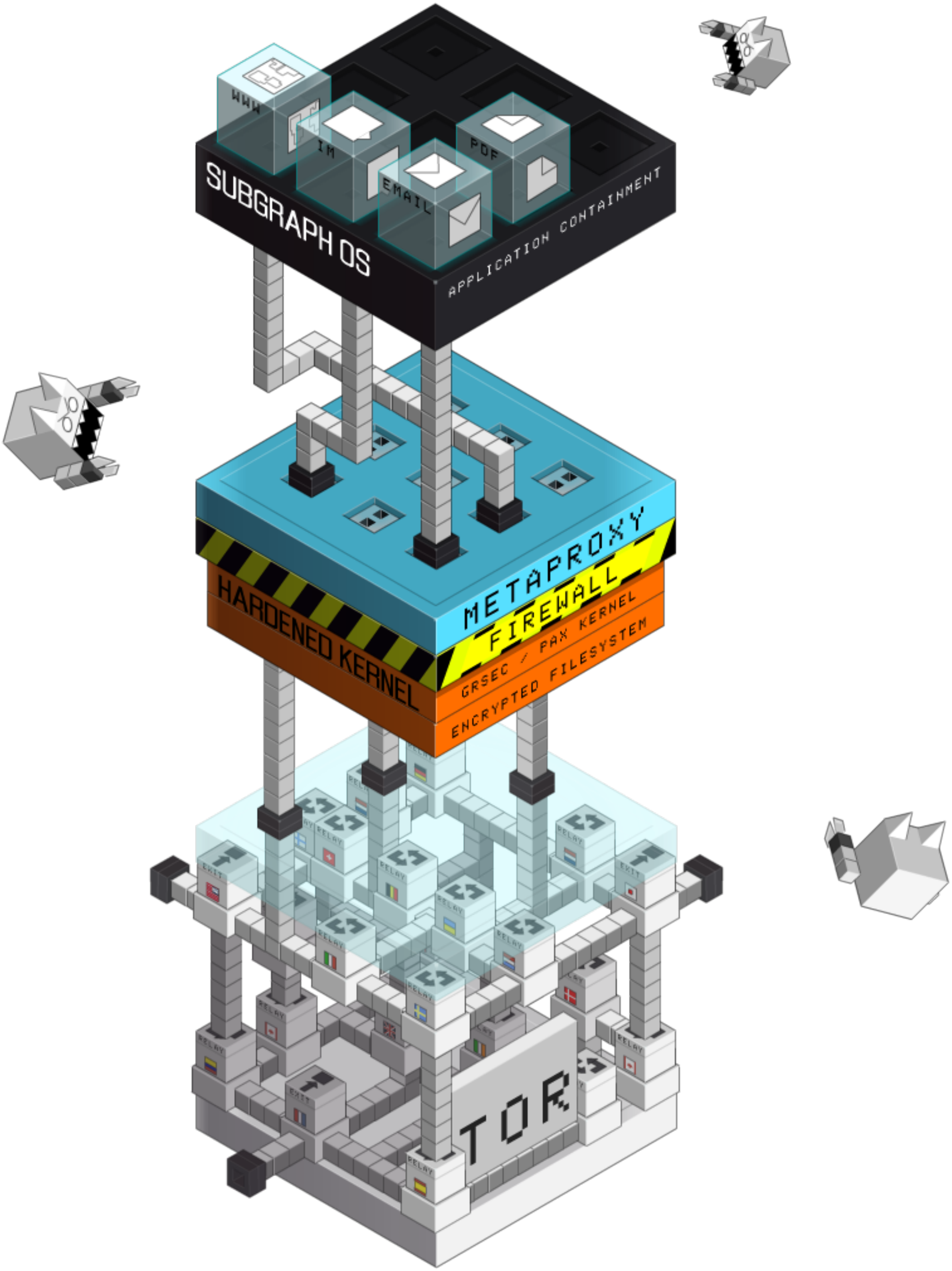
SUBGRAPH OS

HANDBOOK



Subgraph OS Handbook

Contents

Preface

We have created this handbook as an instructional manual on how to use the Subgraph operating system. This handbook also introduces various security and privacy enhancing technologies that we have developed at Subgraph.

We wrote this book for new users of Subgraph OS. Whether you are new to Linux or coming from another Linux-based operating system, we want to ease your transition to Subgraph OS.

In the first section, we describe how to perform common tasks such as installing Subgraph OS and using the various applications that are included. Start here to get up and running with Subgraph OS as quickly as possible.

The next section describes the various features of Subgraph OS that distinguish it from other operating systems. Users can refer to this section to learn the various security and privacy features. Advanced users will find this section useful for configuring operating system features and Subgraph applications.

Subgraph OS

What is Subgraph OS?

Subgraph OS is an *adversary resistant computing* platform.

Subgraph OS empowers people to communicate, share, and collaborate without fear of surveillance and interference. We designed it so that our users can safely perform their day-to-day tasks securely and privately.

In some ways, Subgraph OS is like other operating systems – it is derived from Debian GNU/Linux. It provides the familiar GNOME desktop environment as its graphical user interface. Subgraph OS includes applications found in other Linux distributions. These similarities make Subgraph OS easy to adopt, especially for users with prior Linux experience.

Subgraph OS also has key differences from conventional Linux operating systems. In particular:

1. Subgraph OS anonymizes Internet traffic by sending it through the Tor network
2. Security hardening makes Subgraph OS more resistant to security vulnerabilities
3. Subgraph runs many desktop applications in a security sandbox to limit their risk in case of compromise

What do we mean by security and privacy?

People attach different meanings to the words security and privacy. In computer security, a secure system is one that assures the confidentiality, integrity, and availability of information it stores, processes, or communicates.

> Confidentiality assures that information is not revealed to anybody who is not authorized
>
> Integrity assures that information cannot be modified or tampered with by anybody who is not authorized
>
> Availability assures that information can be reliably accessed by those who are authorized

Privacy is similar to confidentiality. Privacy also relies heavily on the integrity of communications. Our computers and other devices gather a great deal of information about our thoughts, our lives, and our social networks. They transmit this information over the Internet without our knowledge and consent. We have no way to trust the systems and networks that relay our communications over the Internet.

We designed Subgraph OS with these concerns in mind. We did this because we believe people should be able to communicate with each other privately. We also believe that people should not be required to reveal information about themselves or their social network without explicit consent.

What is adversary resistant computing?

We designed Subgraph from the ground up to defend against threats to security and privacy. We aim to provide our users with a computing platform that is *adversary resistant*.

We when use the term *adversary*, we are referring to an actual or hypothetical threat to the confidentiality, integrity, and availability of information.

Hackers who exploit software vulnerabilities are a type of adversary. This is an actual and often active threat to security and privacy.

Adversaries present passive or indirect threats as well. An adversary may be passively gathering network traffic to conduct surveillance on users.

Lastly, adversaries may present theoretical or impractical threats. For example, a cryptography algorithm may have a theoretical weakness. At the time the weakness is discovered, the threat may not *practical* in the real world. As technology and attack methods improve, the weakness ceases to be theoretical and real world attacks emerge.

We use the term *adversary* to cover all of the above possibilities.

Secure systems should be resistant to all of these types of threats.

While no computing platform can anticipate and defend against all possible threats by all possible adversaries, we aspire to make such attacks extremely difficult for adversaries. By making these attacks difficult, they also become more expensive for adversaries. Adversaries must bear the cost at scale if a large number of users deploy strong security and privacy defenses. Through Subgraph OS, we aim to make these defenses freely available and easy to deploy.

Some of our users have critical security and privacy needs. Subgraph OS grants them strong security and privacy to conduct their activities safely. Casual users also gain the same security and privacy benefits without having to sacrifice usability and maintainability.

This is *adversary resistant computing*.

Getting help with Subgraph OS

We hope to address most concerns with this handbook. If you have questions that are not addressed in this handbook, you can contact us through other means.

Contacting Subgraph

Email: Our email address is info <at> subgraph.com

IRC: You can join our IRC channel #subgraph on the OFTC network

Our IRC channel is also available through webchat at:

https://webchat.oftc.net/?channels=#subgraph

Twitter: Our Twitter is **@subgraph**, send us a message

We are also involved in running the **Secure Desktops** mailing list. This discussion group covers the topic of **Secure Desktop** operating systems such as Subgraph OS, Qubes OS, and Tails. Developers from these projects participate in the mailing list.

Further information about **Secure Desktops** can be found here:

https://secure-os.org/desktops/charter/

Reporting bugs

If you find a bug in Subgraph OS, you can report it to us on Github.

Our issue tracker for Subgraph OS is:

https://github.com/subgraph/subgraph-os-issues

You can also find our individual software repositories at:

https://github.com/subgraph

Getting the Subgraph OS Handbook

Up-to-date versions of this handbook can be found on the following page:

https://github.com/subgraph/sgos_handbook

The PDF can be downloaded here:

https://github.com/subgraph/sgos_handbook/raw/master/build/sgos_handbook.pdf

Subgraph OS will also include versions of this handbook in different formats.

14

Installing Subgraph OS

System requirements

Subgraph OS runs on Intel 64-bit computers. These are the system requirements:

- Intel 64-bit processor (Core2 Duo or later)
- 2GB of RAM (4GB recommended)
- At least 20GB of hard-disk space

Downloading and verifying the Subgraph OS ISO

Subgraph OS can be downloaded from our website:

https://subgraph.com/sgos/download/index.en.html

The Subgraph OS download page always has the most up-to-date download links and instructions. You can download the ISO directly from the website or over a Tor hidden service.

You should always verify that the ISO you downloaded is the official version. To verify the ISO, we have included a checksum that is cryptographically signed by our developers.

> *What is a checksum?*
>
> A checksum (or hash) is a string that uniquely identifies some piece of data as being different from another piece of data. It is computed using a special hash algorithm (SHA256 in our case). When data is passed to the hash algorithm, the algorithm will return a shortened string (the checksum) that uniquely identifies the data. Checksums are often used to ensure the integrity of a file. Integrity in this case means that the file has not been corrupted or tampered with during the download.
>
> *What is a cryptographic signature?*
>
> A cryptographic (or digital) signature is a method of authenticating a piece of data. Data is signed with the private signing key of a person who has created or is sending the data. The signature can then be verified by the recipient using the public key of the sender. If the verification is successful, this ensures that the data was created or sent by the person who signed it and not somebody else. This authenticates the identify of the creator or sender.
>
> *Why do we cryptographically sign the checksum?*
>
> The checksum is used to verify the integrity the ISO you have downloaded. However, how do you verify that the checksum on our website was provided by us? By cryptographically signing the checksum with our private key, you can verify the authenticity of the checksum.

Verifying the ISO on a Linux computer

To verify the ISO on a Linux computer, you will need to download the ISO, SHA256 checksum, and the signature for the checksum.

The first step is to download our public key, Our public key can be downloaded with the following command:

```
$ gpg ——recv—key  B55E70A95AC79474504C30D0DA11364B4760E444
```

The second step is to verify the authenticity of the signature for the checksum. Run the following command to verify the signature (note: replace the filenames with the names of the files you downloaded):

```
$ gpg ——verify subgraph—os—alpha_2016—06—16_2.iso.sha256.sig subgraph—os—alpha_2016—06—16_2.iso.sha256
```

After running this command, you should see a Good Signature message. If you have seen this message then you can proceed to the next step.

The third step is to verify the integrity of the ISO using the SHA256 checksum. Run the following command to verify the checksum (note: replace the filenames with the names of the files you downloaded):

```
$ sha256sum —c subgraph—os—alpha_2016—06—16_2.iso.sha256
```

After running the command, you should see:

```
subgraph—os—alpha_2016—06—16_2.iso: OK
```

Congratulations, you have now downloaded and verified the Subgraph OS ISO. You are now ready to try it out!

Installing from a USB drive on a Linux computer

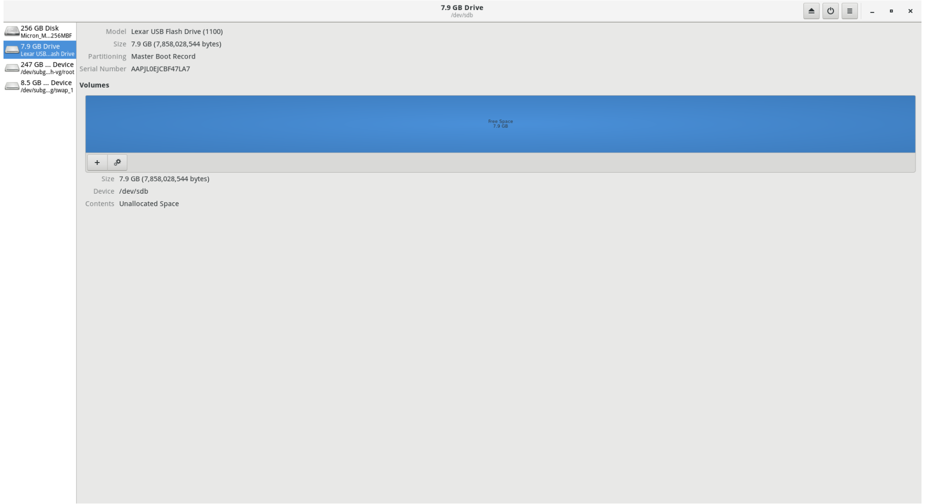
This section describes how to create a USB installer using Linux. One of these methods will work on any Linux computer (even on Subgraph OS). To create an installer you will need a USB drive with a capacity a 2GB or more.

Creating a USB installer using Gnome Disks

If you have a Linux computer that is running the Gnome Desktop, you can use the **Gnome Disks** application to create a USB installer.

The following steps show how to make a USB installer using **Gnome Disks**:

1. Insert a USB drive into your Linux computer
2. Open the **Gnome Disks** application
3. Select your USB drive

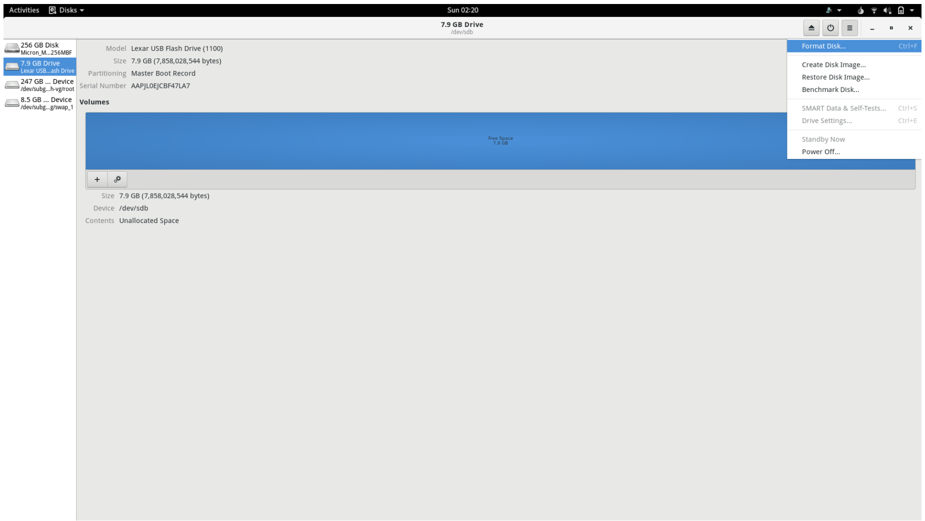


Model Lexar USB Flash Drive (1100)
Size 7.9 GB (7,858,028,544 bytes)
Partitioning Master Boot Record
Serial Number AAPJL0EJCBF47LA7

Size 7.9 GB (7,858,028,544 bytes)
Device /dev/sdb
Contents Unallocated Space

Figure 1: Gnome Disks - select USB drive

4. Select the **Format Disk** option in the top right corner of **Gnome Disks**

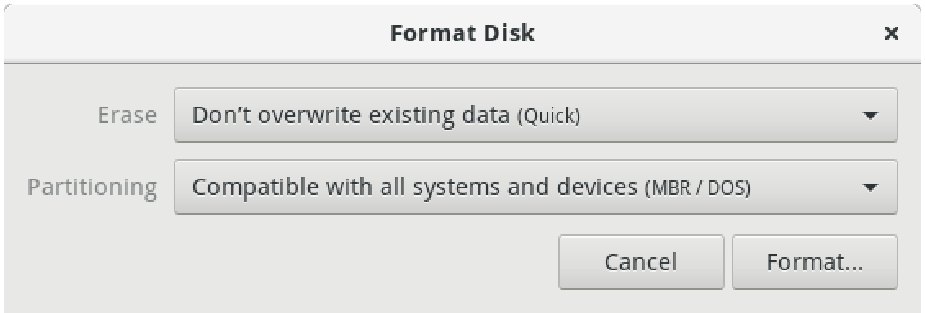


Model Lexar USB Flash Drive (1100)
Size 7.9 GB (7,858,028,544 bytes)
Partitioning Master Boot Record
Serial Number AAPJL0EJCBF47LA7

Size 7.9 GB (7,858,028,544 bytes)
Device /dev/sdb
Contents Unallocated Space

Figure 2: Gnome Disks - Format Disks... option

5. *Format* the USB drive



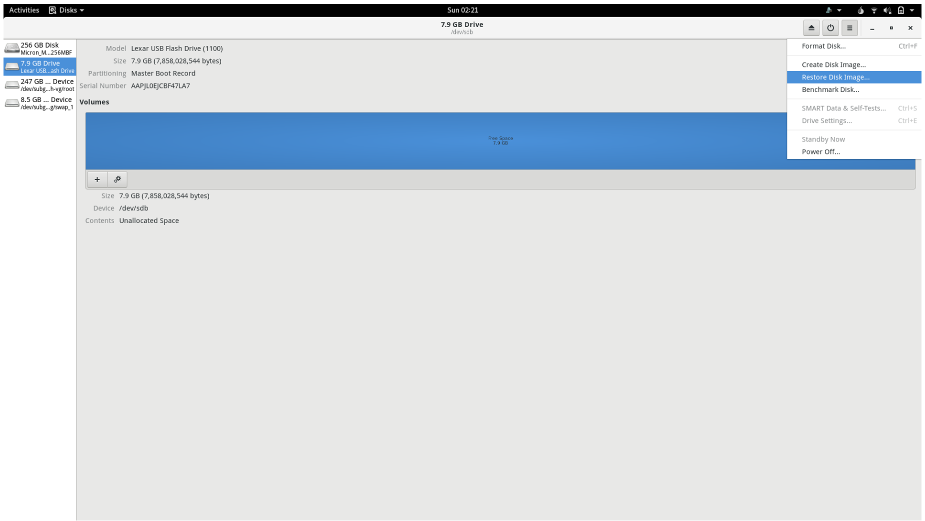
Format Disk

Erase: Don’t overwrite existing data (Quick)

Partitioning: Compatible with all systems and devices (MBR / DOS)

Figure 3: Gnome Disks - Format dialog

6. Select the **Restore Disk Image** option in the top right corner of **Gnome Disks**



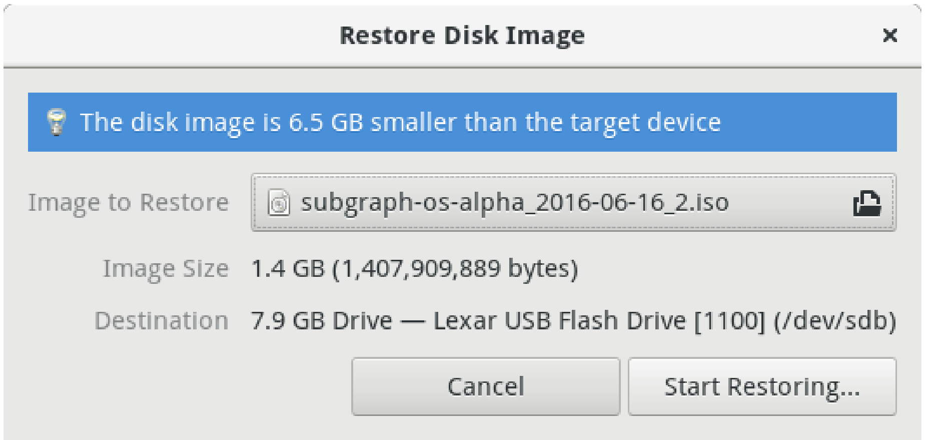
Model Lexar USB Flash Drive (1100)
Size 7.9 GB (7,858,028,544 bytes)
Partitioning Master Boot Record
Serial Number AAPJL0EJCBF47LA7

Size 7.9 GB (7,858,028,544 bytes)
Device /dev/sdb
Contents Unallocated Space

Figure 4: Gnome Disks - Restore Disk Image... option

7. Choose the ISO file you want to *restore* (copy) to the USB drive

8. *Restore* the ISO to the USB drive



Restore Disk Image

The disk image is 6.5 GB smaller than the target device

Image to Restore: subgraph-os-alpha_2016-06-16_2.iso

Image Size: 1.4 GB (1,407,909,889 bytes)

Destination: 7.9 GB Drive — Lexar USB Flash Drive [1100] (/dev/sdb)

Figure 5: Gnome Disks - Restore dialog

It should take a few minutes to copy the ISO to the USB drive.

Creating a USB installer using dd

If your Linux computer is not running Gnome Desktop or you want to create the installer from the command-line, you can use the **dd** utility.

The following steps show how to make a USB installer using **dd**:

1. Insert a USB drive into your computer
2. Open a terminal and run the following command to identify the name of the USB drive:

   ```
   $ lsblk
   ```

   NOTE: You should see a name such as **/dev/sdx** for your drive, for example: **/dev/sdb**. It is important to use only the name without the partition number. If you see something like **/dev/sdb1**, you can omit the **1** at the end. The **dd** command uses the name without the partition number.

3. In the same terminal, run the following command:

   ```
   $ dd bs=4M if=subgraph—os—alpha_2016—06—16_2.iso of=/dev/sdx status=progress && sync
   ```

   NOTE: Replace the path of the ISO with the path of the ISO you have downloaded and verified. Replace **/dev/sdx** with the name of your USB drive, for example: **/dev/sdb**.

Copying the ISO to the USB drive should take a few minutes.

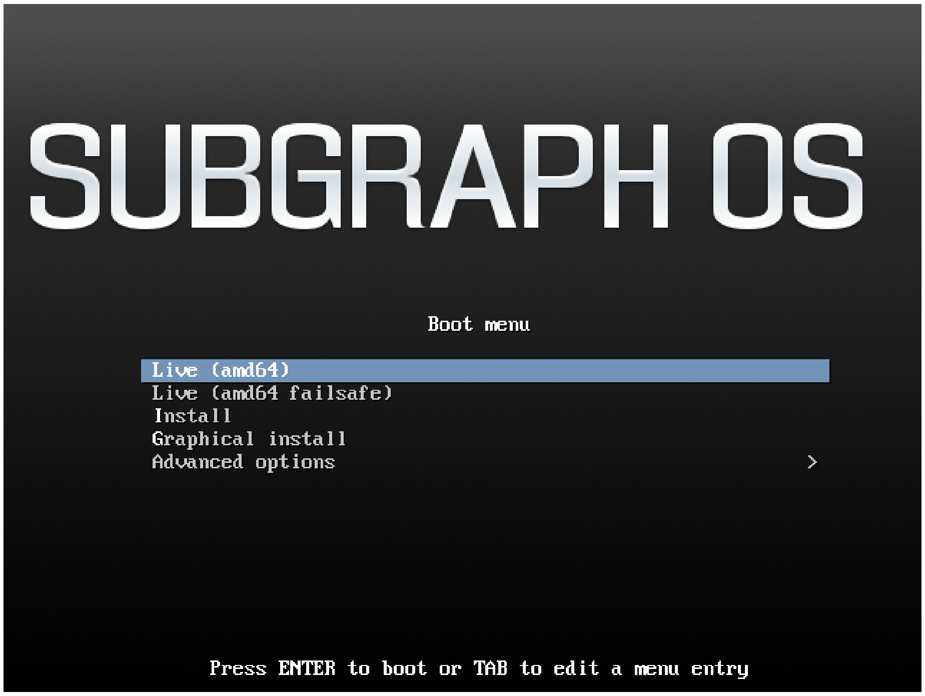
Booting from a USB drive (Live mode)

Subgraph OS also features a ‘live’ mode. Subgraph OS live mode runs in memory, directly from the USB drive. While running in live mode, nothing will be saved to your hard-drive. When the live session ends, any data created during your session will disappear, leaving no traces behind on the hard-disk.

People normally run in live mode for the following reasons:

1. They want to demo Subgraph OS
2. They want to test Subgraph OS with their particular hardware
3. They want to perform certain tasks with extra security and privacy but do not want a permanent installation of Subgraph OS

When the Subgraph OS ISO starts, you will be presented with different options. To start the live mode, select Live (amd64).



Press ENTER to boot or TAB to edit a menu entry

Figure 6: Subgraph OS boot screen

Please note that the user password on the live image is: *live*.

Everyday usage

Subgraph OS comes with a number of applications that may already be familiar. We have also added newer alternatives that may be less familiar. This chapter shows you how to use these applications to perform everyday tasks.

Subgraph OS is also unique because the applications we have included are run inside of a security sandbox. We call this sandbox Oz. Oz helps protect the operating system and your personal files in case an application is compromised by a security vulnerability.

Each application described in this chapter runs inside an Oz sandbox. This means that they can only access the files and directories that they need to. Each of the applications is isolated from each other. They are also isolated from the system itself. Because the applications are isolated, they cannot access common directories such as Pictures or Downloads in the usual way. This chapter shows you how to manage your files in Oz, with some examples for each application.

Browsing the Web with Tor Browser

Tor Browser is the default web browser of Subgraph OS. It has a number of security and privacy advantages over other browsers.

The security and privacy features include:

- Anti-fingerprinting countermeasures to prevent websites from identifying individual users by their browser fingerprint
- A security slider that lets users disable browser features that may pose security and privacy risks

The Tor Browser runs inside a security sandbox, managed by Subgraph Oz. Web browsers represent some of the most complex software available. With complexity comes increased risk to security and privacy. This is what we call the attack surface of an application. Tor Browser is no different than other browsers in that it has a lot of attack surface. A successful compromise of Tor Browser could let an attacker gain access to things such as SSH keys, GPG encryption keys, personal files, email, etc. Our security sandbox technology helps to mitigate these risks.

Configuring the Tor Browser security slider

The Tor Browser includes a security slider that lets users choose the security and privacy features they want to enable. If they enable all of the security and privacy settings, some websites may be slower or may not work as expected. However, the security slider lets them instantly lower the settings if they need a particular website to work better.

We recommend setting the security slider to Medium-High or High. For websites you trust, you can lower the settings to make the website perform better.

We advise against lowering the security slider for any websites that are not accessed over HTTPS. HTTPS helps to make sure that the traffic between the Tor Browser and the website has not been tampered with. This is what we refer to as the ‘integrity’ security property. If you cannot verify the integrity of the traffic originating from a website by using HTTPS, it may be dangerous to visit the website using lowered security and privacy settings.

Downloading and saving files in the Tor Browser

The Tor Browser runs inside of Oz, our application sandbox. When files are downloaded by a sandboxed application such as the Tor Browser, they are saved within the sandbox. When you close the Tor Browser, Oz will cleanup the sandbox, causing files saved in the sandbox to be destroyed.

To allow the Tor Browser to download that can persist after the application is closed, Oz makes a special exception. This special exception is a shared directory where files can be saved and retrieved later, without being destroyed when Tor Browser is closed. Shared directory, in this case, means a directory that is shared inside and outside of the Oz sandbox. Oz sets up the the following shared directory for saving downloaded files:

```
~/Downloads/TorBrowser
```

The shared directory name may be localized depending on the language settings on your computer. In the case of French, the shared directory would be:

```
~/Téléchargements/TorBrowser
```

Files downloaded to the shared directory will persist after closing the Tor Browser.

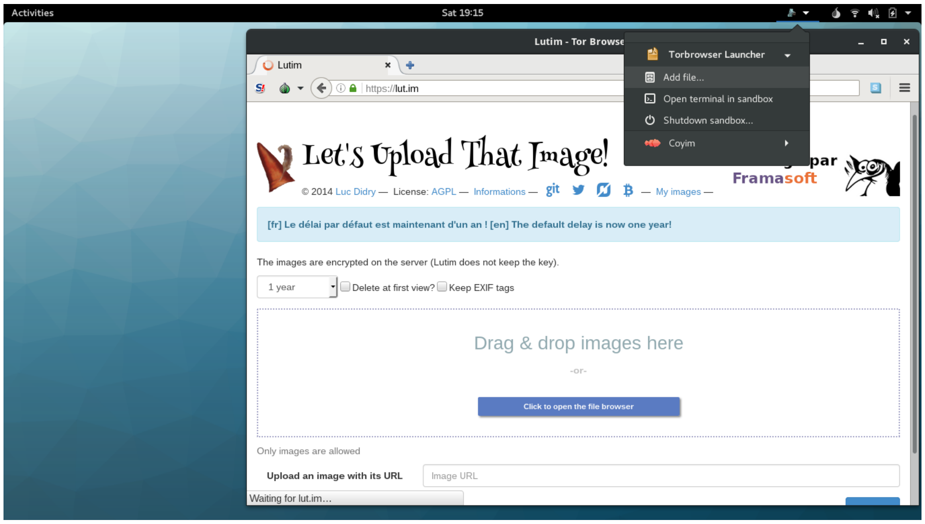
Uploading files in the Tor Browser

When the Tor Browser starts, the Oz sandbox limits its access to files and directories on the computer. For example, a photo from the Pictures directory will not be visible in the sandbox by default. If you want to upload a photo from this directory, you must use the Oz menu to add it to the Tor Browser sandbox. The Oz menu is denoted by the little zebra icon at the top-right corner of the screen.

The following actions may be performed using the Oz menu:

- Add files to sandbox
- Open terminal in sandbox
- Shutdown sandbox

Click on the little zebra and then click Add file ... .



[fr] Le délai par défaut est maintenant d'un an ! [en] The default delay is now one year!

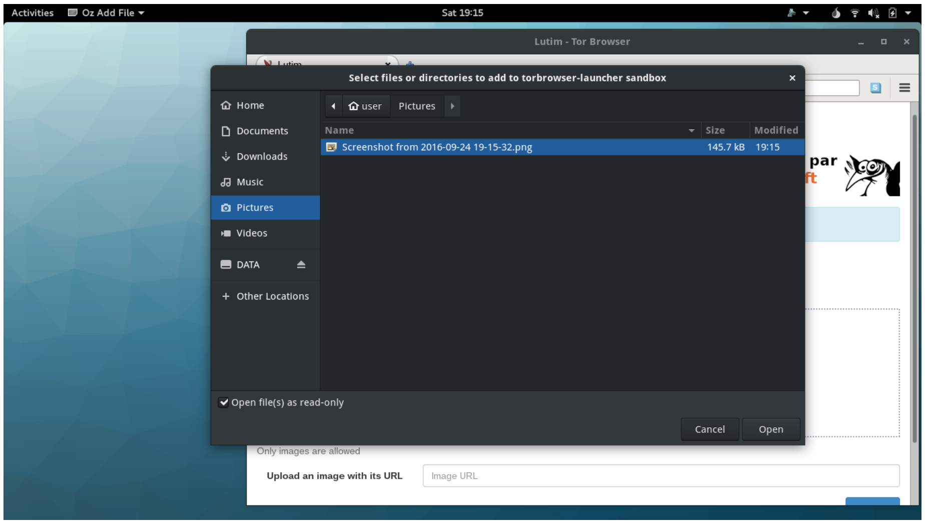
The images are encrypted on the server (Lutim does not keep the key).

□ Delete at first view? □ Keep EXIF tags

Only images are allowed

Figure 7: Oz menu - Add file

You may add more than one file at a time. You may also choose to make these files read—only, meaning that they can only be read and not written to while in the sandbox.



Select files or directories to add to torbrowser-launcher sandbox

Name	Size	Modified
Screenshot from 2016-09-24 19-15-32.png	145.7 kB	19:15

☑ Open file(s) as read-only

Only images are allowed

Upload an image with its URL: Image URL

Figure 8: Oz menu - Select files or directories

Once the file(s) you want to upload are added to the Tor Browser sandbox, you may proceed to upload them normally.

Viewing PDFs

PDFs present security and privacy risks to users. Subgraph OS sandboxes PDFs in a safe environment, minimizing those risks.

PDFs are affected by the following security and privacy risks:

1. PDF readers have security vulnerabilities that can be exploited by opening a malicious PDF
2. PDFs may make outgoing connections to the Internet, compromising the user's privacy either by sending personally identifiable information or network traffic that can be correlated with the user's other activities

To address the first problem, the security hardening in Subgraph OS makes it much more difficult to exploit security vulnerabilities in the PDF reader (**Evince**).

If a malicious PDF bypasses the security hardening in Subgraph OS, it compromises the PDF reader. However, because **Evince** runs inside of a sandbox, this limits what an attacker can do. The sandbox in Subgraph OS is called **Oz**.

The sandbox prevents **Evince** from accessing sensitive files on the computer, such as your encryption keys, email, personal documents, etc. **Evince** only requires access to the PDF(s) it is reading and some other files it needs to operate normally.

Oz also prevents **Evince** from connecting to the Internet. This can prevent malicious code from communicating with the outside world. Privacy is also preserved since **Evince** cannot send data that can *fingerprint* the user.

Lastly, the sandbox limits other types of actions through a Linux feature called *seccomp*.

> *What is a system call?*
>
> System calls provide a way for applications, which run in *user-space*, to ask the kernel (running in *kernel-space*) to do things such as read and write files, communicate over the network, etc.
>
> When a *user-space* application makes a system call to do something such as open a file, the kernel must perform a number of low-level actions. The kernel may be responsible for the file system implementation, authorizing the application to access the file, reading the file contents from the hard-drive, etc. The kernel must run with *elevated privileges* in relation to the application to perform these low-level actions. System calls let applications

cross the boundary between *user-space* and *kernel-space* without requiring the application to run with kernel-level privileges.

System calls are critical to security because they provide an interface for lower-privileged applications to send input to the kernel.

Sandboxed applications in Subgraph OS include a set of policies called a *seccomp whitelist*. If an attacker compromises an application, this security feature can prevent them from gaining elevated privileges on your computer.

What is seccomp?

Seccomp is a security feature of Linux that can restrict access to system calls. If an application tries to run one of the system calls restricted by *seccomp*, it will be killed instead of allowing the system call to run. This can prevent privilege escalation in case malicious code tries to exploit kernel vulnerabilities through system calls. System calls are often used as a *payload* in malicious code to do some things as read files or open network connections. *Seccomp* can also prevent *payloads* from running if they use system calls are that blocked by the policy.

What is a seccomp whitelist?

A *seccomp whitelist* is a list of allowed system calls. If the application tries to call any system call *not* on this list, it is killed by *seccomp*.

What is a seccomp blacklist?

A *seccomp blacklist* is a list of forbidden system calls. If the application tries to call any system call *on* this list, it will be killed by *seccomp*. This is in contract to the whitelist, which blocks the calls *not* on the list.

The **Oz** sandbox in Subgraph OS supports both *seccomp whitelists* and *seccomp blacklists*.

Opening PDFs with Evince in the file explorer

Clicking on a PDF in the file explorer will automatically open the PDF using **Evince** in the **Oz** sandbox.

Adding PDFs to Evince from the Oz menu

If the PDF reader is already open, the PDF can be added to the sandbox by clicking on *Add file…* option of the **Oz** menu for **Evince**.

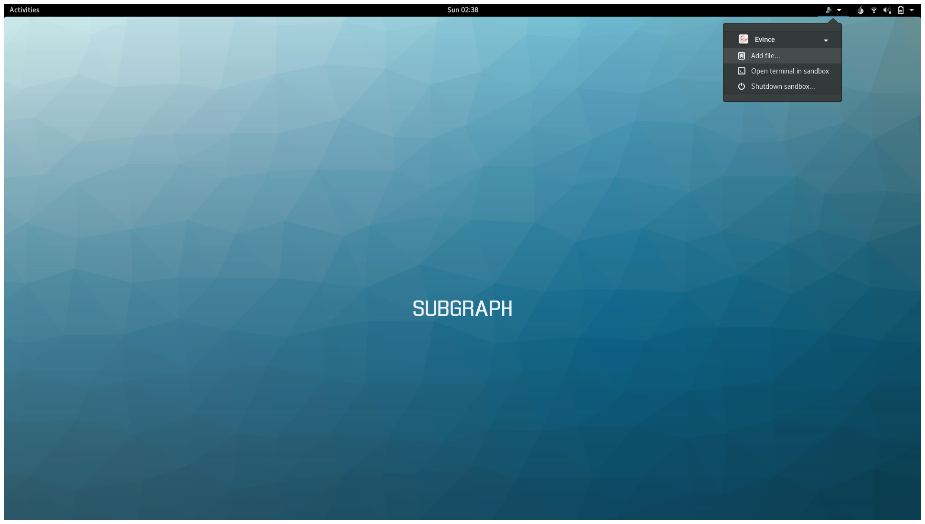
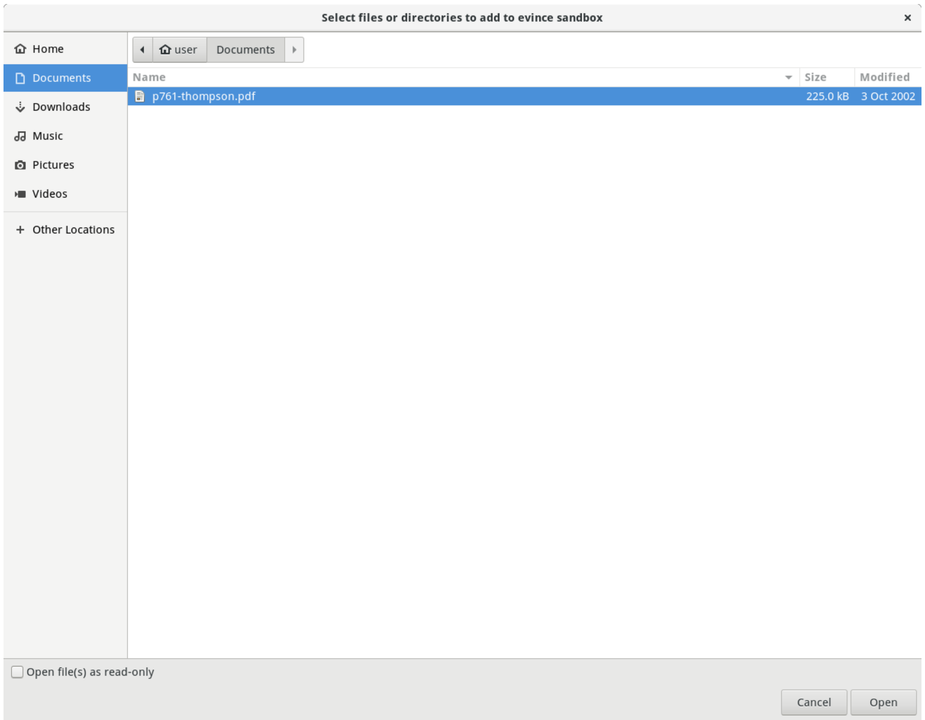


Figure 9: Oz menu - Add file

You may add multiple files. You can also make these files *read-only*, meaning that they cannot be modified in the sandbox.



Select files or directories to add to evince sandbox

Name	Size	Modified
p761-thompson.pdf	225.0 kB	3 Oct 2002

□ Open file(s) as read-only

Figure 10: Oz menu - Select files or directories

Opening PDFs from the command-line terminal

PDFs may also be opened from the terminal.

For example, to open this handbook using **Evince** in the terminal, run the following command:

```
$ evince sgos_handbook.pdf
```

After running the command, you will see the following message:

```
ok received
```

This message indicates that **Oz** has received the request to launch *Evince*.

You may be surprised that opening the PDF from the terminal also opens it in the sandbox. This is because **Oz** re-routes the commands so that they run in the sandbox. For any application that runs in Oz, you may launch it from the desktop *or* the command-line terminal.

Chatting with CoyIM

CoyIM is one of the instant messaging clients in Subgraph OS. **CoyIM** supports the *Jabber/XMPP* instant messaging protocol. All chats are end-to-end encrypted using *OTR* (Off-the-Record) Messaging.

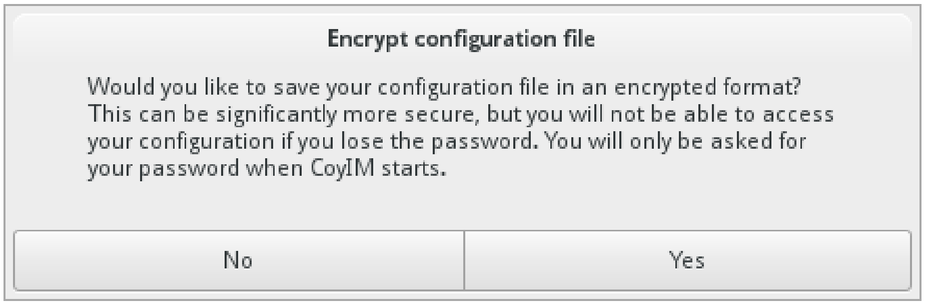
CoyIM is developed by the *ThoughtWorks STRIKE* team as a more secure alternative to chat software such as **Pidgin** and **Adium**.

More information about **CoyIM** can be found here:

https://coy.im/

Adding an XMPP account to CoyIM

When **CoyIM** opens for the first time, it asks you if you want to encrypt your configuration file. We recommend that you encrypt your configuration.

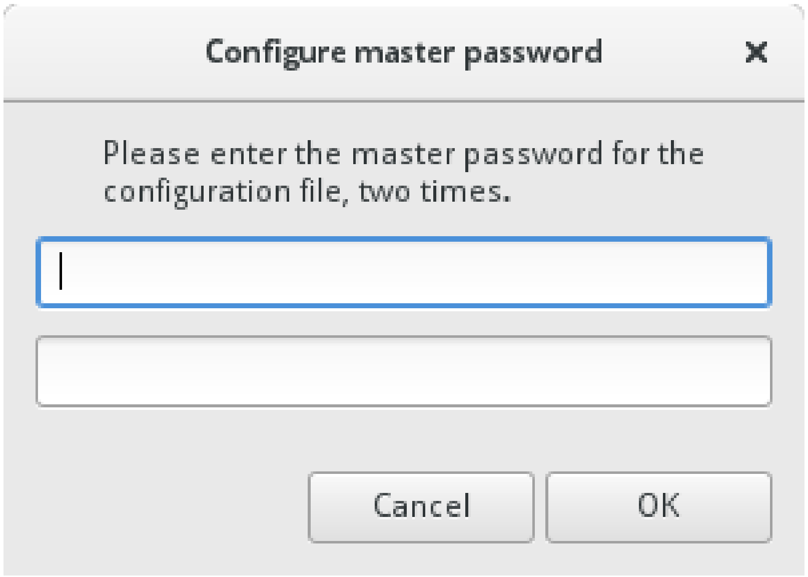


Encrypt configuration file

Would you like to save your configuration file in an encrypted format? This can be significantly more secure, but you will not be able to access your configuration if you lose the password. You will only be asked for your password when CoyIM starts.

Figure 11: CoyIM - Encrypt configuration file

If you have decided to encrypt your configuration file, you will be prompted to configure the master password that will be used to encrypt your configuration file. You will need to re-enter this password each time you use **CoyIM**, so choose something strong but memorable!



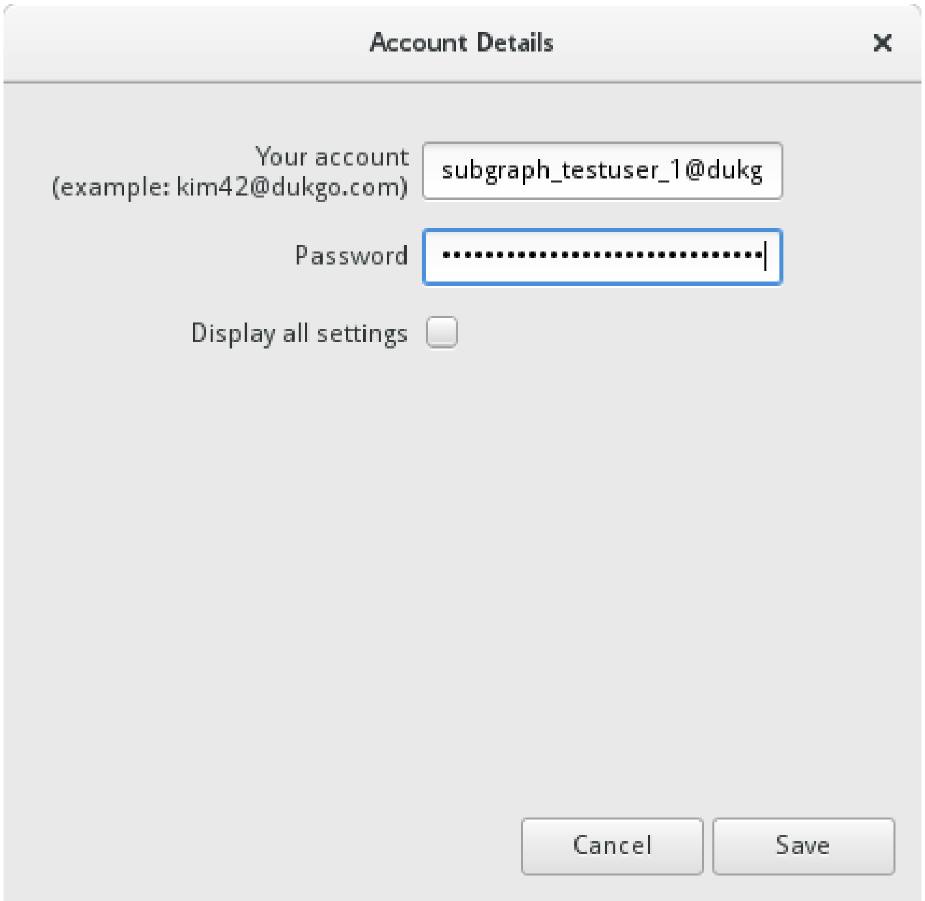
Configure master password

Please enter the master password for the configuration file, two times.

Figure 12: CoyIM - Configure master password

To begin using CoyIM, you must first add an existing account from an *XMPP* network.

Once you had added your account details, you can connect your account. If you have successfully connected to the chat network, a *green* dot will appear to the left of your username.



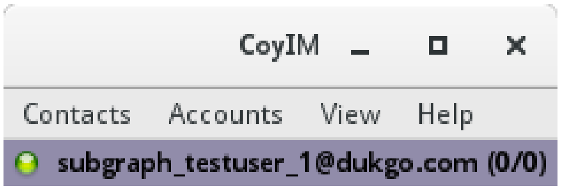
Account Details

Your account (example: kim42@dukgo.com): subgraph_testuser_1@dukg

Password: ••••••••••••••••••••••••••••••••

Display all settings □

Figure 13: CoyIM - Account details: basic configuration



subgraph_testuser_1@dukgo.com (0/0)

Figure 14: CoyIM - Successful connection

Chatting over Tor with Ricochet

Ricochet is an anonymous peer-to-peer instant messaging application. **Ricochet** lets people chat directly with each other over Tor. Unlike other chat services, no intermediate servers are required. This means that **Ricochet** does not store your contact lists and chat histories on a server somewhere in the cloud.

Ricochet is built on top of Tor hidden services. Tor hidden services provide anonymity and end-to-end encryption. This enables people to have conversations that are private and secure.

What is a Tor hidden service?

Tor hidden services provides a means of hosting services on the Tor network. Any type of network service may be hosted as a hidden service (such as web servers, file shares, and instant messaging servers).

Instead of using an IP address or domain name, Tor hidden services are accessed by their *.onion* address. The *.onion* address is an automatically generated name that is derived from the public key of the hidden service.

.onion addresses are only accessible over Tor. **Tor Browser** is one way to access *.onion* addresses. In Subgraph OS, any application can access *.onion* addresses because all applications are routed through Tor.

Tor hidden services provide privacy and anonymity for both the server and the client. Tor hidden services have the following benefits over regular network services:

1. Neither the client nor the server need to know the network location (IP address) of each other. Tor routes traffic through a series of rendezvous points that hide the client IP address from the server. The server's network location (IP address) is also hidden from the client, who connects to the *.onion* address of the server.
2. All traffic between the client and server is end-to-end encrypted. Traffic never leaves the Tor network, meaning that it is only decrypted on either end of the transaction. When Tor is used to connect to the regular Internet, traffic is only encrypted until the *exit-node*. Without using another layer of encryption such as HTTPS, exit nodes can observe traffic. Tor hidden services are not affected by this limitation.

More information about the hidden services protocol can be found here:

https://www.torproject.org/docs/hidden-services.html.en

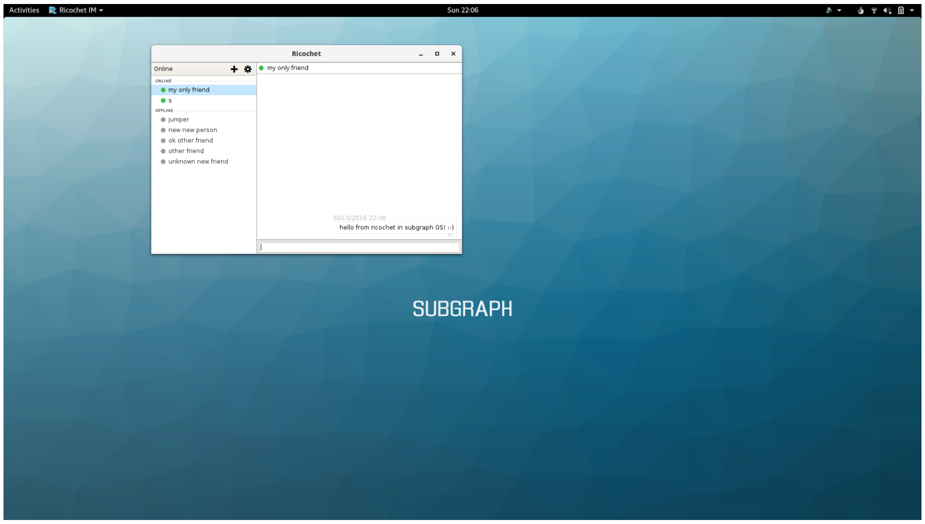
In **Ricochet**, each user has a *contact ID* that maps to a Tor hidden service that is hosted on their computer. The application manages all of the plumbing of creating the hidden service on your computer and communicating with your contacts via their hidden services.

More information about Ricochet can be found on the following pages:

- https://ricochet.im/
- https://github.com/ricochet-im/ricochet/blob/master/doc/design.md

Chatting in Ricochet

Ricochet is similar to other instant messaging clients. The application shows the contacts that are online. You can open chat sessions with your contacts and switch between those sessions like in any other instant messager.

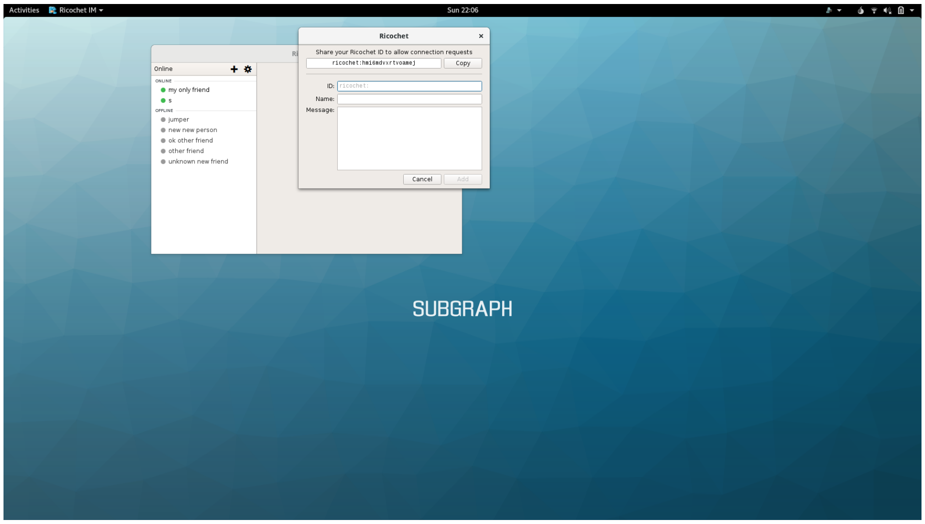


30/10/2016 22:06
hello from ricochet in subgraph OS! :-)

Figure 15: Ricochet - Chatting

Adding a contact in Ricochet

If you know the *contact ID* of another user, you can add them as a contact. To add a contact, click the **+** button in the top-left corner of the application window.



Share your Ricochet ID to allow connection requests

ricochet:hmi6mdvxrtvoamej

ID: ricochet:

Figure 16: Ricochet - Adding a contact

Ricochet contact options

ID: The *contact ID* of the contact you want to add

Name: A nickname for the contact you want to add

Message: The message you want to send when adding the contact

Sharing files with OnionShare

OnionShare is a anonymous, peer-to-peer file sharing application. It lets people share files of any size private and securely.

OnionShare is built on top of Tor hidden services. There are a number of security and privacy advantages to sharing files over Tor hidden services using **OnionShare**.

1. Tor hidden service connections are end-to-end encrypted, meaning that the file transfer is encrypted at every point between the client and server.
2. Tor hidden service connections are anonymous. File transfers can occur without the either the client or the server knowing the IP address of each other. The server is hidden behind an *.onion* address on the Tor network. The client is hidden because it connects to the hidden service through different rendezvous points.
3. *OnionShare* file shares are designed to be short-lived. They can shut down after the file transfer occurs, meaning the server stops listening and the *.onion* address disappears from the Tor network.

Subgraph OS enhances the security of **OnionShare** by sandboxing it with **Oz**. File shares exist in their own sandbox, without access to other sensitive files on the computer. If **OnionShare** is affected by a security vulnerability, running it **Oz** limits the consequences of the vulnerability.

When a user shares files, **OnionShare** starts a hidden service with its own *.onion* address. The user then sends the *.onion* address to the people they wish to share files with. The *.onion* address should be sent over a *secure communication channel*. This is important to prevent unwanted parties from accessing your shared files. Once files are shared, people with the *.onion address* can download the files using the **Tor Browser**.

What is a secure communication channel?

A communication channel is secure if people can communicate with some expectation that their conversation cannot be intercepted or tampered with. Ideally, all communications should be encrypted along with their metadata. Metadata includes things such as the time, date, and frequency of the conversations. It can also include the identities and location of the people who are communicating. Even without the content of a conversation, metadata can reveal a lot about the nature of the communication.

Establishing *truly* secure communications channels is difficult. Many communications tools rely on third-parties, making them privy to communications metadata. This may include the third-party servers themselves or intermediary servers that pass on the communications. Communications, even encrypted ones, often leak metadata as they travel to their final destination.

Subgraph OS includes applications to help our users communicate over secure channels. These examples are ranked according to the amount of metadata they reveal:

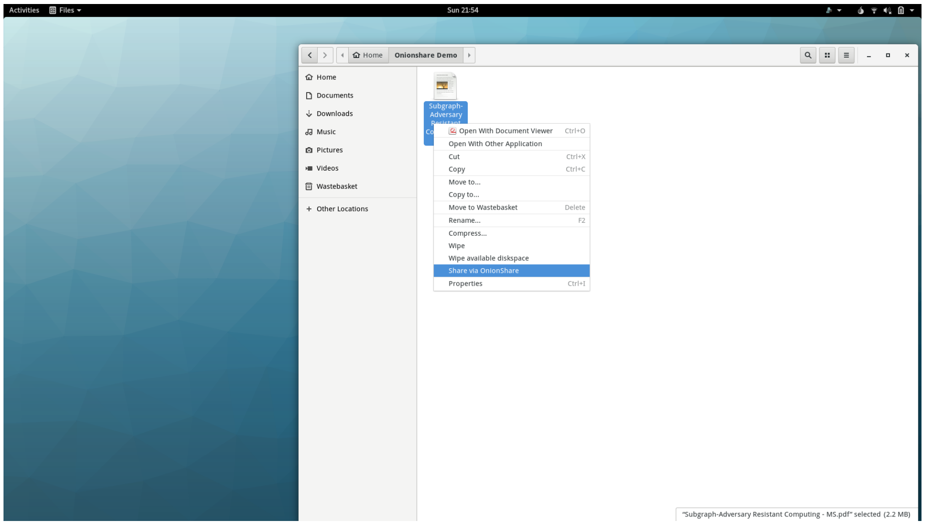
- Ricochet instant messager (uses Tor hidden services for anonymity and end-to-end encryption, no metadata, no third-party servers required)
- CoyIM instant messager (uses the XMPP protocol, some metadata, requires third-party servers)
- Encrypted email using Icedove/Torbirdy (uses the SMTP protocol, lots of metadata, requires third-party servers)

More information about **OnionShare** is on the following website:

https://github.com/micahflee/onionshare

Share via OnionShare

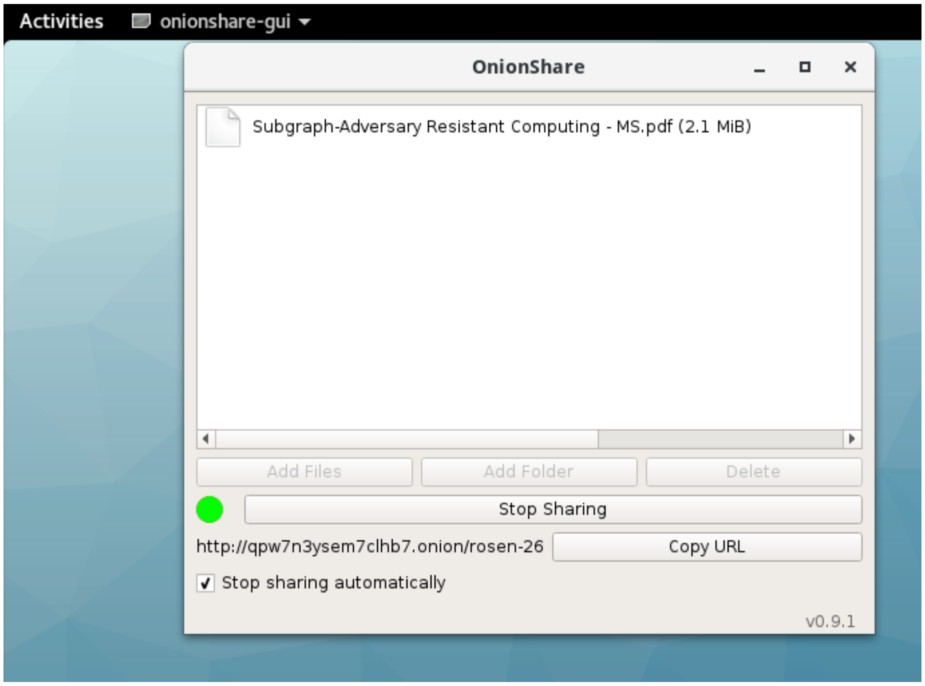
OnionShare is integrated into the file explorer of Subgraph OS. To share a file, *right-click* on the file and select *Share via OnionShare*.



"Subgraph-Adversary Resistant Computing - MS.pdf" selected (2.2 MB)

Figure 17: OnionShare - Share via OnionShare

Selecting *Share via OnionShare* will start **OnionShare** and open the **onionshare-gui**. It may take a few seconds for OnionShare to create the hidden service. The status indicator will turn *green* when it is ready.



Subgraph-Adversary Resistant Computing - MS.pdf (2.1 MiB)

http://qpw7n3ysem7clhb7.onion/rosen-26

☑ Stop sharing automatically

Figure 18: OnionShare - onionshare-gui

onionshare-gui includes the following options to manage shared files:

- Add Files
- Add Folder
- Delete (Files and Folders)
- Stop Sharing (all files and folders, this closes **OnionShare**)

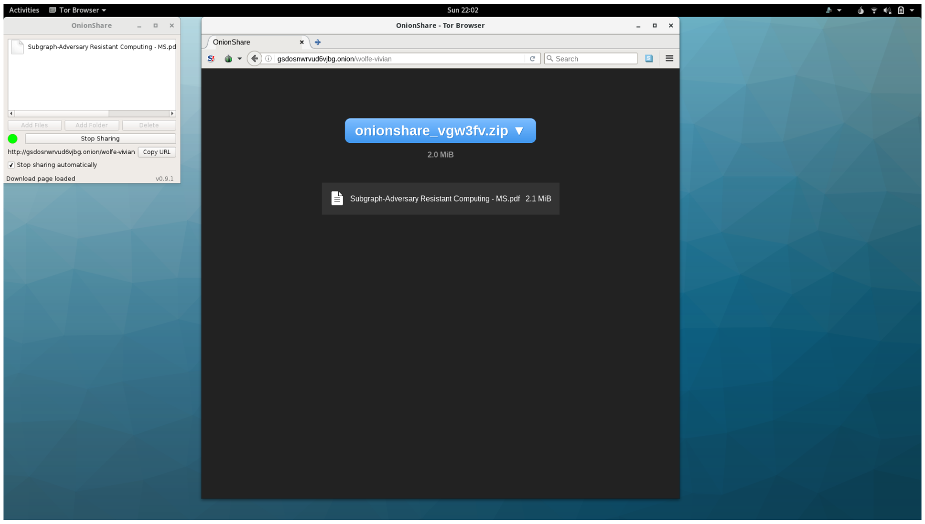
OnionShare runs inside of the **Oz** sandbox. To add files and folders after OnionShare has started, they must be added through the **Oz** menu at the top right corner of the desktop. See the section on *Viewing PDFs* for further information on adding files and folders to an application in the **Oz** sandbox.

The URL for the hidden service (the *.onion* address) is provided along with a button to *Copy URL* to the clipboard. This URL should be sent over a *secure communication channel* to the people you want to share files with.

The *Stop sharing automatically* checkbox determines whether **OnionShare** will close automatically after the file is downloaded by a user. Un-check this option if you are sharing files with multiple users.

Download files from OnionShare

OnionShare runs as a Tor hidden service. To download files over **OnionShare**, you can use the **Tor Browser**. Paste the *.onion* address for the file share into the address bar of **Tor Browser**. This will open the web interface for **OnionShare**.



Subgraph-Adversary Resistant Computing - MS.pd

http://gsdosnwrvud6vjbg.onion/wolfe-vivian

☑ Stop sharing automatically

Download page loaded

gsdosnwrvud6vjbg.onion/wolfe-vivian

onionshare_vgw3fv.zip ▼

2.0 MiB

Subgraph-Adversary Resistant Computing - MS.pdf 2.1 MiB

Figure 19: OnionShare - web interface in Tor Browser

NOTE: In this screenshot, **OnionShare** (the server) and **Tor Browser** (the client) are both running on the same computer. Because the **OnionShare** server is only accessible over a Tor hidden service, **Tor Browser** connects to the file share over Tor. This is the case even if they are running on the same computer. Of course, normally the server and the client would run on different computers

Monitoring outgoing connections with Subgraph Firewall

Subgraph Firewall is an *application firewall* that is included in Subgraph OS. While most firewalls are designed to handle incoming network communications, an *application firewall* can handle outgoing network communications. **Subgraph Firewall** can apply policies to outgoing connections on a per-application basis.

Application firewalls are useful for monitoring unexpected connections from applications. For example, some applications may *phone home* to the vendor's website. Often this activity is legitimate (non-malicious) but it still may violate the user's privacy or expectations of how the software operates. Subgraph Firewall gives users the choice to allow or deny these connections.

Malicious code may also *phone home* to a website or server that is operated by the hacker or malicious code author. Subgraph Firewall can also alert the user of these connections so that they can be denied.

Application firewalls cannot prevent all malicious code from connecting to the Internet. Sophisticated malicious code can subvert the *allowed* connections to bypass the firewall. However, the firewall may alert the user of connection attempts by less sophisticated malicious code.

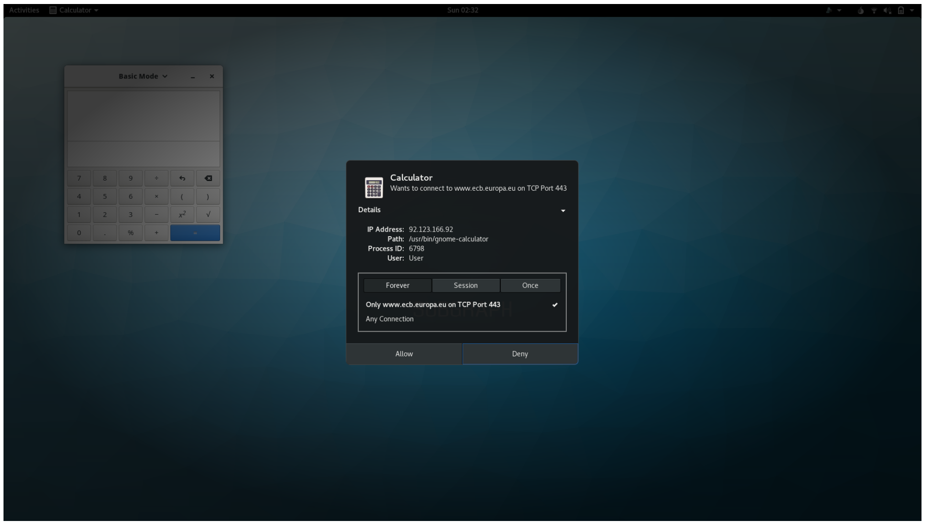
Our *application firewall* makes Subgraph OS unique. It is not found in other Linux distributions. Normally, applications will make outgoing network connections without the knowledge or consent of the user. Subgraph OS helps mitigate these security and privacy risks by making users aware and giving them the power to decide how applications connect to the Internet.

Allowing or denying connections in Subgraph Firewall

When Subgraph Firewall sees a connection it does not have a policy for, it prompts the user to *allow* or *deny* the connection. The prompt includes options to define the duration of the policy and the scope. By scope, we mean apply the policy for the application to a specific destination or to any connection made by the application.

While developing Subgraph Firewall, we noticed some unusual behavior from **Gnome Calculator**. We didn't expect that a calculator would need to connect to the Internet and so we were surprised to see a prompt from **Subgraph Firewall**. **Gnome Calculator** connects to various bank websites to fetch the exchange rates for currency conversions.

This type of unexpected behavior is one of the reasons we created **Subgraph Firewall**. **Gnome Calculator** doesn't give the user the choice to fetch the exchange rates. **Subgraph Firewall** puts that choice back in the hands of the user.



Calculator
Wants to connect to www.ecb.europa.eu on TCP Port 443

Details

IP Address: 92.123.166.92
Path: /usr/bin/gnome-calculator
Process ID: 6798
User: User

Only www.ecb.europa.eu on TCP Port 443 ✓
Any Connection

Figure 20: Subgraph Firewall - allow/deny prompt

Subgraph Firewall Allow/Deny prompt options

At the top of the prompt is the name of the application making the connection as well the destination hostname and port.

IP address: The destination IP address

Path: The path to the application that is making the connection

Process ID: The process ID of the application that is making the connection

User: The user who started the application that is making the connection

Allow/Deny duration

Forever: *Allow* or *Deny* the connection forever (this can be changed afterwards in the Subgraph Firewall settings)

Session: *Allow* or *Deny* the connection until logging out of the desktop session

Once: *Allow* or *Deny* the connection once (the prompt will re-appear if the application attempts the connection again)

Allow/Deny scope

Only hostname on port: Allow/Deny the connection for this application only for the *hostname* and *port* listed at the top of the firewall prompt

Any Connection: Allow/Deny any connection made by the application

Configuring firewall rules in Subgraph Firewall

To configure the firewall rules, select the **Firewall -> Firewall Settings** option from the *Gnome User Menu* at the top right corner of the desktop.

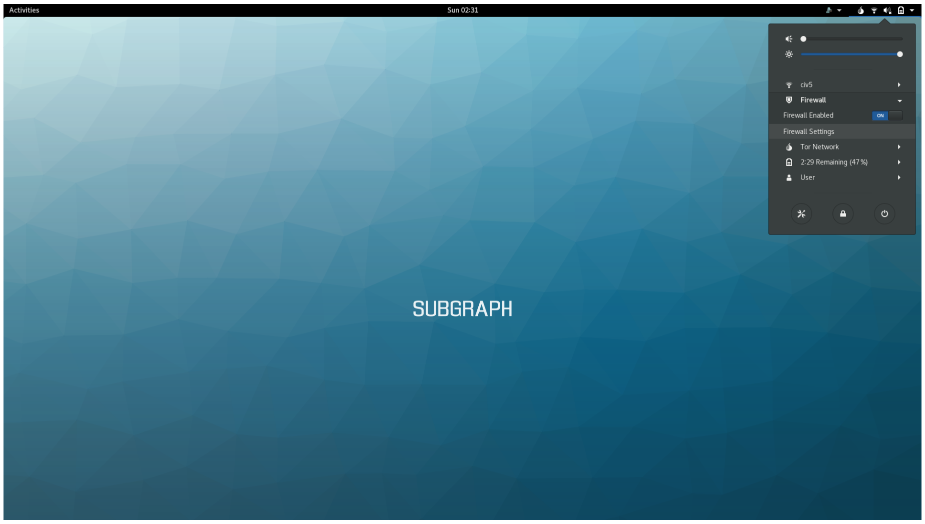
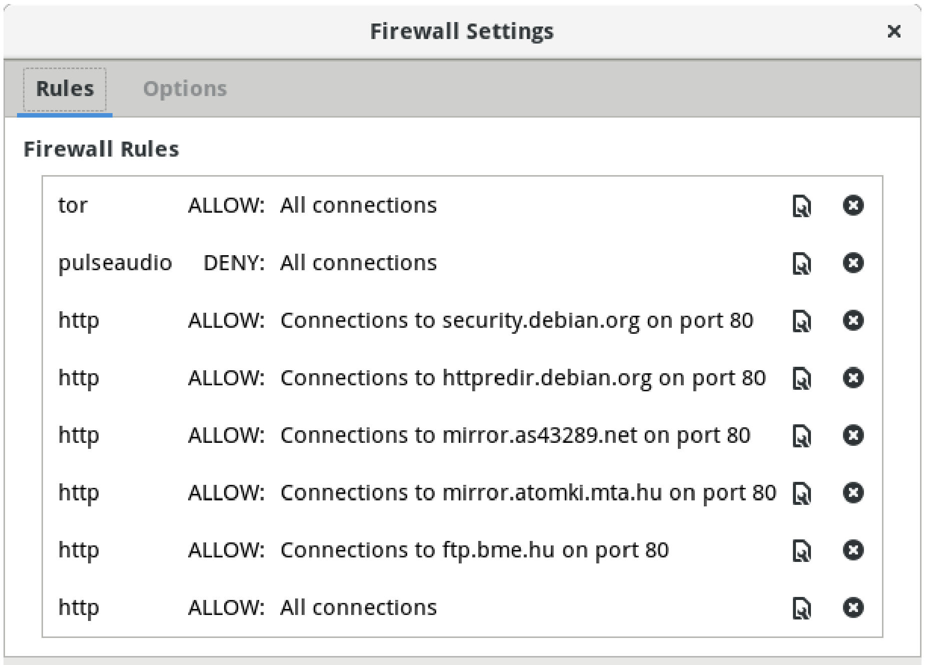


Figure 21: Gnome User Menu - Firewall -> Firewall Settings

This will open the **Firewall Settings** configuration window.



Firewall Rules

tor	ALLOW:	All connections
pulseaudio	DENY:	All connections
http	ALLOW:	Connections to security.debian.org on port 80
http	ALLOW:	Connections to httpredir.debian.org on port 80
http	ALLOW:	Connections to mirror.as43289.net on port 80
http	ALLOW:	Connections to mirror.atomki.mta.hu on port 80
http	ALLOW:	Connections to ftp.bme.hu on port 80
http	ALLOW:	All connections

Figure 22: Subgraph Firewall Settings

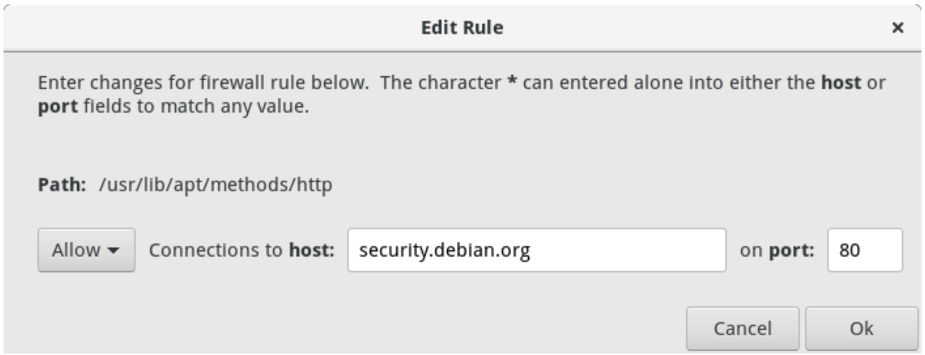
The configuration window shows all of the existing rules.

Each rule has the following columns:

- Application name
- A policy setting of *ALLOW* or *DENY*
- The scope of the policy

The last two options are to *Edit* or *Delete* a firewall rule.

If you click the *Edit* button (the button with the wrench), you will be prompted to edit the *Allow/Deny* policy and its scope.



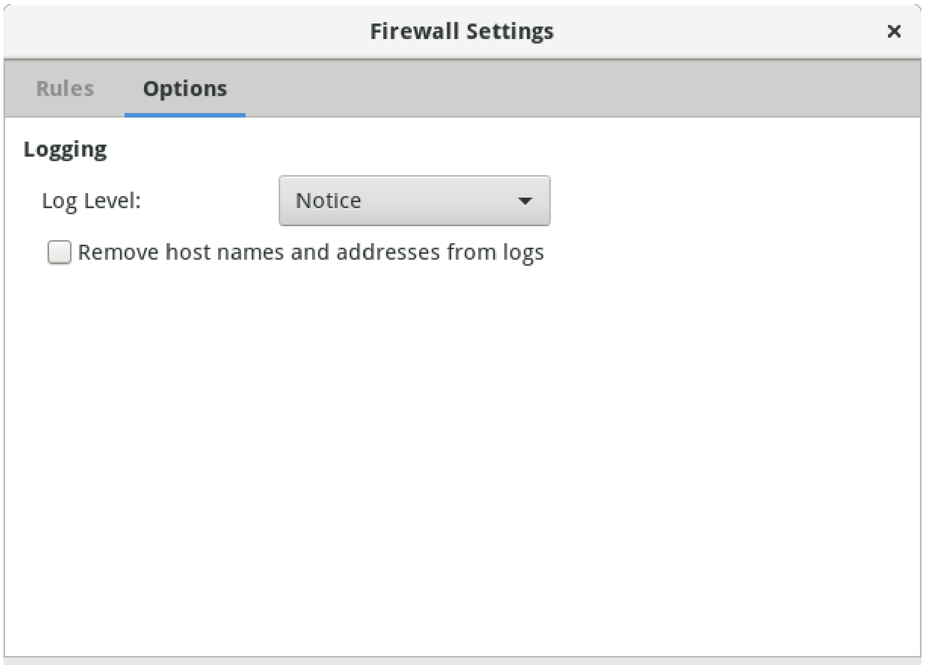
Enter changes for firewall rule below. The character ***** can entered alone into either the **host** or **port** fields to match any value.

Path: /usr/lib/apt/methods/http

Allow Connections to **host:** security.debian.org on **port:** 80

Figure 23: Subgraph Firewall Settings - Edit Rule

The *Options* tab of the **Firewall Settings** window lets you configure general options for **Subgraph Firewall**.



Firewall Settings

Logging

Log Level: Notice

□ Remove host names and addresses from logs

Figure 24: Subgraph Firewall Settings -Options

Features and advanced usage

This chapter describes the unique features of Subgraph OS. These are features that distinguish it from other operating systems. This is where you can find more information about the design of Subgraph OS.

As an *adversary resistant computing* platform, Subgraph OS is designed to resist threats to security and privacy. This chapter includes more information about how our design addresses these threats. We also provide links to external sources where more in-depth technical information is available.

This chapter also provides documentation for some advanced use-cases in Subgraph OS. This content is more technical than previous chapters in the Subgraph OS Handbook. It contains *how-tos* and reference materials intended for users who are comfortable running commands in the terminal and editing configuring files.

Sandboxing applications with Subgraph Oz

Subgraph OS runs desktop applications inside of our security sandbox (Oz). The security sandbox is an additional layer of security, above and beyond the other security features of Subgraph OS. Subgraph OS is hardened to make it very difficult for an attacker to compromise applications. However, it is impossible to prevent every vulnerability. If an attacker compromises an application, Oz can help to protect against further compromise of the computer and the user's sensitive files.

Oz can provide the following protections to sandboxed applications:

- Restrict the files that the application has access to
- Restrict network access
- Restrict audio playback
- Restrict the system calls the application can make (using **seccomp**)
- Restrict malicious interactions between X11 applications (using **xpra**)

Each sandboxed application has its own policies to restrict its capabilities.

The following table shows some of the sandbox policies in Subgraph OS:

Application	Category	Network?	Audio?
Tor Browser	Browser	Proxy port	Yes
Icedove	Email client	Proxy port	No
CoyIM	Instant messager	Proxy port	No
Ricochet	Instant messager	Proxy port	No
Hexchat	IRC client	Proxy port	No
OnionShare	File sharing	Proxy port	No
VLC	Video player	No	Yes
LibreOffice	Office suite	No	No
Evince	PDF reader	No	No
Eog	Image Viewer	No	No

Oz also sandboxes desktop applications from each other. Normally, applications running under the X11 display server can interact with each other. This means that one application can intercept or inject events into another application.

Without Oz or an alternate display server, there is no way to securely prevent applications from interacting with each other. An attacker could abuse this to perform malicious actions such as intercepting the keystrokes from another desktop application. To prevent these attacks, Oz sandboxes use **xpra** to render applications on the desktop. **Xpra** isolates applications by using a separate display server to render each application. Since the applications do not share the same display server, they cannot interact.

For more technical details about Oz and its security features, see the following page:

https://github.com/subgraph/oz/wiki/Oz-Technical-Details

Enabling an Oz profile

Oz profiles can be found in the following directory:

```
/var/lib/oz/cells.d
```

Oz automatically enables profiles in this directory. However, if you need to manually enable a profile, you can do so by running the **oz-setup** command to *install* the profile.

The following example installs the profile for **evince**:

```
$ sudo oz—setup install evince
```

When the profile is installed, Oz will *divert* the path of the program executable. Instead of the program running directly, diverting it lets Oz start the program. So the next time it is started, the program will be sandboxed by Oz.

Disabling an Oz profile

If you want to run a previously sandboxed program outside of the sandbox, you must disable its profile. To disable a profile, run the **oz-setup** command with the *remove* option.

The following example removes the profile for **evince**:

```
$ sudo oz—setup remove evince
```

When the profile is removed, Oz will undo the *divert* of the program path. The program will not run in the Oz sandbox the next time it is started.

Viewing the status of an Oz profile

The status of a program can also be viewed with the **oz-setup** command.

The following example shows the status of **evince**:

```
$ sudo oz—setup status /usr/bin/evince
```

The command prints the following when **evince** profile is installed:

```
Package divert is installed for:     /usr/bin/evince
Package divert is installed for:     /usr/bin/evince—thumbnailer
Package divert is installed for:     /usr/bin/evince—previewer
```

When the **evince** profile is not installed, the command prints the following:

```
Package divert is not installed for: /usr/bin/evince
Package divert is not installed for: /usr/bin/evince—thumbnailer
Package divert is not installed for: /usr/bin/evince—previewer
```

Creating an Oz profile

In this section, we will walk through some of the options in a basic profile.

Oz profiles are written in JSON.

The following is the Oz profile for the **eog** image viewer:

```
{
 ”name”: ”eog”
  , ”path”: ”/usr/bin/eog”
  , ”allow_files”: true
  , ”xserver”: {
          ”enabled”: true
          , ”enable_tray”: false
          , ”tray_icon”:”/usr/share/icons/hicolor/scalable/apps/eog.svg”
  }
  , ”networking”:{
          ”type”:”empty”
  }
  , ”whitelist”: [
          {”path”:”/var/lib/oz/cells.d/eog—whitelist.seccomp”, ”read_only”: true}
  ]
  , ”blacklist”: [
  ]
  , ”environment”: [
          {”name”:”GTK_THEME”, ”value”:”Adwaita:dark”}
          , {”name”:”GTK2_RC_FILES”,
”value”:”/usr/share/themes/Darklooks/gtk—2.0/gtkrc”}
  ]
  , ”seccomp”: {
          ”mode”:”whitelist”
          , ”enforce”: true
          , ”whitelist”:”/var/lib/oz/cells.d/eog—whitelist.seccomp”
  }
  }
```

Example Oz profile configuration options

name: The name of the profile

path: The path to the program executable

allow_files: Allow files to be passed as arguments to the program (such as image files for **eog**)

xserver -> enabled: Enable the use of the Xserver (**xpra**) *xserver* -> *enable_tray*: Enable the **xpra** diagnostic tray (defaults to false, enabling it requires extra software)

xserver -> *tray_icon*: The path to the tray icon

networking -> *type*: The networking configuration type, *empty* disables networking entirely

whitelist -> *path*: The path of a file to add to the sandbox, in this case it is the *seccomp whitelist* for **eog**

whitelist -> *path* -> *read_only*: Whether or not the allowed file is *read-only*, should be *true* in most cases

blacklist: Removes access to a file in the sandbox, accepts the *path* argument

environment -> *name, value*: Adds environment variables by name and value to the sandbox

seccomp -> *mode*: Adds a seccomp policy (either *whitelist* or *blacklist*) to the sandbox

seccomp -> *enforce*": The seccomp enforcement mode

seccomp -> *whitelist*: The path to the whitelist policy

Oz supports a number of different profile configurations. More examples for real applications are located in the profiles directory:

```
/var/lib/oz/cells.d
```

Complete documentation for creating Oz profiles can be found here:

https://github.com/subgraph/oz

Securing system calls with seccomp in Oz

Seccomp is a feature of the Linux kernel to limit exposed system calls. As system calls provide a user interface to the kernel, they expose it to attacks. These attacks can let an attacker elevate their privileges on the computer. The Oz sandbox uses *seccomp* to protect against this type of attack.

Oz supports *seccomp* policies on a per-application basis. *Seccomp* kills applications whenever they violate a policy. This protects the computer in cases where an attacker tries to exploit a vulnerability in the kernel that depends on the blocked system call.

Some attacks also use system calls as part of their *payload*. A *payload* is the malicious code that runs as a result of a successful exploit. The *seccomp* policies in Oz can prevent *payloads* from running if they use a blocked system call.

Oz supports **whitelist** or **blacklist** policies. Whitelist policies are *default deny*. This means that only system calls that are explicitly permitted will be allowed. All other system calls (those not on the **whitelist**) cause the application to be killed.

Blacklist policies are *default allow*. This means that seccomp blocks system calls in the blacklist policy but allows all others (those not on the **blacklist**).

Whitelist policies are appropriate when the application is well understood. By well understood, we mean that the behavior of the application is predictable enough to create a precise profile of allowed system calls. This is more secure than a **blacklist** because known behavior of the application is allowed but unknown behavior is blocked. The disadvantage of this approach is that the **whitelists** must be updated regularly to reflect the known behavior of the application.

Blacklist policies are appropriate for applications that are not as well understood. We use them prior to the creation of a **whitelist** or if there is some other reason a **whitelist** cannot be created.

Oz includes a generic **blacklist** that will work out-of-the-box with many applications. This policy blocks unusual or exotic system calls that applications do not normally use.

The Oz generic **blacklist** is located here:

```
/var/lib/oz/cells.d/generic—blacklist.seccomp
```

In Subgraph OS, we try to create **whitelist** policies for all of our supported applications.

Profiling applications with oz-seccomp-tracer

Oz includes a tool to help with the creation and maintenance of seccomp **whitelists**. The **oz-seccomp-tracer** profiles applications as they run to determine the system calls that they use. This tool will generate a seccomp whitelist after it exits.

To profile Firefox using **oz-seccomp-tracer**, run the following command:

```
$ oz—seccomp—tracer —trace —output firefox—whitelist.seccomp /usr/bin/firefox 2>firefox_syscal
```

You can then use Firefox as you normally would. When you are finished, a seccomp **whitelist** will be saved to **firefox-whitelist.seccomp**. **oz-seccomp-tracer** prints all of the system calls from the application to **stdout**. So we also advise you to redirect this output to a separate file. We use **firefox_syscalls.txt** in this example. You could also redirect this output to **/dev/null** if you don't want to save it.

Adding a seccomp policy to an Oz application profile

Once you are satisfied with the **whitelist**, you can copy it to the following directory:

```
/var/lib/oz/cells.d
```

Using Firefox as an example, the following snippets from */var/lib/oz/cells.d/firefox.json* show how to apply the policy.

First, the seccomp policy file must be added to the list of files allowed in the sandbox:

```
  , ”whitelist”: [
          , {”path”:”/var/lib/oz/cells.d/firefox—whitelist.seccomp”, ”read_only”: true}
```

Then the seccomp policy needs to be enabled to run in *enforce* mode:

```
  , ”seccomp”: {
          ”mode”:”whitelist”
          ,
”whitelist”:”/var/lib/oz/cells.d/firefox—whitelist.seccomp”
          , ”enforce”: true
  }
```

Lastly, the Oz daemon must be restarted to load the seccomp policy. You should save your work at this point as restarting Oz will close all of the open sandboxes. To restart the Oz daemon, run the following command:

```
$ sudo systemctl restart oz—daemon.service
```

Anonymizing communications with Tor

Tor is an essential privacy tool that provides anonymity to its users. In particular, Tor hides the location of its users. By location, we mean your IP address (which can also be used to geo-locate your computer).

Tor hides your location by relaying your traffic through a random series of network connections (called a *circuit*). While your traffic passes through the *hops* in this circuit, the source and destination of the traffic are hidden. The traffic eventually leaves the *circuit* through an *exit node*. The *exit node* relays the traffic to its final destination but is also unaware of the source. They are called *exit nodes* because they are the point where the traffic leaves the Tor network to reach its destination on the regular Internet. *Exit nodes* may observe or tamper with the traffic en-route to its destination, unless an additional layer of encryption is applied such as *TLS*.

Due to the possibility that some *exit nodes* are malicious, we strongly advise you to use Tor with an additional layer of encryption. This means connecting to websites over *HTTPS* only, using TLS with applications such as **Icedove** or **Hexchat**, etc.

NOTE: Tor hidden services provide a way to send network traffic *only* through the Tor network. This eliminates the risks involved when the traffic passes through an *exit node* to the regular Internet. However, this requires that the destination service is configured to run as a hidden service. It also adds more latency to the network traffic because it must pass through more *hops* to reach the hidden service. Tor hidden services are discussed in further detail in other sections of this book.

More information about Tor can be found here:

https://www.torproject.org/about/overview

Tor integration in Subgraph OS

Subgraph OS is integrated with the Tor anonymity network. We include many applications that are designed to be used with Tor. These include:

- **Tor Browser** for browsing the web anonymously and accessing Tor hidden service websites
- **OnionShare** for sharing files anonymously over Tor
- **Ricochet** for chatting anonymously of Tor
- **CoyIM** instant messager, which supports connecting to the *.onion* addresses for *XMPP/Jabber* chat servers

Other parts of Subgraph OS are engineered to integrate with Tor seamlessly. The **Metaproxy** routes non-Tor applications over Tor. Our **Oz** sandbox also lets applications work seamlessly with Tor. We also include a Gnome Shell extension that monitors that status of connections to the Tor network. Lastly, **ROFLCopTor** is a filter for the Tor control port that enforces security policies on applications that run Tor control commands.

Routing applications through Tor with Subgraph Metaproxy

The **Metaproxy** is an important part of Subgraph OS. It runs in the background to help applications connect through the Tor network. This is done transparently, even with applications that are not configured or designed to work with Tor.

On other operating systems, users must specifically configure applications to connect to the Internet through Tor. This normally requires the user to configure proxy settings of the application to connect through Tor's built-in proxies. Some applications do not support or honor proxy settings. To use Tor with these applications, users often run them with using a command-line helper called **torsocks** to *torify* the application. This is a lot of work for users.

Configuring proxies or *torifying* applications by hand is not an adequate solution for Subgraph OS. Usability and maintainability are issues with this approach. In Subgraph OS, some applications simply would not work if there is no easy way to route them through Tor. This is because Subgraph OS blocks outgoing connections that are not routed through Tor. This is to prevent accidental privacy leaks. If an application has no way to communicate over Tor, it may not be able to access the network at all.

The **Metaproxy** addresses this problem by automatically relaying outgoing connections through Tor. When we say this is done transparently, we mean the following two things:

1. Users do not have to manually *torify* their applications or otherwise configure them to use Tor
2. Applications that are already configured to use Tor are ignored by the **Metaproxy**, therefore, it only helps those applications which need it

Securing the Tor control port with ROFLCopTor

The Tor service is managed by a control protocol. This lets users perform various actions such as querying information about Tor connections, starting hidden services, and changing configuration options. However, most applications do not need all of these features. These extra features may actually introduce security and privacy risks if someone gains unauthorized access to the control port. To mitigate these risks, Subgraph OS includes a control port filter called *ROFLCopTor*.

ROFLCopTor is a proxy server that is placed between Tor control clients and the Tor control server port. *ROFLCopTor* handles authentication itself, meaning clients do not need to know the authentication credentials or run with higher privileges to access to Tor control port. It intercepts the incoming commands and outgoing responses. *ROFLCopTor* enforces policies on a per-application basis for the traffic between the Tor control clients and the server.

ROFLCopTor supports policies that are bi-directional. This means that a policy can filter both the incoming commands and the outgoing responses from the Tor control port. Policies can also replace command and response strings. Replacements can be used to filter sensitive information from Tor control port responses.

ROFLCopTor has a number of default policies for applications in Subgraph OS that require access to the Tor control port. The policies work without modification for most use-cases. This section describes how to profile applications to create new policies or modify existing ones.

Profiling applications with ROFLCopTor

ROFLCopTor can profile applications to determine the Tor control commands that they run on a regular basis. This makes it easier to create or edit policies.

Before profiling applications, you should stop the currently running version of *ROFLCopTor*:

```
$ sudo systemctl stop roflcoptor
```

To begin profiling, you must start *ROFLCopTor* in *watch* mode:

```
$ sudo —u roflcoptor roflcoptor watch —log_level DEBUG —config /etc/roflcoptor/roflcoptor_conf
```

The log shows some of the commands that applications tried to run:

```
18:21:53 — DEBU 017 connection received tcp:127.0.0.1:44860 —>
tcp:127.0.0.1:9051
18:21:55 — ERRO 018 filter policy for gnome—shell—torstatus DENY: A—>T: [GETCONF
ORPort]
18:21:55 — ERRO 019 filter policy for gnome—shell—torstatus DENY: A—>T: [GETINFO
events/names]
18:21:55 — ERRO 01a filter policy for gnome—shell—torstatus DENY: A—>T:
[SETEVENTS NOTICE NS NEWDESC NEWCONSENSUS]
18:21:55 — ERRO 01b filter policy for gnome—shell—torstatus DENY: A—>T: [GETINFO
process/user]
18:21:55 — ERRO 01c filter policy for gnome—shell—torstatus DENY: A—>T: [GETINFO
process/pid]
...
```

Press **Ctrl-C** to stop the *ROFLCopTor* watch process. Make sure to restart *ROFLCopTor* normally after you are done profiling. Run the following command to restart *ROFLCopTor*:

```
$ sudo systemctl restart roflcoptor
```

Editing ROFLCopTor policies

Once you have a list of commands required by an application, you can create or edit a policy.

ROFLCopTor policies are written in JSON. Policies can be found in the following directory on Subgraph OS:

```
/etc/roflcoptor/filters/
```

The following is a simple policy for the Tor Status Gnome shell extension in Subgraph OS:

```
{
    ”Name”: ”gnome—shell—torstatus”,
    ”AuthNetAddr” : ”tcp”,
    ”AuthAddr” : ”127.0.0.1:9051”,
    ”client—allowed” : [”GETINFO status/bootstrap—phase”, ”SIGNAL NEWNYM”],
    ”client—allowed—prefixes” : [],
    ”client—replacements” : {},
    ”client—replacement—prefixes” : {},
    ”server—allowed” : [”250 OK”],
    ”server—allowed—prefixes” : [”250−status/bootstrap—phase=”],
    ”server—replacement—prefixes” : {}
}
```

ROFLCopTor policy configuration options

Name: The name of the application to apply the policy to

AuthNetAddr: The protocol used by the Tor control port

AuthAddr: The address of the Tor control port

client-allowed: The list of commands allowed by the client

client-allowed-prefixes: A list of prefixes for partial allowed client commands (commands where the suffix varies) *client-replacements*: A list of commands to replace and their replacement strings

client-replacement-prefixes: A list of client command prefixes to replace and their replacement strings (for commands where the suffix varies)

server-allowed: The list of responses allowed by the server

server-allowed-prefixes: A list of prefixes for partial allowed server responses (responds where the suffix varies)

server-replacement-prefixes: A list of server response prefixes to replace and their replacement strings (for responses where the suffix varies)

The most common configuration task is to add new commands and responses to the *client-allowed*, *client-allowed-prefixes*, *server-allowed*, and *server-allowed-prefixes* options.

More documentation on configuring and using ROFLCopTor is located on the following page: https://github.com/subgraph/roflcoptor

Hardening the operating system and applications with Grsecurity

Grsecurity is a third-party security enhancement to the Linux kernel. It is developed and maintained by the **Grsecurity** team. It is implemented as a patch to the upstream Linux kernel. Subgraph OS ships with a kernel that is patched with **Grsecurity**.

For more information about **Grsecurity**, see the following page:

https://grsecurity.net/

Configuring PaX flags with Paxrat

Paxrat is a utility in Subgraph OS for maintaining the *PaX flags* of applications on the computer.

> *What is PaX?*
>
> *PaX* is a feature of *Grsecurity* that provides *memory protection*. Many security vulnerabilities in applications and the Linux kernel allow attackers to corrupt process memory. *Memory corruption* can be exploited to run the attackers *payload* of malicious code.
>
> PaX protects the computer from *memory corruption* using a number of novel techniques such as:
>
> 1. Randomizing the layout of process memory or *ASLR* (Address Space Layout Randomization), making it harder for attackers to guess where their malicious *payload* is stored in process memory
> 2. Making memory pages *non-executable*, meaning that an attacker's *payload* cannot run if stored in *non-executable* memory
>
> *PaX* includes other *memory protection* and *control flow integrity* features so that it is more difficult for attackers to exploit *memory corruption* vulnerabilities in applications and the kernel.
>
> *PaX* does not prevent all vulnerabilities but it complicates attacks. The difference to an attacker is that with *PaX* they may be required to exploit multiple vulnerabilities to achieve the same effect as a single vulnerability.
>
> More information about *PaX* can be found here:
>
> https://pax.grsecurity.net/

PaX works by killing applications that violate its security policies. This *proactively* prevents attacks from succeeding. However, as part of their normal functions, some applications perform non-malicious actions that violate the security policies. *PaX flags* are exceptions to these policies. They let applications run normally without being killed by *PaX* when they perform an action that appears to violate policies.

Applications such as web browsers need *PaX flags* to be set because they perform actions such as *JIT* (Just in Time compilation). To *PaX*, *JIT* has the same profile as an attack. Applications that use a *JIT* compiler must be flagged as exceptions so that they are not killed.

Paxrat keeps track of the *PaX flags* for applications in Subgraph OS. It is designed to maintain the *PaX* flags between application updates. **Paxrat** runs when the system updates software, automatically re-applying flags to upgraded applications.

Paxrat can only maintain the flags it knows about. If a user discovers that *PaX* is killing an application, the configuration must be changed to disable some *PaX flags*. Instructions are provided in this guide for changing the **Paxrat** configuration. We also advise users to report the exception to us so that we can update the configuration for everybody.

Paxrat configuration files are written in JSON. They are stored in the following directory:

```
/etc/paxrat
```

The following is a snippet of a *PaX flag* configuration for **Tor Browser**:

```
”/home/user/.local/share/torbrowser/tbb/x86_64/tor—browser_en—US/Browser/firefox”:
{
    ”flags”: ”m”,
    ”nonroot”: true
}
```

Paxrat configuration options

The first line of the configuration (in quotes) is the path to the application. In the above example, it is: “/home/user/.local/share/torbrowser/tbb/x86_64/tor-browser_en-US/Browser/firefox”

flags: This a string of letters representing the various PaX flags

nonroot: This indicates whether the application is owned by the *root* user or not, it is *false* by default but *true* in the example because the **Tor Browser** application is owned by a normal user

NOTE: As a security precaution, **Paxrat** will not apply *PaX flags* to an application that is owned by a *nonroot* user unless the *nonroot* option is set to *true*.

There are a number of different *PaX flags* that can be *enabled* or *disabled*. Most are *enabled* by default and must be *disabled*. *Disabled* flags are represented by a lower-case letter such as **m**. Upper-case letters such as **M** represent *enabled* flags.

PaX flags

P/p: *Enable/disable* PAGEXEC

E/e: *Enable/disable* EMUTRAMP

M/m: *Enable/disable* MPROTECT

R/r: *Enable/disable* RANDMAP

X/x: *Enable/disable* RANDEXEC

S/x: *Enable/disable* SEGMEXEC

A detailed description of these flags can be found on the following page:

https://en.wikibooks.org/wiki/Grsecurity/Appendix/PaX_Flags

Working examples can be found in the Subgraph OS **Paxrat** configuration files:

```
/etc/paxrat/paxrat.conf
```

Applying PaX flags

PaX flags must be re-applied after any configuration changes. Run the following command to re-apply *PaX flags*:

```
$ sudo paxrat
```

Anonymizing MAC addresses with Macouflage

MAC addresses are the unique identifiers for the network interface on the computer (such as Ethernet ports and WIFI cards). Due to their unique nature, they can also compromise the privacy of the user.

When connecting to a network, it is possible for other devices on the network to see the MAC address of the network interface that is connected. While this is not much of a concern on networks you trust such as your home network, it may compromise your privacy on those who do not trust. On untrustworthy or hostile networks, uniquely identifying characteristics such as the MAC address may allow others to track your computer.

Subgraph OS mitigates this privacy risk by always creating random MAC addresses for all of your network interfaces. Each time one of your interfaces connects to a network, it will use a different MAC address. This helps to anonymize you across different networks or when connecting to the same network over and over again.

Preventing unauthorized USB access with USB Lockout

USB Lockout is a background feature in Subgraph OS. It protects your computer from unauthorized USB access while your desktop session is locked or you have logged out.

USB Lockout is intended for situations where your computer must be left unattended for short periods. Particularly, in situations where you do not fear your computer will be stolen but you do do not want to expose it to other risks while unattended.

Normally, when you lock the screen or logout, people may still insert a malicious USB device into the computer. While the computer is running, a malicious device can easily compromise it. **USB Lockout** denies all access for new USB devices while the screen is locked or the user is logged out.

USB Lockout works by monitoring the state of the desktop session. When the session is locked or logged out, **USB Lockout** enables the Grsecurity *Deny New USB* setting. When the user unlocks the screen or logs back in, this setting is disabled, allowing access to new USB devices once again.

See the following page page for more information about the Grsecurity *Deny New USB* feature:

```
https://en.wikibooks.org/wiki/Grsecurity/Appendix/Grsecurity_and_PaX_Configuration_Options#Deny_new_USB_
```

Enabling/disabling USB Lockout

While **USB Lockout** runs automatically in the background, you can manually *enable* or *disable* it.

Run the following command to *enable* **USB Lockout**:

```
$ usblockout --enable
```

Run the following command to *disable* **USB Lockout**:

```
$ usblockout --disable
```

Using virtual machines in Subgraph OS

Contrary to popular belief, there is nothing that stops the use of virtual machines in Subgraph OS. While the *Grsecurity* kernel is not compatible with VirtualBox, **Qemu/KVM** works as expected. However, you must install **Qemu/KVM** yourself if you want to run virtual machines.

Running the following command with install **Qemu/KVM**:

```
$ sudo apt install qemu—system qemu—kvm qemu—utils
```

Creating a virtual machine with Qemu

The following sections are recipes on how to use **Qemu/KVM** in Subgraph OS. They are similar to our own workflows for developing and testing Subgraph OS. **Qemu/KVM** supports many more options than what we use in these tutorials. For more detailed information regarding the operation of **Qemu/KVM** virtual machines, see the official **Qemu** manual:

http://wiki.qemu.org/Manual

There are multiple third-party graphical user interfaces for **Qemu/KVM**. These may make it easier to configure and manage virtual machines. You can explore the various options by visiting these pages:

- https://wiki.gnome.org/Apps/Boxes
- http://virt-manager.et.redhat.com
- http://qemuctl.sourceforge.net
- https://launchpad.net/virtualbrick

Creating a basic Linux virtual machine

Prior to creating the virtual machine, you should create a virtual hard-drive image for it:

```
$ qemu—img create —f qcow2 disk.qcow2 8G
```

Your virtual hard-drive is now ready for use. Run the following command to test a virtual machine with the hard-drive:

```
$ qemu—system—x86_64 —enable—kvm —hda ./disk.qcow2 —m 4096
```

To start a virtual machine with an operating system ISO attached to the virtual CDROM, run the following command:

```
$ qemu—system—x86_64 —enable—kvm —hda ./disk.qcow2 —m 4096 —cdrom ./subgraph—os—alpha_2016—06—
```

Qemu/KVM options

-*enable-kvm*: enables **KVM** virtualisation, which is faster than **Qemu's** emulation

-*hda*: This attaches the virtual hard-drive you created

-*m*: This allocates RAM to the virtual machine (4096MB in the example)

-*cdrom*: The path to the operation system ISO

-*boot*: This specifies the boot order for the virtual machine, *d* is the virtual CDROM

This example can be adapted to run the Linux distribution of your choice inside of a virtual machine.

Creating an advanced Debian Stretch virtual machine using debootstrap

To have more control over the installation of Debian inside of a virtual machine, you can use **debootstrap** to install the operating system. Another advantage of this approach is that you can avoid all of the installation dialogs of the **Debian installer**.

This section will show how to install Debian Stretch with the *Grsecurity* kernel from Subgraph OS.

Create a virtual hard-drive image for the operating system

To begin the install, you must set up a virtual hard-drive image. Follow these steps to set it up:

1. Run the following command to create a sparse virtual hard-drive image:

   ```
   $ truncate ——size 8G ./disk.img
   ```

2. To format the virtual hard-drive run the following command:

   ```
   $ /sbin/mkfs.ext4 ./disk.img
   ```

   After formatting the hard-drive, you can create a proper partition table. We will skip this step in the tutorial as it is not strictly necessary to run the virtual machine.

3. Mount the virtual hard-drive:

   ```
   $ sudo mount —o loop ./disk.img /mnt
   ```

NOTE: You should ensure there is enough free space in the image you create. You may want to allocate twice as much if you want to convert the image later on.

The sparse virtual hard-drive image you created will only use as much space as it requires.

Run the following command to show how much space is used by the image:

```
$ du —sh disk.img
```

The amount shown is a fraction of the total amount specified in the *truncate* command:

```
189M        disk.img
```

To verify the total amount that was specified in the *truncate* command, run this command:

```
$ du ——apparent—size —sh disk.img
```

The total amount should correspond with what was specified when you ran *truncate*:

```
8.0G        disk.img
```

Installing the operating system with deboostrap

Now that the virtual disk-image is created, we can now use **debootstrap** to install Debian Stretch. Follow these steps to install it:

1. Run **debootstrap** to install the operating system:

   ```
   $ sudo debootstrap ——variant=mintbase ——include=systemd—sysv stretch /mnt
   ```

2. Set a *root* password for the installed operating system:

   ```
   $ sudo chroot /mnt passwd
   ```

3. Create a standard fstab configuration:

   ```
   $ sudo tee /mnt/etc/fstab << EOL
   /dev/sda    /    ext4    defaults,errors=remount—ro  0   1
   EOL
   ```

Installing the Grsecurity kernel in the operating system

Run the following commands to install the Subgraph OS *Grsecurity* kernel in your virtual machine:

```
$ cd /tmp
$ apt—get download linux—{image,headers}—grsec—amd64—subgraph linux—{image,headers}—$(uname —r)
$ sudo cp ./linux—{image,headers}—$(uname —r) /mnt/tmp
$ sudo chroot /mnt
$ dpkg —i /tmp/linux—{image,headers}—*
$ update—initramfs —u —k all
$ exit
```

The kernel and initramfs are inside of your mounted virtual hard-drive image. You must copy them to a directory on your computer to boot the virtual machine using these files. Run the following command to copy the files to the directory you want to start the virtual machine from:

```
$ cp /mnt/boot/vmlinuz—<version>—amd64 /mnt/boot/initrd.img—<version>—amd64 /home/user/path/to/vm
```

Finalizing the installation of the operating system

As the final step, we will sync the filesystem and unmount the virtual hard-drive image:

```
$ sync
$ sudo umount /mnt
```

(Optional) If you prefer, you may convert the virtual hard-drive image to the *qcow2* format:

```
$ qemu—img convert —f raw —O qcow2 ./disk.img ./disk.qcow2
```

Starting the Debian Stretch virtual machine

Now you are ready to start the virtual machine. Run the following command to start it:

```
$ qemu—system—x86_64 —enable—kvm —hda ./disk.qcow2 \
    —kernel ./vmlinuz—<version>—amd64 \
    —initrd ./initrd.img—<version>—amd64 \
    —append root=/dev/sda
```

NOTE: This assumes you converted the virtual hard-drive image to the *qcow2*. If not, replace **disk.qcow2** with the correct name of your image.

Qemu/KVM options

This section uses some new options for **Qemu/KVM**.

-*kernel*: This is the operating system kernel to boot when starting a virtual machine

-*initrd*: This is the initramfs to boot when starting a virtual machine

-*append*: These are options to append to the kernel command line when starting a virtual machine

If you want to install grub to keep the kernel and initrd images inside the virtual machine you'll have to create a full partition table. You may also need to create a separate **/boot** partition. But this is out of scope for this tutorial.

Setting up simple networking in Qemu/KVM

By default, **Qemu** will transparently *NAT* your virtual machines to the host network. This can be disabled by using the **-net none** flag.

Alternatively, you can also open simple tunnels between the host and the virtual machine using the port redirection mechanism with the **-redir** flag:

```
—redir tcp:55700::55700
```

For more on networking in **Qemu/KVM** see:

- http://wiki.qemu.org/Documentation/Networking
- https://en.wikibooks.org/wiki/QEMU/Networking