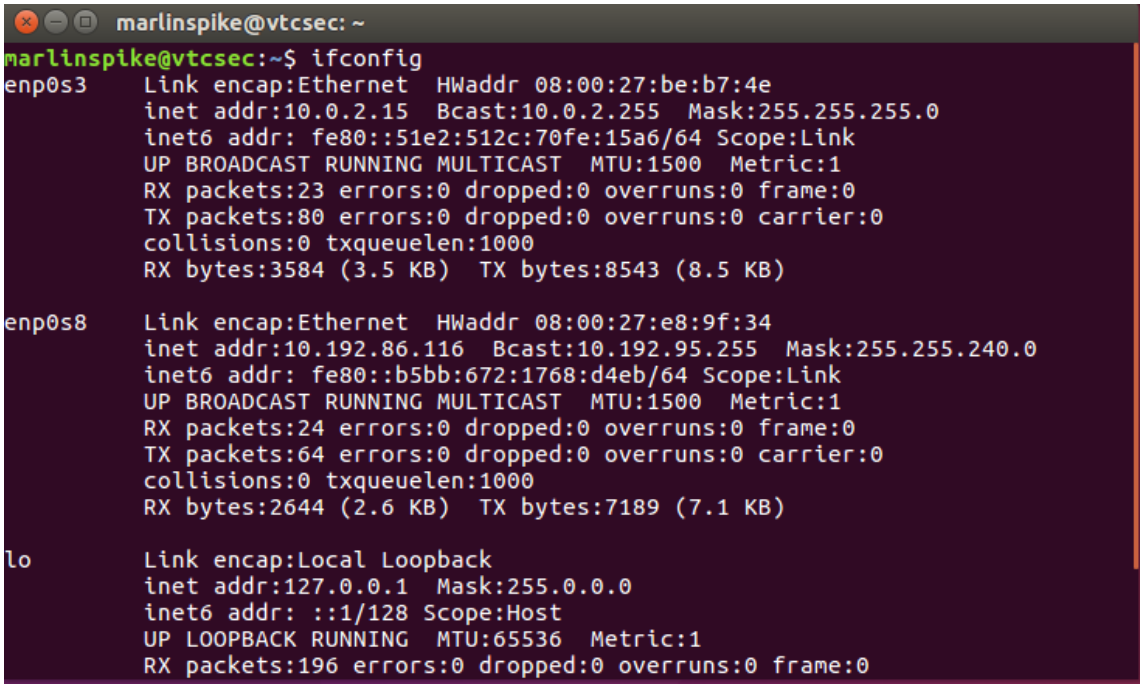


Figure 1 Using ifconfig on the Ubunto OS

To be able to exploit the Ubunto OS, we used the Metasploit Framework, a modular penetration testing platform that enables writing, testing and execution of exploit codes. To have access to the Metasploit Framework, we used the MSFconsole, which provides a command line interface to access and work with Matasploit [1]. To find the exploit, we ran *nmap -A*. Such command is an aggressive scan. It enables OS detection (-O), version scanning (-sV), script scanning (-sC) and traceroute (--tracerout). The point is to enable a comprehensive set of scan options without having to remember a large set of flags. So, instead of using the other (simpler) scans, we chose to use -A. However, it should be used with caution, only using it in networks that you have permission [2]. After running *nmap- A* we detected that the port 21 was open in

Ubunto. Furthermore, we could also see the service (ftp) and the version (ProFTPD 1.3.3c) (Figure 2).

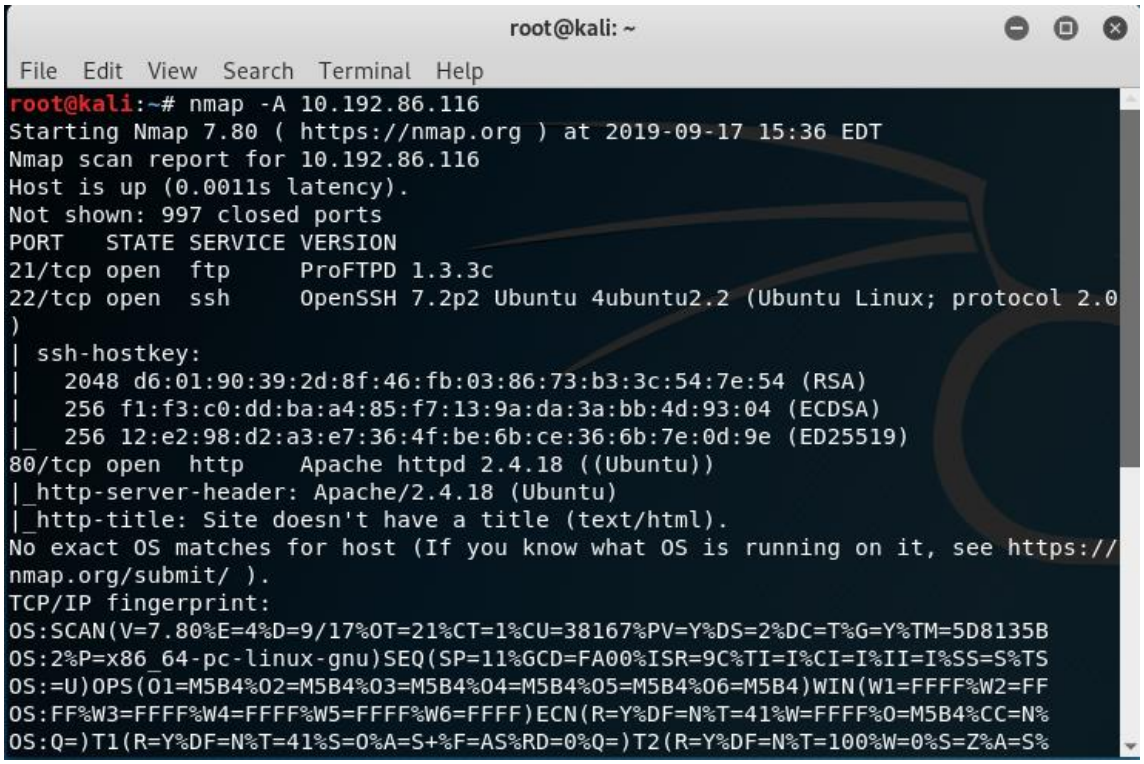


Figure 2 The port 21 is open in Ubunto

To exploit the vulnerability the command used was *use exploit/unix/ftp/proftpd_133c_backdoor*. Then the remote host, local host, local port (a random port), payload and target were set. A payload is the shell code that runs after an exploit successfully compromises a system. It enables you to define how you want to connect to the shell and what you want to do to the target system after you take control of it [2]. Finally *exploit* was executed, and we could take control of the Ubunto OS (Figure 3).

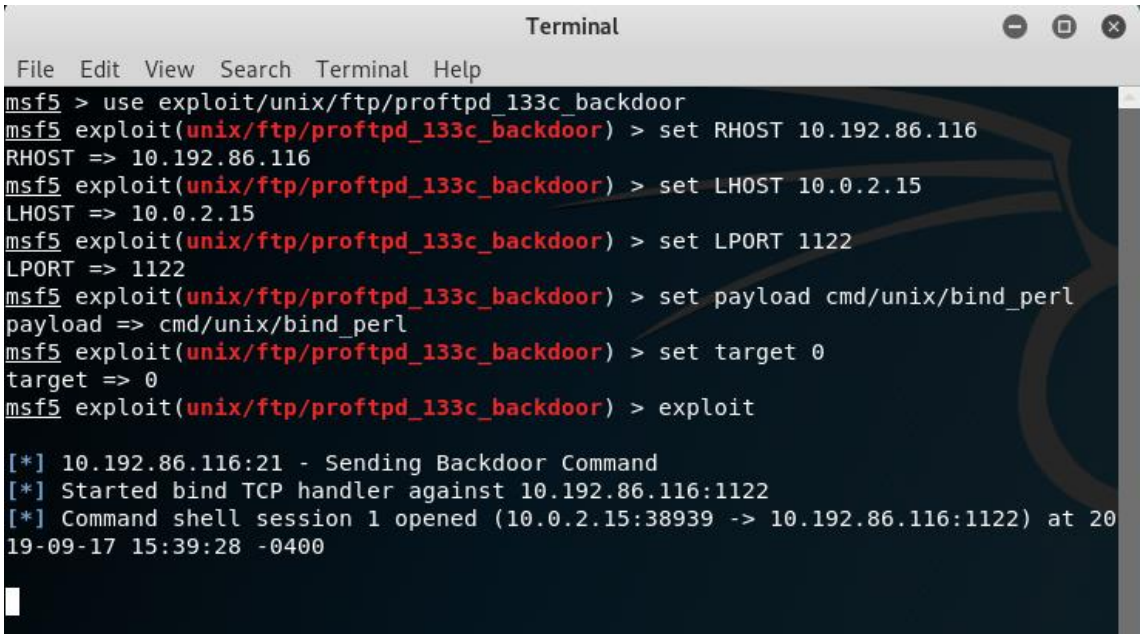


Figure 3 Setting payload and taking control of Ubunto

Updating the system to a newer version solves the vulnerability. For FortiGate IPS user, turning on the following IPS signatures can prevent exploitation of these vulnerabilities [3]. It is also possible to detect and prevent this kind of attack with TippingPoint and the filter 10641 [4]. Furthermore, to prevent clients from exploiting this vulnerability while a fix is was being deployed, the following a certain code could have been used in the proftpd.conf file [5].

[1] Rapid7. (2019). *Metasploit Framework*. [online] Available at: https://metasploit.help.rapid7.com/docs/msf-overview [Accessed 21 Sep. 2019].

[2] die.net. (2019). *nmap(1) - Linux man page*. [online] Available at: https://linux.die.net/man/1/nmap [Accessed 21 Sep. 2019].

[3] FortiGuard. (2019). *ProFTPD.Prior.to.1.3.3c.Multiple.Vulnerabilities | IPS*. [online] Available at: https://fortiguard.com/encyclopedia/ips/32568/proftpd-prior-to-1-3-3c-multiple-vulnerabilities [Accessed 21 Sep. 2019].

[4] Vuldb. (2019). *PROFTPD 1.3.2/1.3.3 TELNET NETIO.C PR_NETIO_TELNET_GETS MEMORY CORRUPTION*. [online] Available at: https://vuldb.com/?id.55410 [Accessed 21 Sep. 2019].

[5] Sourceforge.net. (2019). *ProFTPD Server Software / Re: [Proftpd-user] ProFTPD 1.3.3c released!*. [online] Available at: https://sourceforge.net/p/proftp/mailman/message/26515227/ [Accessed 21 Sep. 2019].