PRACTICAL OPSEC

Table of Contents

OPSEC – Intro

Originally, the term comes from the USMIL and means „Operations Security“. From Wikipedia:

„[…] is a process that identifies critical information to determine if friendly actions can be observed by enemy intelligence, determines if information obtained by adversaries could be interpreted to be useful to them, and then executes selected measures that eliminate or reduce adversary exploitation of friendly critical information.

In a more general sense, OPSEC is the process of protecting individual pieces of data that could be grouped together to give the bigger picture (called aggregation). OPSEC is the protection of critical information deemed mission essential from military commanders, senior leaders, management or other decision-making bodies. The process results in the development of countermeasures, which include technical and non-technical measures such as the use of email encryption software, taking precautions against eavesdropping, paying close attention to a picture you have taken (such as items in the background), or not talking openly on social media sites about information on the unit, activity or organization's Critical Information List.“

So, perhaps that can mostly be summarized in security auditing, Data Loss Prevention (DLP) and the implementation

I‘d like to expand above term into **Operational Security** which basically means the same thing in a non-military context and roughly refers to a rather secure operation of any privately-owned information technology assets that are commonly available:

- **smartphone, tablet**
- **workstation, laptop**
- **server**

OPSEC – Bad Examples

Some bad realworld examples:

- Talking about work in public forums (often including specific logfiles and details)
- Show youtube-channel password @ TV5 :)
- Share personal or potentially sensitive data via dropbox, pastebin
- Use Google Mail, Facebook, WhatsApp
- Use sh*tty passwords
- Not being aware of updates (O/S and other S/W)
- Use of unencrypted protocols – Wall of Sheep @ Defcon

Ideally, you should ask yourself beforehand:

What kind of data gets shared, how and w/ whom? Which other ways could be possible?

Basically **you decide** what data you create, and how many and which ppl you share it with.

But on the other hand, you might not be fully aware of how much data is really collected by taking a simple picture w/ your smartphone when it comes to the Exif header:

- Type of device, phone or camera
- Exact time and date
- Potentially unique settings
- Software involved

„The Exif format has standard tags for location information. As of 2014 many cameras and mobile phones have a built-in GPS receiver that stores the location information in the Exif header when a picture is taken.“

To remove any exif data from images, you can use a small script using exiftool like this

```
!/bin/bash
exiftool -all= $1
```

OPSEC – Bad Examples contd.

Wireless Beacons:

- Wireless Networks, 802.11a/b/g/n
- Bluetooth Devices: Mobile, Headset, Speaker
- GSM :|

At the penultimate C4MP, running aircrack-ng revealed a lot of devices browsing for their home or office network access points, which at a second glance is a very good method to uniquely identify and follow persons and specific devices, offering at least data in the form of:

- Wireless MAC Address
- Device Name
- Commonly used home and office networks
- operating system and device infos

If the devices also offer services that might not be explicitely firewalled, all this potentially creates a prettey decent overall attack surface.

To leave your device in a more camouflaged state, you can do simple yet effective things:

- change your Wireless MAC Address (iwconfig hw ether, macchanger etc.)
- remove cached and/or preconfigured Network Names
- do not use wireless interfaces at all :)

More practically, it might be a better approach to

- do a config backup of your normal wireless configurations
- encrypt and wipe => eliminate all configs prior to visiting other places
- just bring explicit, event-only „burner“ devices (but what about your mobile phone?)

OPSEC – Transport Encryption

Transport Encryption

One of the most important elements of OPSEC is to use encrypted protocols to prevent very simple attacks which might occur in any public wifi or local area network.

For E-mail, if you use TLS encrypted IMAP and SMTP together w/ PGP/GPG, you decrease the chances of third-party-eavesdropping drastically.

For simple file transfers, ftp-tls, rsync or scp/sftp should be used, and when it comes to instant messaging, classically jabber-tls combined w/ Off-the-Record messaging (OTR) or the newer OMEMO and applications like threema, signal, telegram or wire on mobile devices are good choices.

Self-Hosting

Generally and most importantly, ask yourself which services can be run by yourself or by people you trust, if the skill-level is accurate:

- webserver, mail server
- telephone, fax, VOIP based communications
- jabber server
- storage and file sharing: nextcloud, privatebin, ...
- anonymization: VPN using OpenVPN or IPsec, tor, ...

Consider running most if not all required services via VPN as an additional layer of security, b/c in most cases, configuration and key material have to be identical at both ends of the communication channel and mean at least a higher effort for the attacker or eavesdropper, especially if you are controlling these assets and have implemented good network security (and) monitoring (NSM).

Question: Which insecure services do YOU or YOUR FRIENDS use?

OPSEC – Storage Encryption

Storage Encryption

Transport Encryption is a must, ideally strengthened by VPN communication channels. Full Disk Encryption (FDE) is another good way of securing your data on nowadays powerful and fast hardware, for example by using geli on BSD or luks on Linux. This covers not only the scenario of somebody trying to „legally“[1] access your device, but also does not automatically give away all your data in case of device loss or stealing.

Additionally, on a SSD, it is nearly impossible to really remove files completely, so having an encrypted FS can be seen as mandatory from the beginning on to overcome that problem. In theory, there are tools like scrub and manufacturer implemented (thus untrusted) special commands like ATA secure erase, but by the nature of the flash memory design, traces of original contents might still exist somewhere.

On a per-file-basis, if you use gpg or openssl, you can even use public and potentially insecure backup transfer and storage mechanisms.

Example: WL insurance file:

```
$ openssl enc -d -aes256 -in insurance.aes256 > cables.scv
```

The required password is also a good example for a secure one:

```
AcollectionOfDiplomaticHistorySince_1966_ToThe_PresentDay#
```

In an ideal world, the user understands the sensible nature of certain data and takes care of encrypting it beforehand[2]. Subsequently, the whole issue of encryption – which still means overhead as it introduces additional layers of complexity – can be taken out of the servers and providers responsibility AND everybody could store everything no matter where in a rather secure way.

[1] if the country you are travelling into is not allowed to force you to reveal any passwords
[2] if the password isn't '123') :)

OPSEC – Redundancy

Redundancy and High Availability (HA)

A single backup is more like the only chance to be able to restore data than a real guarantee. Only redundant backup(s), ideally also on different media and devices, are real backups. Additionally, backups should be restored from time to time to see if they are really working and no errors have been introduced.

This can easily be achieved by having two identical systems and using the backup to synchronize data in between these two regularly.

So, we see that redundant H/W also makes sense, b/c there is no strong dependency onto a single device if some system does not work for a while. Also, you can potentially revert changes that were not yet applied to all of your devices or media, which offers some sort of time machine or desaster control if you would like to put it that way.

From my experience, redundancy is very easy to achieve and even cheap by following some easy rules:

- **rather buy two or more smaller instead of one big new harddisk/microSD/USB**
- **rather buy two or more of the older and cheaper phone/laptop/desktop than the brand-new one**

On a professional level, High Availability (HA) in a server and firewall setup gives you more flexibility, especially for maintenance sessions, as you can rollout and test new things w/o having to touch the productive system(s) directly. Also, as previously mentioned, if one device fails, there is no immediate urge to panic and fix that right away. Especially for complex setups in the IaaS area, it might become handy to have two completely independent setups.

OPSEC – Integrity

File Integrity Checksums

To be able to notice (sensitive) file modifications, it is good practice to create and store its checksum:

```
$ sha256sum .ssh/authorized_keys
9d7af6ee5e2a703f710adae09b8050ecaeb16d8c15b0d2fce785e7e669da461c  .ssh/authorized_keys
```

It is crucial to store the checksum on multiple and/or remote media to prevent tampering, and a little skript we recently created might also become handy:

```
#!/bin/bash
#
# Create sha256 checksum for all files in directory
# and encode sha256 checksum of the checksumfile in QR
#
echo "[+] Creating checksums..."
find . -type f -exec sha256sum {} \; | tee SHA256.SUM
echo "[+] Creating QR..."
qrencode "`sha256sum SHA256.SUM`" -o SHA256_QR.png
exit 0
```

Other values of interest might be modification, access and creation times:

```
d1g@0x220a:~$ stat .ssh/authorized_keys
  Datei: .ssh/id_rsa.pub
  Größe: 391           Blöcke: 8          EA Block: 4096   Normale Datei
Gerät: fd01h/64769d    Inode: 7211648     Verknüpfungen: 1
Zugriff: (0644/-rw-r--r--)  Uid: ( 1000/     d1g)   Gid: (  129/     kvm)
Zugriff    : 2015-08-21 12:22:45.301465225 +0200
Modifiziert: 2015-08-08 22:57:01.608459894 +0200
Geändert   : 2015-08-21 12:15:44.804232224 +0200
 Geburt    : -
```

Which can also be used as `-ctime`, `-mtime` and `-atime` options for the `find` command as well.

OPSEC – Modularization

Modularization

It becomes very handy if you modularize your data into seperate elements that can be compressed, encrypted and redundantly archived, for example

- nextcloud storage folder
- thunderbird e-mail application data
- firefox/chrome/chromium browser application data
- local databases, e.g. for wiki content
- GnuPG keyrings, public keys, private keys
- system configuration backups
- virtual machine images of server VMs

and so on, b/c you are by then able to have fully-fledged identical system up and running in a very short timespan which might otherwise easily involve a few days if you start thinking about what you had on your old system and what could still be missing.

Ideally, if you have two or more identical and at the same time modular systems (and encrypted + redundant, locally dispersed backups of that) available, you can recover from nearly all accidentally issued commands as well by simply copying over lost contents right away.

From an operational perspective, it always makes sense to ask yourself the following question: W**hat is the effort of achieving an identical system running on a completely fresh H/W?**

It might be okay that your nextcloud storage and the e-mails can be recovered from your server, but it might not be okay that synchronizing that data might take hours or even days depending on what data plan you are able to use in a personal desaster scenario.

OPSEC – Basic Hardening

Basic Hardening Concepts

So, if we think about all the previous advices, you still haven‘t looked into the devices operating system security itself. Basically, using S/W that is built upon open source is the better choice, and personally, I would not like to use any device which I do not have full privileges on as well (so only rooted smartphones for example). Also, **regular updates are mandatory.**

Smartphone Hardening:

- Run LineageOS (formerly known as Cyanogenmod) instead of stock Android
- Activate Filesystem Encryption
- Use AFWall+ (netfilter based kernel firewall) to restrict network communications
- **Connect the device to your own, FOSS based AP to inspect ongoing communications**

Laptop and Desktop Hardening:

- Avoid M$ w1nbl0wz
- Use FOSS and POSIX based O/S like Linux or BSD
- Use stateful packet filtering
- Implement VPN tunnels to all important assets
- **Export and collect logfiles (ideally redundantly and via VPN)**

Server Hardening:

- **Chose a POSIX based O/S and additionally harden it, deploy in HA setup, NETWORK SEGMENTATION**
- **Carefully chose which services to run and inspect configuration(s)**
- **Implement a strong monitoring, a restrictive egress firewall ruleset and VPN(-only) access**
- **Implement anomaly detection, regular check and export / monitor system logfiles**
- **Conduct security audits (or even pentests) to verify and optimize your hardening efforts**