

```
#!/usr/bin/env python
# nmapqt - a PyQt graphical front-end for nmap (http://www.insecure.org/nmap/) based on nmapfe
# By Fredrik Formwall <fredrikformwall@gmail.com>
#
# Thanks to Matthias Votisky <matthias.votisky@gmail.com> for bug fix

import posix, sys

try:
    from qt import *
except:
    print 'ERROR: PyQt (http://www.riverbankcomputing.co.uk/pyqt/) could not be found - exiting'
    sys.exit(1)

SCAN_TYPE = 0
RPC_SCAN = 1
OS_DETECTION = 3
OS_SCAN_LIMIT = 4
VERSION_PROBE = 5
SCANNED_PORTS = 6
DNS_RESOLUTION = 14
VERBOSITY = 15
PACKET_TRACE = 16
DONT_PING = 20
ICMP_ECHO = 21
ICMP_TIMESTAMP = 22
ICMP_NETMASK = 23
TCP_ACK_PING = 24
TCP_SYN_PING = 25
UDP_PING = 26
TIMING_POLICY = 30
IPV4_TTL = 31
MIN_PARALLEL = 32
MAX_PARALLEL = 33
INITIAL_RTT = 34
MIN_RTT = 35
MAX_RTT = 36
HOST_TIMEOUT = 37
SCAN_DELAY = 38
INPUT_FILE = 40
OUTPUT_FILE = 41
APPEND_OUTPUT = 42
SOURCE_DEVICE = 50
SOURCE_PORT = 51
SOURCE_IP = 52
SOURCE_DECOY = 53
FRAGMENTATION = 54
IPV6 = 55
ORDERED_PORTS = 56

isRoot = posix.getuid() == 0
flags = { INPUT_FILE : '' }

class ScanTab(QWidget):
    def __init__(self, parent):
        QWidget.__init__(self, parent)
        self.layout = QGridLayout(self, 2, 2, 5, 5)

        scanTypeGroupBox = QGroupBox('Scan Type', self)
        self.layout.addWidget(scanTypeGroupBox, 0, 0)
        scanTypeGroupBox.setColumnLayout(2, Qt.Vertical)
        self.scanTypeCombo = QComboBox(scanTypeGroupBox)
        QToolTip.add(self.scanTypeCombo, """<qt><b>TCP Connect Scan:</b> The most basic form of TCP scanning using
the connect() system call.<br><b>Ping Sweep:</b> Checking if a host is up.<br><b>List Scan:</b> Generates and
prints a list of IPs/names without scanning them.<br><b>FTP bounce attack:</b> Using an "ftp" proxy.<br><b>SYN
Stealth Scan:</b> Avoid opening a full TCP connection to avoid being logged.<br><b>ACK Stealth Scan:</b> Sends an
ACK packet to each specified ports and marks a port as filtered if nothing comes back.<br><b>FIN | ACK Stealth
Scan:</b> Sends a tcp packet with the FIN and ACK flags set, and get the answer from the remote server. If the
remote port is closed, then the remote host will send a packet containing the RST flag set. Because this check does
not perform the three-way handshake, it may not be logged by older scan detectors. Note that Windows boxes answer
that every port is closed!<br><b>FIN Stealth Scan:</b> Use a bare FIN packet as a probe.<br><b>NULL Stealth
Scan:</b> Turns off all flags in the probe packet.<br><b>Xmas Tree Stealth Scan:</b> Turns on the FIN, URG and PUSH
flags in the probe packet.<br><b>TCP Window Scan:</b> Very similar to the ACK scan, except that it can sometimes
detect open ports as well as filtered/unfiltered due to an anomaly in the TCP window size reporting by some
operating systems.<br><b>UDP Port Scan:</b> Determine which UDP ports are open by sending 0 byte UDP packets to
each port and checking if an ICMP port unreachable message is received.<br><b>IP Protocol Scan:</b> Determine which
IP protocols are supported by sending raw IP packets without any further protocol header to each specified protocol
"""
```

and assumes a protocol is open if no ICMP protocol unreachable message is received.
Idle Scan: An advanced scan method which allows a scan to be performed without sending any packet to the target from your real IP address. """)

```

self.scanTypeCombo.insertItem('TCP Connect Scan (-sT)')
self.scanTypeCombo.insertItem('Ping Sweep (-sP)')
self.scanTypeCombo.insertItem('List Scan (-sL)')
self.scanTypeCombo.insertItem('FTP Bounce Attack (-b)')
if isRoot:
    self.scanTypeCombo.insertItem('SYN Stealth Scan (-sS)')
    self.scanTypeCombo.insertItem('ACK Stealth Scan (-sA)')
    self.scanTypeCombo.insertItem('FIN | ACK Stealth Scan (-sM)')
    self.scanTypeCombo.insertItem('FIN Stealth Scan (-sF)')
    self.scanTypeCombo.insertItem('NULL Stealth Scan (-sN)')
    self.scanTypeCombo.insertItem('XMas Tree Stealth Scan (-sX)')
    self.scanTypeCombo.insertItem('TCP Window Scan (-sW)')
    self.scanTypeCombo.insertItem('UDP Port Scan (-sU)')
    self.scanTypeCombo.insertItem('IP Protocol Scan (-sO)')
    self.scanTypeCombo.insertItem('Idle Scan (-sI)')
    self.scanTypeCombo.setCurrentItem(4)
relayHostRow = QHBoxLayout(scanTypeGroupBox)
relayHostRow.setSpacing(5)
self.relayHostLabel = QLabel('Relay Host:', relayHostRow)
self.relayHostLabel.setDisabled(True)
self.relayHostEdit = QLineEdit(relayHostRow)
self.relayHostEdit.setDisabled(True)

self.scannedPortsGroupBox = QGroupBox('Scanned Ports', self)
self.layout.addWidget(self.scannedPortsGroupBox, 0, 1)
self.scannedPortsGroupBox.setColumnLayout(2, Qt.Vertical)
self.scannedPortsCombo = QComboBox(self.scannedPortsGroupBox)
QToolTip.add(self.scannedPortsCombo, """<qt><b>Default:</b> Scan all ports between 1 and 1024 as well as
any ports listed in the services file which comes with nmap.<br><b>All:</b> Scan all 65535 ports on a
host.<br><b>Fast:</b> Specifies that you only wish to scan for ports listed in the services file which comes with
nmap (or the protocols file for -sO). This is obviously much faster than scanning all 65535 ports on a
host.<br><b>Range Given Below:</b> This option specifies what ports you want to specify. For example "23" will only
try port 23 of the target host(s). "20-30,139,60000-" scans ports between 20 and 30, port 139, and all ports
greater than 60000. For IP protocol scanning (-sO), this specifies the protocol number you wish to scan for
(0-255). When scanning both TCP and UDP ports, you can specify a particular protocol by preceding the port numbers
by "T:" or "U:". The qualifier lasts until you specify another qualifier. For example, the argument
"U:53,111,137,T:21-25,80,139,8080" would scan UDP ports 53,111,and 137, as well as the listed TCP ports. Note that
to scan both UDP & TCP, you have to specify -sU and at least one TCP scan type (such as -sS, -sF, or -sT). If no
protocol qualifier is given, the port numbers are added to all protocol lists.</qt>""")
self.scannedPortsCombo.insertItem('Default')
self.scannedPortsCombo.insertItem('All (-p-)')
self.scannedPortsCombo.insertItem('Most Important [fast] (-F)')
self.scannedPortsCombo.insertItem('Range Given Below (-p)')
rangeRow = QHBoxLayout(self.scannedPortsGroupBox)
rangeRow.setSpacing(5)
self.rangeLabel = QLabel('Range:', rangeRow)
self.rangeLabel.setDisabled(True)
self.rangeEdit = QLineEdit(rangeRow)
self.rangeEdit.setDisabled(True)

self.scanExtensionGroupBox = QGroupBox('Scan Extensions', self)
self.layout.addMultiCellWidget(self.scanExtensionGroupBox, 1, 1, 0, 1)
self.scanExtensionGroupBox.setColumnLayout(4, Qt.Horizontal)
self.rpcScanCheck = QCheckBox('RPC Scan (-sR)', self.scanExtensionGroupBox)
QToolTip.add(self.rpcScanCheck, """<qt>This method works in combination with the various port scan methods
of Nmap. It takes all the TCP/UDP ports found open and then floods them with SunRPC program NULL commands in an
attempt to determine whether they are RPC ports, and if so, what program and version number they serve up. Thus you
can effectively obtain the same info as "rpcinfo -p" even if the target's portmapper is behind a firewall (or
protected by TCP wrappers). Decoys do not currently work with RPC scan, at some point I may add decoy support for
UDP RPC scans.""")
self.osDetectionCheck = QCheckBox('OS Detection (-O)', self.scanExtensionGroupBox)
QToolTip.add(self.osDetectionCheck, """<qt>This option activates remote host identification via TCP/IP
fingerprinting. In other words, it uses a bunch of techniques to detect subtleties in the underlying operating
system network stack of the computers you are scanning. It uses this information to create a "fingerprint" which it
compares with its database of known OS fingerprints (the nmap-os-fingerprints file) to decide what type of system
you are scanning. The -O option also enables several other tests such as "Uptime" measure and TCP Sequence
Predictability Classification ((a measure that describes approximately how hard it is to establish a forged TCP
connection against the remote host). When verbose mode (-v) is on with -O, IPID Sequence Generation is also
reported. Most machines are in the "incremental" class, which means that they increment the "ID" field in the IP
header for each packet they send. This makes them vulnerable to several advanced information gathering and spoofing
attacks.""")
self.osDetectionCheck.setDisabled(not isRoot)
self.osDetectionCheck.setChecked(isRoot)
self.osScanLimitCheck = QCheckBox('OS Scan Limit (--osscan_limit)', self.scanExtensionGroupBox)

```

QToolTip.add(self.osScanLimitCheck, ""<qt>OS detection is far more effective if at least one open and one closed TCP port are found. Set this option and Nmap will not even try OS detection against hosts that do not meet this criteria. This can save substantial time, particularly on -P0 scans against many hosts. It only matters when OS detection is requested (-O or -A options).""")

```
self.versionProbeCheck = QCheckBox('Version Probe (-sV)', self.scanExtensionGroupBox)
```

QToolTip.add(self.versionProbeCheck, ""<qt>After TCP and/or UDP ports are discovered using one of the other scan methods, version detection communicates with those ports to try and determine more about what is actually running. A file called nmap-service-probes is used to determine the best probes for detecting various services and the match strings to expect. Nmap tries to determine the service protocol (e.g. ftp, ssh, telnet, http), the application name (e.g. ISC Bind, Apache httpd, Solaris telnetd), the version number, and sometimes miscellaneous details like whether an X server is open to connections or the SSH protocol version). If Nmap was compiled with OpenSSL support, it will connect to SSL servers to deduce the service listening behind the encryption. When RPC services are discovered, the Nmap RPC grinder is used to determine the RPC program and version numbers. Note that the Nmap -A option also enables this feature. For a much more detailed description of Nmap service detection, read our paper at <http://www.insecure.org/nmap/versionscan.html>. There is a related --version_trace option which causes Nmap to print out extensive debugging info about what version scanning is doing (this is a subset of what you would get with --packet_trace).""")

```
QObject.connect(self.rpcScanCheck, SIGNAL('clicked()'), self.update)
```

```
QObject.connect(self.osDetectionCheck, SIGNAL('clicked()'), self.update)
```

```
QObject.connect(self.osScanLimitCheck, SIGNAL('clicked()'), self.update)
```

```
QObject.connect(self.versionProbeCheck, SIGNAL('clicked()'), self.update)
```

```
QObject.connect(self.scannedPortsCombo, SIGNAL('activated(int)'), self.update)
```

```
QObject.connect(self.rangeEdit, SIGNAL('textChanged(const QString&)'), self.update)
```

```
QObject.connect(self.relayHostEdit, SIGNAL('textChanged(const QString&)'), self.update)
```

```
QObject.connect(self.scanTypeCombo, SIGNAL('activated(int)'), self.update)
```

```
def update(self):
```

```
    item = self.scanTypeCombo.currentItem()
```

```
    disableAll = item == 1 or item == 2
```

```
    protocolScan = item == 12
```

```
    self.scannedPortsGroupBox.setTitle(['Scanned Ports', 'Scanned Protocols'][protocolScan * 1])
```

```
    self.scannedPortsGroupBox.setDisabled(disableAll)
```

```
    self.scanExtensionGroupBox.setDisabled(disableAll)
```

```
    self.osScanLimitCheck.setDisabled(not self.osDetectionCheck.isChecked())
```

```
    flags[SCAN_TYPE] = ['-sT', '-sP', '-sL', '-b ' + str(self.relayHostEdit.text()),
                       '-sS', '-sA', '-sM', '-sF', '-sN', '-sX', '-sW', '-sU', '-sO',
                       '-sI ' + str(self.relayHostEdit.text())][item]
```

```
    flags[RPC_SCAN] = ['', '-sR'][(not protocolScan) * (not disableAll) * self.rpcScanCheck.isChecked() * 1]
```

```
    flags[OS_DETECTION] = ['', '-O'][(not disableAll) * self.osDetectionCheck.isChecked()]
```

```
    flags[OS_SCAN_LIMIT] = ['', '--osscan_limit'][self.osScanLimitCheck.isChecked() * 1]
```

```
    flags[VERSION_PROBE] = ['', '-sV'][(not protocolScan) * (not disableAll) *
```

```
self.versionProbeCheck.isChecked() * 1]
```

```
    flags[SCANNED_PORTS] = ['', '-p-', '-F', '-p ' + str(self.rangeEdit.text())][(not disableAll) *
```

```
self.scannedPortsCombo.currentItem()]
```

```
    self.rangeLabel.setDisabled(self.scannedPortsCombo.currentItem() != 3)
```

```
    self.rangeEdit.setDisabled(self.scannedPortsCombo.currentItem() != 3)
```

```
    self.fragmentationCheck.setDisabled(not 4 <= item <= 10)
```

```
    flags[FRAGMENTATION] = ['', '-f'][(self.fragmentationCheck.isEnabled() * self.fragmentationCheck.isChecked())
```

```
* 1]
```

```
    self.relayHostLabel.setDisabled(False)
```

```
    self.relayHostEdit.setDisabled(False)
```

```
    if self.scanTypeCombo.currentItem() == 3:
```

```
        self.relayHostLabel.setText('Bounce Host:')
```

```
    elif self.scanTypeCombo.currentItem() == 13:
```

```
        self.relayHostLabel.setText('Zombie Host:')
```

```
    else:
```

```
        self.relayHostLabel.setDisabled(True)
```

```
        self.relayHostEdit.setDisabled(True)
```

```
    widget.updateCommand()
```

```
class DiscoverTab(QWidget):
```

```
    def __init__(self, parent):
```

```
        QWidget.__init__(self, parent)
```

```
        self.layout = QGridLayout(self, 2, 1, 5, 5)
```

```
        self.dontPingCheck = QCheckBox('Don\'t Ping (-P0)', self)
```

QToolTip.add(self.dontPingCheck, ""<qt>Do not try and ping hosts at all before scanning them. This allows the scanning of networks that don't allow ICMP echo requests (or responses) through their firewall. microsoft.com is an example of such a network, and thus you should always use -P0 or -PT80 when portscanning microsoft.com.""")

```
        self.layout.addWidget(self.dontPingCheck, 0, 0)
```

```
        self.pingTypesGroupBox = QGroupBox('Ping Types', self)
```

```

self.layout.addWidget(self.pingTypesGroupBox, 1, 0)
self.pingTypesGroupBox.setColumnLayout(1, Qt.Vertical)
grid = QWidget(self.pingTypesGroupBox)
grid.layout = QGridLayout(grid, 3, 4, 5, 5)

self.icmpEchoCheck = QCheckBox('ICMP Echo (-PE)', grid)
QToolTip.add(self.icmpEchoCheck, """<qt>This option uses a true ping (ICMP echo request) packet. It finds
hosts that are up and also looks for subnet-directed broadcast addresses on your network. These are IP addresses
which are externally reachable and translate to a broadcast of incoming IP packets to a subnet of computers. These
should be eliminated if found as they allow for numerous denial of service attacks (Smurf is the most common).""")
grid.layout.addWidget(self.icmpEchoCheck, 0, 0)
self.ackPingCheck = QCheckBox('TCP ACK Ping (-PT)', grid)
self.ackPingCheck.setChecked(True)
QToolTip.add(self.ackPingCheck, """<qt>Use TCP "ping" to determine what hosts are up. Instead of sending
ICMP echo request packets and waiting for a response, we spew out TCP ACK packets throughout the target network (or
to a single machine) and then wait for responses to trickle back. Hosts that are up should respond with a RST. This
option preserves the efficiency of only scanning hosts that are up while still allowing you to scan networks/hosts
that block ping packets. For non root users, we use connect(). To set the destination ports of the probe packets
use -PT<port1>,<port2>[...]. The default port is 80, since this port is often not filtered out. Note that this
option now accepts multiple, comma-separated port numbers.""")
grid.layout.addWidget(self.ackPingCheck, 0, 1)
self.ackPingPortsLabel = QLabel('Port(s):', grid)
grid.layout.addWidget(self.ackPingPortsLabel, 0, 2)
self.ackPingEdit = QLineEdit(grid)
grid.layout.addWidget(self.ackPingEdit, 0, 3)

self.icmpTimeStampCheck = QCheckBox('ICMP Timestamp (-PP)', grid)
QToolTip.add(self.icmpTimeStampCheck, """<qt>Uses an ICMP timestamp request (code 13) packet to find
listening hosts.""")
grid.layout.addWidget(self.icmpTimeStampCheck, 1, 0)
self.synPingCheck = QCheckBox('TCP SYN Ping (-PS)', grid)
QToolTip.add(self.synPingCheck, """<qt>This option uses SYN (connection request) packets instead of ACK
packets for root users. Hosts that are up should respond with a RST (or, rarely, a SYN|ACK). You can set the
destination ports in the same manner as -PT above.""")
grid.layout.addWidget(self.synPingCheck, 1, 1)
self.synPingPortsLabel = QLabel('Port(s):', grid)
grid.layout.addWidget(self.synPingPortsLabel, 1, 2)
self.synPingEdit = QLineEdit(grid)
grid.layout.addWidget(self.synPingEdit, 1, 3)

self.icmpNetmaskCheck = QCheckBox('ICMP Netmask (-PM)', grid)
QToolTip.add(self.icmpNetmaskCheck, """<qt>Same as -PE and -PP except uses a netmask request (ICMP code
17).""")
grid.layout.addWidget(self.icmpNetmaskCheck, 2, 0)
self.udpPingCheck = QCheckBox('UDP Ping (-PU)', grid)
QToolTip.add(self.udpPingCheck, """<qt>This option sends UDP probes to the specified hosts, expecting an
ICMP port unreachable packet (or possibly a UDP response if the port is open) if the host is up. Since many UDP
services won't reply to an empty packet, your best bet might be to send this to expected-closed ports rather than
open ones.""")
grid.layout.addWidget(self.udpPingCheck, 2, 1)
self.udpPingPortsLabel = QLabel('Port(s):', grid)
grid.layout.addWidget(self.udpPingPortsLabel, 2, 2)
self.udpPingEdit = QLineEdit(grid)
grid.layout.addWidget(self.udpPingEdit, 2, 3)

if not isRoot:
    self.icmpEchoCheck.setDisabled(True)
    self.icmpTimeStampCheck.setDisabled(True)
    self.icmpNetmaskCheck.setDisabled(True)
    self.synPingCheck.setDisabled(True)
    self.udpPingCheck.setDisabled(True)
else:
    self.icmpEchoCheck.setChecked(True)

QObject.connect(self.dontPingCheck, SIGNAL('clicked()'), self.update)
QObject.connect(self.icmpEchoCheck, SIGNAL('clicked()'), self.update)
QObject.connect(self.ackPingCheck, SIGNAL('clicked()'), self.update)
QObject.connect(self.ackPingEdit, SIGNAL('textChanged(const QString&)'), self.update)
QObject.connect(self.icmpTimeStampCheck, SIGNAL('clicked()'), self.update)
QObject.connect(self.synPingCheck, SIGNAL('clicked()'), self.update)
QObject.connect(self.synPingEdit, SIGNAL('textChanged(const QString&)'), self.update)
QObject.connect(self.icmpNetmaskCheck, SIGNAL('clicked()'), self.update)
QObject.connect(self.udpPingCheck, SIGNAL('clicked()'), self.update)
QObject.connect(self.udpPingEdit, SIGNAL('textChanged(const QString&)'), self.update)

def update(self):
    ping = not self.dontPingCheck.isChecked()

```

```

self.pingTypesGroupBox.setEnabled(not ping)

flags[DONT_PING] = ['-P0', ''][ping * 1]

ackPing = self.ackPingCheck.isChecked()
flags[TCP_ACK_PING] = ['', '-PT' + str(self.ackPingEdit.text())][ping * ackPing * 1]
self.ackPingEdit.setEnabled(not ackPing)

synPing = self.synPingCheck.isChecked()
flags[TCP_SYN_PING] = ['', '-PS' + str(self.synPingEdit.text())][ping * synPing * 1]
self.synPingEdit.setEnabled(not synPing)

udpPing = self.udpPingCheck.isChecked()
flags[UDP_PING] = ['', '-PU' + str(self.udpPingEdit.text())][ping * udpPing * 1]
self.udpPingEdit.setEnabled(not udpPing)

flags[ICMP_ECHO] = ['', '-PE'][ping * self.icmpEchoCheck.isChecked() * 1]
flags[ICMP_TIMESTAMP] = ['', '-PP'][ping * self.icmpTimeStampCheck.isChecked() * 1]
flags[ICMP_NETMASK] = ['', '-PM'][ping * self.icmpNetmaskCheck.isChecked() * 1]

widget.updateCommand()

class TimingTab(QWidget):
    def __init__(self, parent):
        QWidget.__init__(self, parent)
        self.layout = QGridLayout(self, 5, 4, 5, 5)
        self.layout.setColStretch(0, 1)
        self.layout.setColStretch(2, 1)

        self.throttlingCombo = QComboBox(self)
        QToolTip.add(self.throttlingCombo, """<qt>These are canned timing policies for conveniently expressing your
priorities to Nmap. <b>Paranoid</b> mode scans very slowly in the hopes of avoiding detection by IDS systems. It
serializes all scans (no parallel scanning) and generally waits at least 5 minutes between sending packets.
<b>Sneaky</b> is similar, except it only waits 15 seconds between sending packets. <b>Polite</b> is meant to ease
load on the network and reduce the chances of crashing machines. It serializes the probes and waits at least 0.4
seconds between them. Note that this is generally at least an order of magnitude slower than default scans, so only
use it when you need to. <b>Normal</b> is the default Nmap behaviour, which tries to run as quickly as possible
without overloading the network or missing hosts/ports. <b>Aggressive</b> can make certain scans (especially SYN
scans against heavily filtered hosts) much faster. It is recommended for impatient folks with a fast net
connection. <b>Insane</b> is only suitable for very fast networks or where you don't mind losing some information.
It times out hosts in 15 minutes and won't wait more than 0.3 seconds for individual probes. It does allow for very
quick network sweeps though :). You can also reference these by number (0-5). For example, "-T0" gives you Paranoid
mode and "-T5" is Insane mode.""")
        self.throttlingCombo.insertItem('Paranoid (-T0)')
        self.throttlingCombo.insertItem('Sneaky (-T1)')
        self.throttlingCombo.insertItem('Polite (-T2)')
        self.throttlingCombo.insertItem('Normal')
        self.throttlingCombo.insertItem('Aggressive (-T4)')
        self.throttlingCombo.insertItem('Insane (-T5)')
        self.throttlingCombo.setCurrentItem(3)
        self.layout.addWidget(QLabel('Timing Policy:', self), 0, 0)
        self.layout.addWidget(self.throttlingCombo, 0, 1)

        self.ipv4TtlCheck = QCheckBox('IPv4 TTL (-ttl)', self)
        QToolTip.add(self.ipv4TtlCheck, """<qt>Sets the IPv4 time to live field in sent packets to the given
value.""")
        self.layout.addWidget(self.ipv4TtlCheck, 2, 0)
        self.ipv4TtlSpin = QSpinBox(1, 255, 1, self)
        self.ipv4TtlSpin.setValue(127)
        self.layout.addWidget(self.ipv4TtlSpin, 2, 1)

        self.minParallelCheck = QCheckBox('Min. Parallel (--min_parallelism)', self)
        QToolTip.add(self.minParallelCheck, """<qt>Tells Nmap to scan at least the given number of ports in
parallel. This can speed up scans against certain firewalled hosts by an order of magnitude. But be careful --
results will become unreliable if you push it too far.""")
        self.layout.addWidget(self.minParallelCheck, 3, 0)
        self.minParallelSpin = QSpinBox(1, 150, 1, self)
        self.layout.addWidget(self.minParallelSpin, 3, 1)

        self.maxParallelCheck = QCheckBox('Max. Parallel (-M)', self)
        QToolTip.add(self.maxParallelCheck, """<qt>Specifies the maximum number of scans Nmap is allowed to perform
in parallel. Setting this to one means Nmap will never try to scan more than 1 port at a time. It also effects
other parallel scans such as ping sweep, RPC scan, etc.""")
        self.layout.addWidget(self.maxParallelCheck, 4, 0)
        self.maxParallelSpin = QSpinBox(1, 1500, 1, self)
        self.layout.addWidget(self.maxParallelSpin, 4, 1)

```

```

    self.initialRttCheck = QCheckBox('Initial RTT (--initial_rtt_timeout)', self)
    QToolTip.add(self.initialRttCheck, """<qt>Specifies the initial probe timeout. This is generally only
useful when scanning firewalled hosts with -P0. Normally Nmap can obtain good RTT estimates from the ping and the
first few probes. The default mode uses 6000.""")
    self.layout.addWidget(self.initialRttCheck, 0, 2)
    self.initialRttSpin = QSpinBox(0, 9999999, 10, self)
    self.initialRttSpin.setValue(6000)
    self.layout.addWidget(self.initialRttSpin, 0, 3)

    self.minRttCheck = QCheckBox('Min. RTT (--min_rtt_timeout)', self)
    QToolTip.add(self.minRttCheck, """<qt>When the target hosts start to establish a pattern of responding very
quickly, Nmap will shrink the amount of time given per probe. This speeds up the scan, but can lead to missed
packets when a response takes longer than usual. With this parameter you can guarantee that Nmap will wait at least
the given amount of time before giving up on a probe.""")
    self.layout.addWidget(self.minRttCheck, 1, 2)
    self.minRttSpin = QSpinBox(1, 9999999, 10, self)
    self.minRttSpin.setValue(6000)
    self.layout.addWidget(self.minRttSpin, 1, 3)

    self.maxRttCheck = QCheckBox('Max. RTT (--max_rtt_timeout)', self)
    QToolTip.add(self.maxRttCheck, """<qt>Specifies the maximum amount of time Nmap is allowed to wait for a
probe response before retransmitting or timing out that particular probe. The default mode sets this to about
9000.""")
    self.layout.addWidget(self.maxRttCheck, 2, 2)
    self.maxRttSpin = QSpinBox(6, 9999999, 10, self)
    self.maxRttSpin.setValue(6000)
    self.layout.addWidget(self.maxRttSpin, 2, 3)

    self.hostTimeoutCheck = QCheckBox('Host Timeout (--host_timeout)', self)
    QToolTip.add(self.hostTimeoutCheck, """<qt>Specifies the amount of time Nmap is allowed to spend scanning a
single host before giving up on that IP. The default timing mode has no host timeout.""")
    self.layout.addWidget(self.hostTimeoutCheck, 3, 2)
    self.hostTimeoutSpin = QSpinBox(201, 9999999, 10, self)
    self.hostTimeoutSpin.setValue(6000)
    self.layout.addWidget(self.hostTimeoutSpin, 3, 3)

    self.scanDelayCheck = QCheckBox('Scan Delay (--scan_delay)', self)
    QToolTip.add(self.scanDelayCheck, """<qt>Specifies the minimum amount of time Nmap must wait between
probes. This is mostly useful to reduce network load or to slow the scan way down to sneak under IDS thresholds.""")
    self.layout.addWidget(self.scanDelayCheck, 4, 2)
    self.scanDelaySpin = QSpinBox(1, 9999999, 10, self)
    self.scanDelaySpin.setValue(6000)
    self.layout.addWidget(self.scanDelaySpin, 4, 3)

    if not isRoot:
        self.ipv4TtlCheck.setDisabled(True)
        self.minParallelCheck.setDisabled(True)
        self.maxParallelCheck.setDisabled(True)
        self.initialRttCheck.setDisabled(True)
        self.minRttCheck.setDisabled(True)
        self.maxRttCheck.setDisabled(True)
        self.hostTimeoutCheck.setDisabled(True)
        self.scanDelayCheck.setDisabled(True)

    QObject.connect(self.throttlingCombo, SIGNAL('activated(int)'), self.update)
    QObject.connect(self.ipv4TtlCheck, SIGNAL('clicked()'), self.update)
    QObject.connect(self.minParallelCheck, SIGNAL('clicked()'), self.update)
    QObject.connect(self.maxParallelCheck, SIGNAL('clicked()'), self.update)
    QObject.connect(self.initialRttCheck, SIGNAL('clicked()'), self.update)
    QObject.connect(self.minRttCheck, SIGNAL('clicked()'), self.update)
    QObject.connect(self.maxRttCheck, SIGNAL('clicked()'), self.update)
    QObject.connect(self.hostTimeoutCheck, SIGNAL('clicked()'), self.update)
    QObject.connect(self.scanDelayCheck, SIGNAL('clicked()'), self.update)
    QObject.connect(self.ipv4TtlSpin, SIGNAL('valueChanged(int)'), self.update)
    QObject.connect(self.minParallelSpin, SIGNAL('valueChanged(int)'), self.update)
    QObject.connect(self.maxParallelSpin, SIGNAL('valueChanged(int)'), self.update)
    QObject.connect(self.initialRttSpin, SIGNAL('valueChanged(int)'), self.update)
    QObject.connect(self.minRttSpin, SIGNAL('valueChanged(int)'), self.update)
    QObject.connect(self.maxRttSpin, SIGNAL('valueChanged(int)'), self.update)
    QObject.connect(self.hostTimeoutSpin, SIGNAL('valueChanged(int)'), self.update)
    QObject.connect(self.scanDelaySpin, SIGNAL('valueChanged(int)'), self.update)

    def update(self):
        self.ipv4TtlSpin.setDisabled(not self.ipv4TtlCheck.isChecked())
        self.minParallelSpin.setDisabled(not self.minParallelCheck.isChecked())
        self.maxParallelSpin.setDisabled(not self.maxParallelCheck.isChecked())
        self.initialRttSpin.setDisabled(not self.initialRttCheck.isChecked())

```

```

self.minRttSpin.setDisabled(not self.minRttCheck.isChecked())
self.maxRttSpin.setDisabled(not self.maxRttCheck.isChecked())
self.hostTimeoutSpin.setDisabled(not self.hostTimeoutCheck.isChecked())
self.scanDelaySpin.setDisabled(not self.scanDelayCheck.isChecked())

flags[TIMING_POLICY] = ['-T0', '-T1', '-T2', '', '-T4', '-T5'][self.throttlingCombo.currentItem()]
flags[IPV4_TTL] = ['', '--ttl %s' % self.ipv4TtlSpin.value()][self.ipv4TtlCheck.isChecked() * 1]
flags[MIN_PARALLEL] = ['', '--min_parallelism %s' %
self.minParallelSpin.value()][self.minParallelCheck.isChecked() * 1]
flags[MAX_PARALLEL] = ['', '--max_parallelism %s' %
self.maxParallelSpin.value()][self.maxParallelCheck.isChecked() * 1]
flags[INITIAL_RTT] = ['', '--initial_rtt_timeout %s' %
self.initialRttSpin.value()][self.initialRttCheck.isChecked() * 1]
flags[MIN_RTT] = ['', '--min_rtt_timeout %s' % self.minRttSpin.value()][self.minRttCheck.isChecked() * 1]
flags[MAX_RTT] = ['', '--max_rtt_timeout %s' % self.maxRttSpin.value()][self.maxRttCheck.isChecked() * 1]
flags[HOST_TIMEOUT] = ['', '--host_timeout %s' %
self.hostTimeoutSpin.value()][self.hostTimeoutCheck.isChecked() * 1]
flags[SCAN_DELAY] = ['', '--scan_delay %s' % self.scanDelaySpin.value()][self.scanDelayCheck.isChecked() *
1]

```

```
widget.updateCommand()
```

```
class FilesTab(QHBox):
```

```

def __init__(self, parent):
    QHBox.__init__(self, parent)
    self.setSpacing(4)
    self.setMargin(4)

    inputFileGroupBox = QGroupBox('Input File', self)
    inputFileGroupBox.setColumnLayout(2, Qt.Vertical)
    self.inputFileCheck = QCheckBox('Input File (-iL)', inputFileGroupBox)
    QToolTip.add(self.inputFileCheck, """<qt>Reads target specifications from the file specified RATHER than
from the command line. The file should contain a list of host or network expressions separated by spaces, tabs, or
newlines. Use a hyphen (-) as input filename if you want nmap to read host expressions from stdin (like at the
end of a pipe). See the section target specification for more information on the expressions you fill the file
with.""")
    inputRow = QHBox(inputFileGroupBox)
    inputRow.setSpacing(5)
    self.inputFileEdit = QLineEdit(inputRow)
    self.inputBrowseButton = QPushButton('Browse', inputRow)

    outputFileGroupBox = QGroupBox('Output File', self)
    outputFileGroupBox.setColumnLayout(4, Qt.Vertical)
    topRow = QHBox(outputFileGroupBox)
    self.outputFileCheck = QCheckBox('Output File', topRow)
    QToolTip.add(self.outputFileCheck, """Tell Nmap to write a log to a file as well as stdout. You may control
the format of the log below""")
    middleRow = QHBox(outputFileGroupBox)
    middleRow.setSpacing(5)
    self.outputFileEdit = QLineEdit(middleRow)
    self.outputBrowseButton = QPushButton('Browse', middleRow)
    bottomRow = QHBox(outputFileGroupBox)
    self.outputFormatLabel = QLabel('Output Format', bottomRow)
    self.outputFormatCombo = QComboBox(bottomRow)
    QToolTip.add(self.outputFormatCombo, """<qt>Specify the format which you want Nmap to use when writing its
log to the specified file.<br><b>Normal:</b> Normal human readable form.<br><b>Grepable:</b> This logs the results
of your scans in a grepable form into the file you specify as an argument. This simple format provides all the
information on one line (so you can easily grep for port or OS information and see all the IPs. This used to be the
preferred mechanism for programs to interact with Nmap, but now we recommend XML output (-oX instead). This simple
format may not contain as much information as the other formats. You can give the argument "-" (without quotes) to
shoot output into stdout (for shell pipelines, etc). In this case normal output will be suppressed. Watch out for
error messages if you use this (they will still go to stderr). Also note that "-v" will cause some extra
information to be printed.<br><b>XML:</b> This logs the results of your scans in XML form into the file you specify
as an argument. This allows programs to easily capture and interpret Nmap results. You can give the argument "-"
(without quotes) to shoot output into stdout (for shell pipelines, etc). In this case normal output will be
suppressed. Watch out for error messages if you use this (they will still go to stderr). Also note that "-v" may
cause some extra information to be printed. The Document Type Definition (DTD) defining the XML output structure is
available at http://www.insecure.org/nmap/nmap.dtd.<br><b>All:</b> This tells Nmap to log in ALL the major formats
(normal, grepable, and XML). You give a base for the filename, and the output files will be base.nmap, base.gnmap,
and base.xml.<br><b>Script Kiddie:</b> thIs 10gz th3 r3suLts of YouR ScanZ iN a s|&lt;ipT kiDd|3 f0rM iNtO The fiL3
U sPecfy 4s an arGuMEnt! U kAn gIv3 the 4rgument "-" (wItHOUT qUoteZ) to sh00t output iNtO stDout!@!""")
    self.outputFormatCombo.insertItem('Normal (-oN)')
    self.outputFormatCombo.insertItem('Grepable (-oG)')
    self.outputFormatCombo.insertItem('XML (-oX)')
    self.outputFormatCombo.insertItem('All (-oA)')
    self.outputFormatCombo.insertItem('Script Kiddie (-oS)')
    self.appendOutputCheck = QCheckBox('Append to File (--append_output)', outputFileGroupBox)

```

```

QToolTip.add(self.appendOutputCheck, """<qt>Tells Nmap to append scan results to any output files you have
specified rather than overwriting those files""")

QObject.connect(self.inputFileCheck, SIGNAL('clicked()'), self.update)
QObject.connect(self.inputFileEdit, SIGNAL('textChanged(const QString&)'), self.update)
QObject.connect(self.outputFileCheck, SIGNAL('clicked()'), self.update)
QObject.connect(self.outputFileEdit, SIGNAL('textChanged(const QString&)'), self.update)
QObject.connect(self.appendOutputCheck, SIGNAL('clicked()'), self.update)
QObject.connect(self.inputBrowseButton, SIGNAL('clicked()'), self.getInputFile)
QObject.connect(self.outputBrowseButton, SIGNAL('clicked()'), self.getOutputFile)
QObject.connect(self.outputFormatCombo, SIGNAL('activated(int)'), self.update)

def getOutputFile(self):
    self.outputFileEdit.setText(QFileDialog.getSaveFileName())
    self.update()

def getInputFile(self):
    self.inputFileEdit.setText(QFileDialog.getOpenFileName())
    self.update()

def update(self):
    inputFile = self.inputFileCheck.isChecked()
    widget.targetEdit.setDisabled(inputFile)
    self.inputFileEdit.setDisabled(not inputFile)
    self.inputBrowseButton.setDisabled(not inputFile)
    flags[INPUT_FILE] = [' ', '-iL ' + str(self.inputFileEdit.text())][inputFile * 1]

    outputFile = self.outputFileCheck.isChecked()
    self.outputFileEdit.setDisabled(not outputFile)
    self.outputBrowseButton.setDisabled(not outputFile)
    self.outputFormatLabel.setDisabled(not outputFile)
    self.outputFormatCombo.setDisabled(not outputFile)
    self.appendOutputCheck.setDisabled(not outputFile)
    if outputFile:
        flags[OUTPUT_FILE] = ['-oN ', '-oG ', '-oX ', '-oA ', '-oS '][self.outputFormatCombo.currentItem()] +
str(self.outputFileEdit.text())
        flags[APPEND_OUTPUT] = [' ', '--append_output'][self.appendOutputCheck.isChecked() * 1]
    else:
        flags[OUTPUT_FILE] = flags[APPEND_OUTPUT] = ''

    widget.updateCommand()

class OptionsTab(QWidget):
    def __init__(self, parent):
        QWidget.__init__(self, parent)
        self.layout = QGridLayout(self, 2, 3, 5, 5)

        dnsResolutionGroupBox = QGroupBox('Reverse DNS Resolution', self)
        QToolTip.add(dnsResolutionGroupBox, """<qt>Normally, Nmap do reverse DNS resolution on a target IP address
when a machine is found to be alive. Since DNS is often slow, telling Nmap to never do this may speed things up.
You can also specify that reverse DNS resolution on the target IP address always should be done, regardless of any
machine found active or not.""")
        dnsResolutionGroupBox.setColumnLayout(1, Qt.Vertical)
        self.layout.addWidget(dnsResolutionGroupBox, 0, 0)
        self.dnsResolutionCombo = QComboBox(dnsResolutionGroupBox)
        self.dnsResolutionCombo.insertItem('Always (-R)')
        self.dnsResolutionCombo.insertItem('When Required')
        self.dnsResolutionCombo.insertItem('Never (-n)')
        self.dnsResolutionCombo.setCurrentItem(1)

        verbosityGroupBox = QGroupBox('Verbosity', self)
        QToolTip.add(verbosityGroupBox, """<qt>Controls how much output Nmap should generate. The verbose mode (-v)
is highly recommended as it gives out more information about what is going on.""")
        verbosityGroupBox.setColumnLayout(2, Qt.Vertical)
        self.layout.addWidget(verbosityGroupBox, 1, 0)
        self.verbosityCombo = QComboBox(verbosityGroupBox)
        self.verbosityCombo.insertItem('Quiet')
        self.verbosityCombo.insertItem('Verbose (-v)')
        self.verbosityCombo.insertItem('Very Verbose (-vv)')
        self.verbosityCombo.insertItem('Debug (-d)')
        self.verbosityCombo.insertItem('Verbose Debug (-d2)')
        self.packetTraceCheck = QCheckBox('Packet trace (--packet_trace)', verbosityGroupBox)
        QToolTip.add(self.packetTraceCheck, """<qt>Tells Nmap to show all the packets it sends and receives
in a tcpdump-like format. This can be tremendously useful for debugging, and is also a good learning tool.""")

        sourceGroupBox = QGroupBox('Source', self)
        if not isRoot:

```

```

        sourceGroupBox.setDisabled(True)
        sourceGroupBox.setColumnLayout(1, Qt.Vertical)
        self.layout.addMultiCellWidget(sourceGroupBox, 0, 1, 1, 1)
        sourceGrid = QWidget(sourceGroupBox)
        sourceGrid.layout = QGridLayout(sourceGrid, 4, 2, 5, 5)
        self.sourceDeviceCheck = QCheckBox('Device (-e)', sourceGrid)
        QToolTip.add(self.sourceDeviceCheck, ""<qt>Tells nmap what interface to send and receive packets on. Nmap
should be able to detect this but it will tell you if it cannot.""")
        sourceGrid.layout.addWidget(self.sourceDeviceCheck, 0, 0)
        self.sourceDeviceEdit = QLineEdit(sourceGrid)
        sourceGrid.layout.addWidget(self.sourceDeviceEdit, 0, 1)
        self.sourcePortCheck = QCheckBox('Port (-g)', sourceGrid)
        QToolTip.add(self.sourcePortCheck, ""<qt>Sets the source port number used in scans. Many naive firewall
and packet filter installations make an exception in their ruleset to allow DNS (53) or FTP-DATA (20) packets to
come through and establish a connection. Obviously this completely subverts the security advantages of the firewall
since intruders can just masquerade as FTP or DNS by modifying their source port. Obviously for a UDP scan you
should try 53 first and TCP scans should try 20 before 53. Note that this is only a request -- nmap will honor it
only if and when it is able to. For example, you can't do TCP ISN sampling all from one host:port to one host:port,
so nmap changes the source port even if you used -g. Be aware that there is a small performance penalty on some
scans for using this option, because I sometimes store useful information in the source port number.""")
        sourceGrid.layout.addWidget(self.sourcePortCheck, 1, 0)
        self.sourcePortEdit = QLineEdit(sourceGrid)
        sourceGrid.layout.addWidget(self.sourcePortEdit, 1, 1)
        self.sourceIpCheck = QCheckBox('IP (-S)', sourceGrid)
        QToolTip.add(self.sourceIpCheck, ""<qt>In some circumstances, nmap may not be able to determine your
source address (nmap will tell you if this is the case). In this situation, use -S with your IP address (of the
interface you wish to send packets through). Another possible use of this flag is to spoof the scan to make the
targets think that someone else is scanning them. Imagine a company being repeatedly port scanned by a competitor!
This is not a supported usage (or the main purpose) of this flag. I just think it raises an interesting
possibility that people should be aware of before they go accusing others of port scanning them. -e would generally
be required for this sort of usage.""")
        sourceGrid.layout.addWidget(self.sourceIpCheck, 2, 0)
        self.sourceIpEdit = QLineEdit(sourceGrid)
        sourceGrid.layout.addWidget(self.sourceIpEdit, 2, 1)
        self.sourceDecoyCheck = QCheckBox('Decoy (-D)', sourceGrid)
        QToolTip.add(self.sourceDecoyCheck, ""<qt>Causes a decoy scan to be performed which makes it appear to the
remote host that the host(s) you specify as decoys are scanning the target network too. It is generally an
extremely effective technique for hiding your IP address. Separate each decoy host with commas, and you can
optionally use "ME" as one of the decoys to represent the position you want your IP address to be used (random by
default). If your put "ME" in the 6th position or later, some common port scan detectors (such as Solar Designer's
excellent scanlogd) are unlikely to show your IP address at all. The decoys should be up or you might accidently
SYN flood your targets. You might want to use IP addresses instead of names (so the decoy networks don't see you in
their nameserver logs). Also note that some (stupid) "port scan detectors" will firewall/deny routing to hosts that
attempt port scans. Thus you might inadvertently cause the machine you scan to lose connectivity with the decoy
machines you are using. This could cause the target machines major problems if the decoy is, say, its internet
gateway or even "localhost". Thus you might want to be careful of this option. Decoys are used both in the initial
ping scan (using ICMP, SYN, ACK, or whatever) and during the actual port scanning phase. Decoys are also used
during remote OS detection( -O ). It is worth noting that using too many decoys may slow your scan and potentially
even make it less accurate. Also, some ISPs will filter out your spoofed packets, although many (current most) do
not restrict spoofed IP packets at all.""")
        sourceGrid.layout.addWidget(self.sourceDecoyCheck, 3, 0)
        self.sourceDecoyEdit = QLineEdit(sourceGrid)
        sourceGrid.layout.addWidget(self.sourceDecoyEdit, 3, 1)

        miscGroupBox = QGroupBox('Misc. Options', self)
        miscGroupBox.setColumnLayout(3, Qt.Vertical)
        self.fragmentationCheck = QCheckBox('Fragmentation (-f)', miscGroupBox)
        QToolTip.add(self.fragmentationCheck, ""<qt>This option causes the requested SYN, FIN, XMAS, or NULL scan
to use tiny fragmented IP packets. The idea is to split up the TCP header over several packets to make it harder
for packet filters, intrusion detection systems, and other annoyances to detect what you are doing. Be careful with
this! Some programs have trouble handling these tiny packets. My favorite sniffer segmentation faulted immediately
upon receiving the first 36-byte fragment. After that comes a 24 byte one! While this method won't get by packet
filters and firewalls that queue all IP fragments (like the CONFIG_IP_ALWAYS_DEFRAG option in the Linux kernel),
some networks can't afford the performance hit this causes and thus leave it disabled. Note that I do not yet have
this option working on all systems. It works fine for my Linux, FreeBSD, and OpenBSD boxes and some people have
reported success with other *NIX variants.""")
        self.ipv6Check = QCheckBox('IPv6 (-6)', miscGroupBox)
        QToolTip.add(self.ipv6Check, ""<qt>This options enables IPv6 support. All targets must be IPv6 if this
option is used, and they can be specified via normal DNS name (AAAA record) or as a literal IP address such as
3ffe:501:4819:2000:210:f3ff:fe03:4d0. Currently, connect() TCP scan and TCP connect() Ping scan are supported. If
you need UDP or other scan types, have a look at http://nmap6.sourceforge.net/.""")
        self.orderedPortsCheck = QCheckBox('Ordered Ports (-r)', miscGroupBox)
        QToolTip.add(self.orderedPortsCheck, ""<qt>Tells Nmap <b>NOT</b> to randomize the order in which ports are
scanned.""")
        self.layout.addMultiCellWidget(miscGroupBox, 0, 1, 2, 2)

    if not isRoot:

```

```

        miscGroupBox.setEnabled(True)

QObject.connect(self.verbosityCombo, SIGNAL('activated(int)'), self.update)
QObject.connect(self.packetTraceCheck, SIGNAL('clicked()'), self.update)
QObject.connect(self.dnsResolutionCombo, SIGNAL('activated(int)'), self.update)
QObject.connect(self.sourceDeviceCheck, SIGNAL('clicked()'), self.update)
QObject.connect(self.sourcePortCheck, SIGNAL('clicked()'), self.update)
QObject.connect(self.sourceIpCheck, SIGNAL('clicked()'), self.update)
QObject.connect(self.sourceDecoyCheck, SIGNAL('clicked()'), self.update)
QObject.connect(self.fragmentationCheck, SIGNAL('clicked()'), self.update)
QObject.connect(self.ipv6Check, SIGNAL('clicked()'), self.update)
QObject.connect(self.orderedPortsCheck, SIGNAL('clicked()'), self.update)
QObject.connect(self.sourceDeviceEdit, SIGNAL('textChanged(const QString&)'), self.update)
QObject.connect(self.sourcePortEdit, SIGNAL('textChanged(const QString&)'), self.update)
QObject.connect(self.sourceIpEdit, SIGNAL('textChanged(const QString&)'), self.update)
QObject.connect(self.sourceDecoyEdit, SIGNAL('textChanged(const QString&)'), self.update)

def update(self):
    self.sourceDeviceEdit.setEnabled(not self.sourceDeviceCheck.isChecked())
    self.sourcePortEdit.setEnabled(not self.sourcePortCheck.isChecked())
    self.sourceIpEdit.setEnabled(not self.sourceIpCheck.isChecked())
    self.sourceDecoyEdit.setEnabled(not self.sourceDecoyCheck.isChecked())

    flags[SOURCE_DEVICE] = [' ', '-e ' + str(self.sourceDeviceEdit.text())][self.sourceDeviceCheck.isChecked() *
1]
    flags[SOURCE_PORT] = [' ', '-g ' + str(self.sourcePortEdit.text())][self.sourcePortCheck.isChecked() * 1]
    flags[SOURCE_IP] = [' ', '-S ' + str(self.sourceIpEdit.text())][self.sourceIpCheck.isChecked() * 1]
    flags[SOURCE_DECOY] = [' ', '-D ' + str(self.sourceDecoyEdit.text())][self.sourceDecoyCheck.isChecked() * 1]
    flags[DNS_RESOLUTION] = ['-R', '', '-n'][self.dnsResolutionCombo.currentItem()]
    flags[VERBOSITY] = [' ', '-v', '-vv', '-d', '-d2'][self.verbosityCombo.currentItem()]
    flags[PACKET_TRACE] = [' ', '--packet_trace'][self.packetTraceCheck.isChecked() * 1]
    flags[FRAGMENTATION] = [' ', '-f'][self.fragmentationCheck.isEnabled() * self.fragmentationCheck.isChecked()
* 1]

    flags[IPV6] = [' ', '-6'][self.ipv6Check.isChecked() * 1]
    flags[ORDERED_PORTS] = [' ', '-r'][self.orderedPortsCheck.isChecked() * 1]

    widget.updateCommand()

class MainWindow(QWidget):
    def __init__(self, *args):
        QWidget.__init__(self, *args)
        self.proc = QProcess()
        self.layout = QGridLayout(self, 5, 1, 5, 10)

        topRow = QHBoxLayout(self)
        self.layout.addWidget(topRow, 0, 0)
        topRow.setSpacing(5)
        QLabel('Target(s):', topRow)
        self.targetEdit = QLineEdit(topRow)
        self.targetEdit.setText('localhost')
        self.scanButton = QPushButton('E&ecute', topRow)
        #self.scanButton.setFixedSize(self.scanButton.size()) # So it does not resize when changing text to 'Stop'
        QToolTip.add(self.scanButton, "Start scanning the specified target(s) by executing the command displayed at
the bottom")

        #exitButton = QPushButton('E&xit', topRow)
        #QToolTip.add(exitButton, 'Exit the application')

        tabWidget = QTabWidget(self)
        self.layout.addWidget(tabWidget, 1, 0)
        self.scanTab = ScanTab(tabWidget)
        self.discoverTab = DiscoverTab(tabWidget)
        self.timingTab = TimingTab(tabWidget)
        self.filesTab = FilesTab(tabWidget)
        self.optionsTab = OptionsTab(tabWidget)
        self.scanTab.fragmentationCheck = self.optionsTab.fragmentationCheck
        tabWidget.addTab(self.scanTab, 'Scan')
        tabWidget.addTab(self.discoverTab, 'Discover')
        tabWidget.addTab(self.timingTab, 'Timing')
        tabWidget.addTab(self.filesTab, 'Files')
        tabWidget.addTab(self.optionsTab, 'Options')

        self.log = QTextEdit(self)
        QToolTip.add(self.log, ""<qt>Here you can see the messages from nmapqt (<b><ex>blue</ex></b>) as well as
output from nmap to stdout (<b>black</b>) and stderr (<b><er>red</er></b>)"")
        self.log.setMaxLogLines(5000)
        #self.log.setWordWrap(QTextEdit.WidgetWidth)

```

```

self.layout.addWidget(self.log, 2, 0)
self.layout.setRowStretch(2, 1)
#self.log.setTextFormat(QTextEdit.LogText)
self.log.setReadOnly(True)
styleSheetItem = QStyleSheetItem(self.log.styleSheet(), 'ex')
styleSheetItem.setColor(QColor(0, 0, 255))
styleSheetItem = QStyleSheetItem(self.log.styleSheet(), 'er')
styleSheetItem.setColor(QColor(255, 0, 0))
if isRoot:
    self.log.append('<ex>You are root - all options granted.</ex>')
else:
    self.log.append('<ex>You are *NOT* root - some options aren\'t available.</ex>')
bottomRow = QHBoxLayout(self)
bottomRow.setSpacing(5)
QLabel('Command:', bottomRow)
self.commandDisplay = QLineEdit(bottomRow)
self.commandDisplay.setReadOnly(True)
self.layout.addWidget(bottomRow, 3, 0)
QToolTip.add(self.commandDisplay, "<qt>This is the command, created from the specified options above, which
will be executed when you click \"Scan\"")

QObject.connect(self.proc, SIGNAL('readyReadStdout()'), self.readStdout)
QObject.connect(self.proc, SIGNAL('readyReadStderr()'), self.readStderr)
QObject.connect(self.proc, SIGNAL('processExited()'), self.processExited)
QObject.connect(self.targetEdit, SIGNAL('textChanged(const QString&)'), self.updateCommand)
QObject.connect(self.targetEdit, SIGNAL('returnPressed()'), self.startProcess)
QObject.connect(self.scanButton, SIGNAL('clicked()'), self.startProcess)
#QObject.connect(exitButton, SIGNAL('clicked()'), QApplication.quit)

def clearLog(self):
    self.log.clear()

def initialUpdate(self):
    for widget in [self.scanTab, self.discoverTab, self.timingTab, self.filesTab, self.optionsTab]:
        widget.update()
    self.updateCommand()

def updateCommand(self):
    self.command = 'nmap'
    for value in flags.values():
        if not len(value) == 0: self.command += ' ' + value
    if len(flags[INPUT_FILE]) == 0:
        self.command += ' ' + str(self.targetEdit.text())
    self.commandDisplay.setText(self.command)

def readStdout(self):
    while (self.proc.canReadLineStdout()):
        self.log.append(escapeChars(self.proc.readLineStdout()))

def readStderr(self):
    while self.proc.canReadLineStderr():
        line = self.proc.readLineStderr()
        self.log.append(escapeChars(line).prepend('<er>').append('</er>'))

def startProcess(self):
    if self.proc.isRunning():
        self.proc.kill()
        self.log.append('<ex>Killing process</ex>')
        return
    text = self.commandDisplay.text()
    self.proc.setArguments(QStringList.split(' ', text))
    self.log.append(escapeChars(text).prepend('<ex>').append('</ex>'))
    self.scanButton.setText('S&top')
    if not self.proc.start():
        self.log.append('<ex>Could not execute command - check that nmap is installed!</ex>')

def processExited(self):
    self.log.append('\n\n\n')
    self.scanButton.setText('E&xecute')

def escapeChars(s):
    return s.replace('<', '&lt;').replace('>', '&gt;')

def showAboutData():
    QMessageBox.information(mainWindow, 'About', 'nmap - Network exploration tool and security scanner<br>Fyodor
&lt;fyodor@insecure.org&gt;<br><br>nmapqt - Graphical frontend for nmap<br>Fredrik Fornwall
&lt;fredrikfornwall@gmail.com&gt;')

```

```

def openLog():
    fileName = QFileDialog.getOpenFileName()
    if fileName.isEmpty(): return
    try:
        logFile = open(str(fileName), 'r')
    except:
        QMessageBox.warning(widget, 'Error', 'Could not open file for reading')
    return
    widget.log.setText(logFile.read())
    logFile.close()

def saveLog():
    fileName = QFileDialog.getSaveFileName()
    if fileName.isEmpty(): return
    try:
        logFile = open(str(fileName), 'w')
    except:
        QMessageBox.warning(widget, 'Error', 'Could not open file for writing')
    return
    logFile.write(str(widget.log.text()))
    logFile.close()

def nmapVersion():
    widget.proc.clearArguments()
    widget.proc.addArgument('nmap')
    widget.proc.addArgument('--version')
    widget.log.append('<ex>nmap --version</ex>')
    if not widget.proc.start():
        widget.log.append('<ex>Could not execute command - check that nmap is installed!</ex>\n')

def toggleDisplayTooltips():
    settingsMenu.setItemChecked(enableTooltipsMenuId, not settingsMenu.isChecked(enableTooltipsMenuId))
    QToolTip.setGloballyEnabled(settingsMenu.isChecked(enableTooltipsMenuId))

if __name__ == '__main__':
    if '-h' in sys.argv:
        print 'nmapqt 3.73 Usage: nmap [-h] [-d]\n  -h Shows this help\n  -d Initially disable tooltips\nIt also
supports the common Qt/X11 command line options'
        sys.exit(0)

    app = QApplication(sys.argv)

    mainWindow = QMainWindow();
    mainWindow.setCaption('nmapqt 3.73')

    widget = MainWidget(mainWindow)
    widget.initialUpdate()
    widget.setFocus()

    mainWindow.setCentralWidget(widget)
    mainWindow.setGeometry((app.desktop().width() - 800) / 2, (app.desktop().height() - 900) / 2, 800, 900)
    mainWindow.show()

    fileMenu = QPopupMenu(mainWindow.menuBar())
    fileMenu.insertItem('&Open Log', openLog)
    fileMenu.insertItem('&Save Log', saveLog)
    fileMenu.insertSeparator()
    fileMenu.insertItem('&Clear Output', widget.clearLog)
    fileMenu.insertSeparator()
    fileMenu.insertItem('E&xit', QApplication.quit)
    mainWindow.menuBar().insertItem('&File', fileMenu)

    settingsMenu = QPopupMenu(mainWindow.menuBar())
    settingsMenu.setCheckable(True)
    enableTooltipsMenuId = settingsMenu.insertItem('Enable &Tooltips', toggleDisplayTooltips)
    settingsMenu.setItemChecked(enableTooltipsMenuId, True)
    if '-d' in sys.argv: toggleDisplayTooltips()
    mainWindow.menuBar().insertItem('&Settings', settingsMenu)

    helpMenu = QPopupMenu(mainWindow.menuBar())
    helpMenu.insertItem('Nmap &version', nmapVersion)
    helpMenu.insertSeparator()
    helpMenu.insertItem('&About', showAboutData)
    mainWindow.menuBar().insertItem('&Help', helpMenu)

```

```
app.setMainWidget(mainWindow)  
app.exec_loop()
```