The Hacker's Sandbox 1.9

Mission Pack Scripting

Written by Unix-Ninja

An Overview

To make The Hacker's Sandbox (THS) more flexible, its engine has been designed to be extendable through Lua script. Using Lua, you can access and manipulate most features of the engine, crafting your own Mission Packs, and creating rich, interactive worlds for players to explore and learn with.

The basic component of any Mission Pack is the Virtual Machine (VM). These are the simulated Unix-like boxes players will interact with directly and attempt to hack. Every Mission Pack must have at least ONE VM in order for the engine to properly load it.

Once loaded, you can use various functions to manipulate the VM before creating another one.

The only limit to the number of VMs you can create is the amount of RAM the player's machine can support. Each VM takes a fairly small amount of space, so you can realistically have many many VMs before you come into a problem.

An Example

To illustrate a working Mission Pack, we will build a short “hello, world” for our engine. Then, I will describe each part of the example below.

```
1 -- THS Hello World!
2 -- A demonstration
3 -- Written by Unix-Ninja
4
5 newVM("localhost")
6 addUser("bob","password")
7 addFile({ name = "/home/bob/hello", content = "Hello, Bob!\n" })
8 mail({ to = "bob", from = "an email", subject = "Your test
  message", body = "Can you see me?\n" })
```

The first 3 lines are Lua comments. Use these to create annotations or blocks of information that the THS engine will ignore.

Line 4 is blank, and will be ignored by the engine.

Line 5 creates a new VM with the name "localhost".

Line 6 creates a new user named "bob", and gives him the password "password". This password will automatically be registered in the global rainbow table. It then automatically creates a home directory called "/home/bob/".

Line 7 creates a file named "hello" in the directory /home/bob/. It then places the content into the file, which can later be viewed inside the engine with utilities like *cat*.

Line 8 will send a mail message to user "bob" on the currently active VM (in this case "localhost"). The message will contain a from, subject, and body message as specified on that line.

Of course, your Mission Pack can become far more complex than this. With a little bit of imagination, you can script custom utilities and sequences to re-create just about any scenario.

As an aid to those learning Mission Pack scripting, the default Mission Pack "Destiny" has been left uncompiled and will run in interpreted form. This should make it easier for you to hack apart the code and see how various real-world challenges can be accomplished within the scripting engine.

Please note: Only the base, string, and math Lua libraries have been enabled in the engine. Other Lua libraries will not currently be available.

The Hacker's Sandbox API available to Lua script

Below is a listing of the API functions available to Lua script. Any function with "{}" specified as an argument takes a Lua table as its only argument. Often, not all of the fields in these tables are required to be passed. Required fields will be underlined for clarification.

addDir ({ })

This function will add a directory to the currently active VM. It takes the following arguments:

acl	Unix access control list. (i.e. 777)
name	Name of the directory to be created

addDNS (name, record)

Insert A record entries into the global DNS map.

getDNS (name)

Retrieve the record for a specified entry from the global DNS map.

addFile ({ })

Add a file to the currently active VM. The parent directory for this file must exist in order for the file to be accessible. If an ACL is not specified, 644 will be defaulted. It takes the following arguments:

acl	Unix access control list. (i.e. 777)
name	Name of the file to be created
content	The content of the file to be seen when using cat
exec	If specified, this is the name of a custom Lua function the engine can run when the file is executed. A few builtin procedures are available (p_*) to be used as well. These will be described below
on_delete	If specified, this is the name of a custom Lua function the engine can run when the file is deleted

addGPU (description, power)

Add a virtual GPU compute device to the active VM. It takes the following arguments:

description	A short description of the card. This will appear in lspci when the machine is queried.
power	The compute power of the device. Devices with a higher computer power will perform compute operations faster. The cumulative power of all compute devices on your VM will be used when determining compute operations.

addNetDomain ([id|name])

Use this to add membership of a VM to the specified Network Domain. You can pass either a Domain ID or a Domain Name to this function.

addService ({ })

Add a networking service to the currently active VM. It takes the following arguments:

port	Port number the service will run on
name	Name of the service daemon
exec	
poll	This is the string that will be returned when the service is polled. (i.e., when using portscan against the VM)
start	If set to “true”, the service will begin started

addUser (username, password)

This will add a user with the specified username and password to the active VM. When a user is added, the password is registered with the global rainbow table, and a home directory is automatically created in the form of: /home/<*username*>/

cwd ()

This function will return the current working directory of the active VM.

echo (...)
This function is similar to Lua's "print" function, except that it will not print a trailing newline character in the output.

garbage ([size])
Return a string of garbage characters. If you can pass an integer for the size, which will return a string of the specified number of characters.

getCWait (score)
Retrieve the number of seconds the compute system thinks you should sleep for the currently active VM based on the compute score given. You can pass the results of this function to sleep().

hash (string)
Return the vHash representation of the string passed to it. This function will register the plain with the global rainbow table.

hostname ()
This function will return the hostname of the currently active VM.

inNetDomain (hostname, [id|name])
Returns a boolean value indicating if the given hostname is a member of the specified Network Domain. You can pass either a Domain ID or a Domain Name to this function.

input ()
Prompt the user for input and returns the input as a string.

isDir (string)
Using this function will return a boolean value. If the string entered exists **and** is a directory, it will return true.

isFile (string)
Using this function will return a boolean value. If the string entered exists **and** is a file, it will return true.

login ({ })
This function will attempt to log the player into a specific user account on a given VM.

host	Name of the VM to log in to
username	The username of the user to log in to
password	If specified, it will check to make sure the given password matches the user's password before allowing the player to log in. If not specified, the engine will attempt to log the player in to the account without requiring a password.

logout ()

Logs the user out of whatever the topmost account in the shell stack is for the current VM.

mail ({ })

Used to send a mail message to a given user. It takes the following arguments:

to	The user to send the message to. If just the username is give (i.e. "root"), then the mail will be sent to the user on the currently active VM. You can direct the message to a particular VM by specifying the route in the username via the "@" character. (i.e. to send a message to root on a VM named "jupiter", you would write the to field as: "root@jupiter")
from	The name of the source of the message
subject	A subject for your message
body	The content of the message to be seen in the mail app

md5 (string)

Returns a hex-encoded MD5 hash of the string passed to this function.

newVM (hostname)

Create a new VM with the given hostname. Once specified, it becomes the active VM and all functions affecting the active VM will affect this machine. Once the game begins, the first machine is reselected as the active machine. From that point, the active machine is any machine the player is logged into.

pause ()

Used to pause the game engine until keyboard input is detected.

quit ()

This function will end the game and leave the engine.

readfile (filename)

Grab the content of the file with the given filename on the active VM.

serviceRunning (hostname, servicename)

Returns a boolean value of the state of the given service on the VM specified by hostname. True means the service is currently running. False means the service is not currently running.

setProperty ({ })

This function will set VM properties for the currently active VM. It takes the following arguments:

uname	Sets the uname of the VM
ip	Sets the IP address of the VM (VMs are not reachable over a network connection unless they have a unique IP address.) When an IP is added, the machine's hostname is automatically added to the global DNS map with this IP address.

on_root	If specified, this is the name of a custom Lua function to run when the player has achieved root access on the active VM. This will not trigger more than once per VM.
on_login	Calls the specified function whenever the player logs into an account on the VM. This function will trigger whenever a successful login is detected.
on_logout	Calls the specified function whenever the player logs out of an account on the VM.
hint	Set the text to be displayed when the player types "hint" while logged into this VM.
netd	Set the default Network Domain for this VM (additional Network Domains can be added later.) If this option is not set, the VM is automatically added to Network Domain 1 (called "default").

sleep (time)
Pauses the running engine for the number of seconds specified in time.

startService (servicename)
This will start the service with the given service daemon name on the active VM.

stopService (servicename)
This will stop the service with the given service daemon name on the active VM.

timestamp()
Return the current Unix timestamp.

Builtin Exec Procedures

These procedures can be passed to the exec param when adding files to a VM.

p_hashcat
This procedure links to the built-in virtual hashcat instance. The virtual hashcat program uses a dynamically generated global rainbow table to perform lookups against the users you have added to a mission's VMs, and will return a match of the plain text password if found.

p_portscan
This procedure links to the built-in portscan utility. This will query a virtual machine for running network services, and if accessible, return the output.

p_pwd
This procedure links to the built-in pwd utility. It will return a string with the current working directory.

Hacking to Learn

The default mission pack (destiny.mis) is a regular ascii text file. It has been left

uncompiled so you can feel free to examine the code, hack it apart, and get a good idea of how mission packs can be put together. The different missions in this mission pack try to use different aspects of the API (sometimes in different ways) specifically so one may be able to learn some of the various techniques available. You should definitely take a little bit of time to read through this code and experiment. Happy hacking!