YAADA

Stijn Derks[1*], Nick van de Waterlaat[1†]

Abstract
YAADA, **Y**et **A**nother **A**RP poisoning - **D**NS spoofing **A**pplication, is a graphical user interface based application which has been developed for the lab assignment of the course 'Lab on Offensive Computer Security' (2IC80) taught at Eindhoven University of Technology. YAADA can perform fully-fledged ARP poisoning and DNS cache poisoning. YAADA can be released and used by people in different environments. The application considers a 'plug-and-play' fully automated implementation that adapts to several scenarios. Different IP addresses and a various number of hosts can be used as victims for the attacks implemented in YAADA. This article provides a technical and operative description of the attack mechanisms. Additionally, a description of the set up used to reproduce the attack, a breakdown of attack procedure, and a description of the produced code are provided.

[1] *Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands*
*__Corresponding author__: s.derks@student.tue.nl - 1008002
†__Corresponding author__: n.p.m.v.d.waterlaat@student.tue.nl - 1009357

Contents

Introduction

There exist applications that provide *Address Resolution Protocol* (ARP) poisoning and *Domain Name System* (DNS) spoofing capabilities. One of these applications is Ettercap [1]. In this paper, we propose **Y**et **A**nother **A**RP poisoning - **D**NS spoofing **A**pplication (YAADA), which aims to improve the current 'semi-automatic' approach a-là Ettercap. YAADA offers ARP poisoning and DNS cache poisoning with different modalities that persistently poisons ARP caches and uses DNS queries to poison the recursive DNS cache. Furthermore, YAADA takes different environments and functionalities into consideration in order to provide 'plug-and-play' capabilities in various environments. Moreover, the integration of the attacks is done in a 'seamless' and systematic manner. Additionally, the application incorporates the ability to scan the local network for potential victims, a way to save the traffic in a pcap file, and the ability to set a timer for when the attacker has to occur. The goal of the application is to reach as wide of an audience as possible. Therefore, the application incorporates a *Graphical User Interface* (GUI) that eases attack execution and victim selection.

1. Attack Description

The attacks considered in this paper consist of persistent ARP poisoning and DNS cache poisoning that uses ARP poisoning between, for example, an authoritative DNS server and DNS name server together with packet sniffing. Both of these attacks are elaborated on in Section 1.1 and 1.2 respectively.

1.1 ARP poisoning

ARP poisoning, also known as ARP cache poisoning or ARP spoofing, is an attack in which an attacker changes the *Media Access Control* (MAC) address of a victim or multiple victims to a manually chosen target MAC address in the ARP cache of a victim or a group of victims.

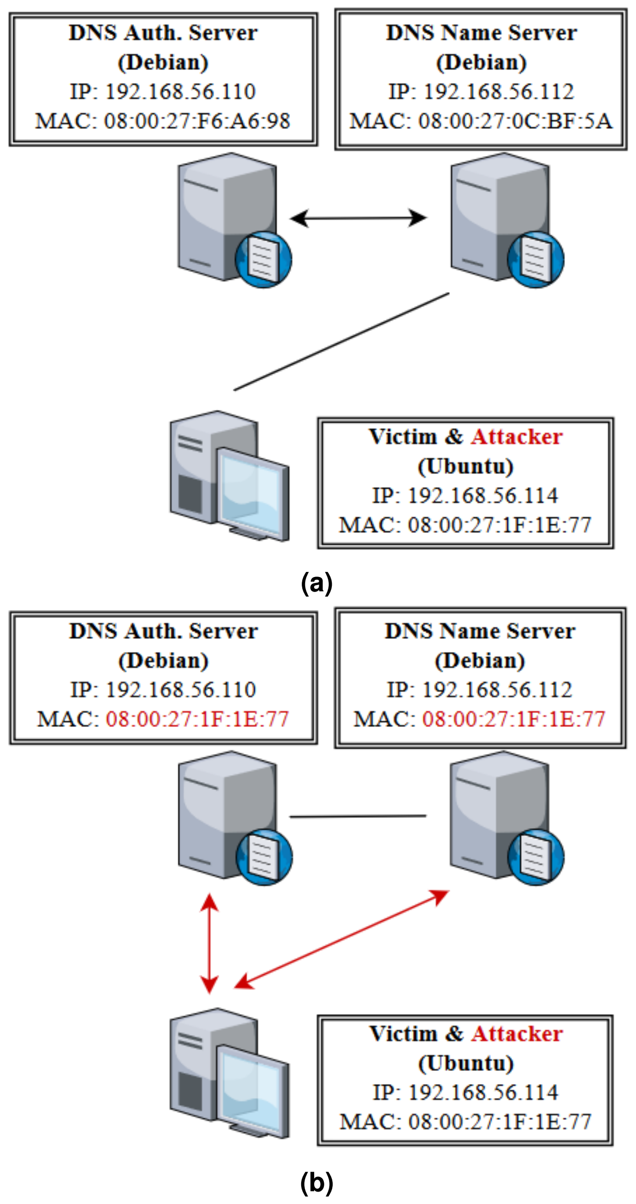


Figure 1. (a) The situation before a successful ARP poisoning attack. This is just regular communication between the authoritative DNS server and the DNS name server. (b) The situation after a successful ARP poisoning attack. Here, the attacker can intercept data sent between the authoritative DNS server and DNS name server. Note that the MAC addresses for the authoritative and name server are from an outsider's point of view.

As a result of this, the *Local Area Network* (LAN) is attacked as an entry in the victim's ARP cache is modified through forged ARP packets. Through this, the MAC address of a victim is changed to the target MAC address set by the attacker. YAADA implements this attack through two way ARP poisoning between each pair of two victims in the group of all victims. This means that between any pair of two victims, the entry of the other victim in the ARP cache is modified such that it points towards the target MAC address, which can be chosen by the attacker.

Let us take a closer look at the attack. Figure 1 depicts the before and after situation of an ARP poisoning attack executed with YAADA. In Figure 1a, we observe the direct communication between machines 'DNS Auth. Server' and 'DNS Name Server' in a 'normal' situation (before the attack). Now, let the 'DNS Auth. Server' and 'DNS Name Server' be the victims and 'Victim & Attacker' the attacker of an ARP poisoning attack. Assume that the authoritative and name server want to communicate with each other and suppose that there does not exist sufficient countermeasures against ARP poisoning attacks. Let us assume the attacker executes an ARP poisoning attack with YAADA on these servers with themselves as target. Then, the cached entries that consist of the MAC address of the authoritative server and the name server for the IP address of name server and authoritative server respectively will be modified to the MAC address of the attacker. This has been depicted in Figure 1b. To clarify, the attacker sent packets to both servers stating that the MAC address for the other server was the MAC address that the attacker has chosen. Now, if the authoritative and name server start to communicate over the local network, they send messages to the MAC address belonging to the IP address of the name and authoritative server respectively. Here, this IP address is bound to the MAC address of the attacker. As the transport layer is used for sending packets within networks, the MAC address of the attacker will be used for the packet routing and thus, for the communication between the authoritative and name server. Now, the attacker can monitor the communication between the authoritative and name server and forward all packets from the authoritative to the name server and vice versa such that neither of these machines notice missing packets. As a result, both the data and privacy of these servers are compromised.

In this example, two specific victims have been chosen, but no further details about them were given. It would be possible to have all users of a local network and the gateway as victims. Then, the attacker can spy on all the communication of the users of a local network with the internet. Additionally, the connection to the internet could be manipulated such that all devices lose access to the internet. Furthermore, an attacker could use ARP poisoning to redirect all traffic to another computer, which could cause a *Denial of Service* (DoS) on the targeted computer.

1.2 DNS cache poisoning

DNS spoofing is an attack which is used to supply false DNS information to victims such that when they attempt to visit a particular domain, they are sent to a fake domain residing at a different IP address than the original domain the victim wanted to visit. Once the DNS resolver receives the false information, it is stored in the DNS cache for the *Time To Live* (TTL). This means that the attacker has to provide false information at fixed time intervals. In order to exploit this flaw in the DNS resolver implementation, an attacker must be able to predict the *DNS transaction identifier* (TXID) and the UDP source port for the DNS query message correctly. Attackers can use this exploitation technique to redirect users from legitimate sites to malicious sites. Moreover, attackers can also inform the DNS resolver to use a malicious name server that is providing information used for malicious activities. Let us consider an example based on nodes in the network that were also considered in the previous example. In this example, M1 tries to visit the TUe website in Figure 2a. In this case, the visit will be successful. In Figure 2b, however, an attack (M3) has injected a fake DNS entry for the website M1 is trying to visit. Now, M1 is redirected to a fake website that is bound to the forged entry that M3 injected.

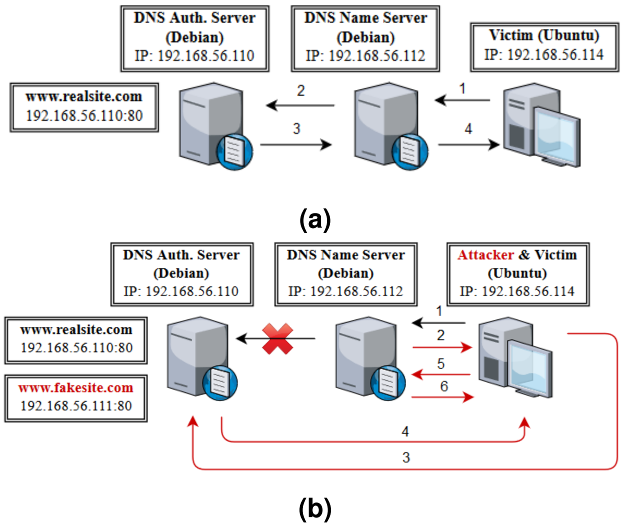

DNS Auth. Server (Debian) IP: 192.168.56.110
DNS Name Server (Debian) IP: 192.168.56.112
Victim (Ubuntu) IP: 192.168.56.114
www.realsite.com 192.168.56.110:80
1
2
3
4
(a)
DNS Auth. Server (Debian) IP: 192.168.56.110
DNS Name Server (Debian) IP: 192.168.56.112
Attacker & Victim (Ubuntu) IP: 192.168.56.114
www.realsite.com 192.168.56.110:80
www.fakesite.com 192.168.56.111:80
1
2
5
6
4
3
(b)

Figure 2. (a) The situation before a successful DNS cache poisoning attack. Victim visits www.realsite.com. (b) The situation after a successful DNS cache poisoning attack. Here, attacker injected a fake DNS entry for www.realsite.com that now instead goes to www.fakesite.com.

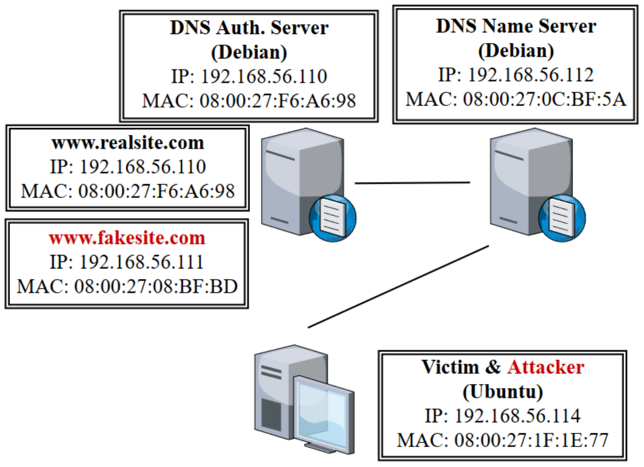

DNS Auth. Server (Debian) IP: 192.168.56.110 MAC: 08:00:27:F6:A6:98
DNS Name Server (Debian) IP: 192.168.56.112 MAC: 08:00:27:0C:BF:5A
www.realsite.com IP: 192.168.56.110 MAC: 08:00:27:F6:A6:98
www.fakesite.com IP: 192.168.56.111 MAC: 08:00:27:08:BF:BD
Victim & Attacker (Ubuntu) IP: 192.168.56.114 MAC: 08:00:27:1F:1E:77

Figure 3. The virtual machine setup consisting of a victim and an attacker, a DNS name server, and an authoritative DNS server. Two websites, namely www.realsite.com (192.168.56.110) and www.fakesite.com (192.168.56.111) are hosted on the authoritative server. This setup is used for ARP poisoning and DNS cache poisoning.

2. Technical Setup

In this section, various technical aspects of the set up that have been used to both develop and test the application will be elaborated on. In particular, the set up used for reproducing the attacks with the application are going to be touched upon. Additionally, other applications that have been employed to analyse the attacks and support the development will be referenced.

Environment Three virtual machines were used to reproduce the ARP poisoning attack and the DNS cache poisoning attack. The set up of these virtual machines is visualised in Figure 3. This set up is equal to the set up provided in the examples of ARP poisoning and DNS cache poisoning in Section 1.1 and 1.2 respectively. This set up has been used to both test and analyse the attacks. The first virtual machine, an Ubuntu installation, acts as both the attacker and the victim. The second virtual machine, a Debian server, acts as the DNS name server (NS) that sends queries from the victim and attacker to the authoritative server. Furthermore, it forwards responses from the authoritative server to the victim and attacker. The third virtual machine, also a Debian server, acts as the authoritative DNS server. This authoritative DNS server receives queries from the name server and responds to the queries from this name server such that the name server can forward these responses to the victim and attacker (the Ubuntu machine). Furthermore, the third machine also consists of two web sites that are hosted on the local network through Apache2. These web services represent a real website and a fake website that are to be used for the DNS cache poisoning attack. The machine makes use of Apache2 in order to host these websites at `www.RealSite.com` and `www.FakeSite.com` respectively. The DNS service makes use of Bind9 in order to provide the DNS capabilities. This service uses a fixed port, namely port 22222 and a transaction ID, which is represented as a 16-bit number.

The network scanning capabilities incorporated in YAADA ease the ARP poisoning and DNS cache poisoning as the end-user does not have to provide IP addresses and MAC addresses themselves. Thus, network scanning is an integral part of the application that eases the execution of the attacks. We do this as we do not want to bother end-users of the application to figure out who and what is connected to the local network. Furthermore, rather than asking the end-user for a network interface to use during the scan, all interfaces are scanned. Using all interfaces results in longer wait time for the end-user, but with this, we abstract this choice from the end-user who might not have any clue about network interfaces. This, in turn, reduces the complexity of the execution of attacks with YAADA. Inspiration for the local network scanning feature was obtained from Ettercap [1], a suite for *Man-In-The-Middle* (MITM) attacks. After network scanning, ARP poisoning, and DNS cache poisoning capabilities were correctly implemented and were working in different environments. The set up was extended with other functionalities for various testing purposes. One of these extra functionalities consists of saving the traffic during the ARP poisoning and DNS cache poisoning attack as a pcap file. This file can then later on be explored in Wireshark.

Testing For testing the major functionalities of the application, Wireshark [2] and unit tests have been used. Wireshark, specifically, was used in the testing phase in order to verify that the implemented attacks provided the right capabilities. The unit tests have been used for verifying essential functionalities that should be provided by the application. These can be found under the `/test/` folder. Note that these tests consider environment specific variables from the network it was tested on.

Graphical User Interface As coined previously, the application has to be easy to use for end-users. That is, the application should not only appeal to technically inclined people but also less technically inclined people in order to showcase how easy it is to execute forms of ARP poisoning and DNS cache poisoning attacks. This incentive goes hand-in-hand with the development of a GUI. For this reason, Python's de-facto standard GUI package 'Tkinter' has been used. Tkinter provides 'themed Tk' (ttk) functionality, which allows Tk widgets to look like the native desktop environment in which the application is running. This functionality promotes the understandability of the application in different desktop environments (e.g., Windows, Linux, Mac OS). The application uses the 'Scapy' module offered by Python for the implementation of the attacks.

3. Attack Analysis

3.1 Characteristics

ARP poisoning and DNS cache poisoning attacks as they are implemented in YAADA are not exactly the most reliable types of attacks. The success of the attack depends a lot on the security measures taken by the victim(s). When victims take appropriate countermeasures against these attacks, the attacks are less likely to be successful or as impactful as they can be at their full strength. Fortunately for victims, the attacks have certain limitations. ARP poisoning, for example, works only on local networks as this is where MAC addresses are actually meaningful. That is, for communication between different nodes in different networks, IP addresses are used. On the other hand, however, routers and local DNS servers have MAC addresses too. This means that these can also be victims of ARP poisoning attacks. Furthermore, the ARP poisoning attack only works because systems are not authenticated. We consider some of these countermeasures in Section 3.2. The attacks implemented in YAADA also put a requirement on the location of the authoritative server. That is, this server has to be present in the local network in order to perform an ARP poisoning attack on them. This can then be followed by sniffing the network and looking for communication about a certain domain between a name and authoritative server. Then, it is possible to perform a DNS cache poisoning attack. This does, however, not work when the authoritative server is not present in the local network. Something that is, however, possible is to perform the DNS cache poisoning attack not on the name and authoritative server, but on users in the network and a local DNS server. While this is not the DNS cache poisoning attack coined previously, we still achieve the same results to a large degree. That is, it is still possible to redirect users to a different website. A disadvantage, however, is that each victim has to be selected rather than simply considering all victims in a network, which is the case when the DNS cache of a local DNS server is poisoned.

3.2 Defense

Let us consider a few defense measures that can be taken against ARP poisoning and DNS cache poisoning respectively in the next paragraphs. We consider more general forms of defenses rather than defenses against the particular attacks implemented in YAADA as these measures would also work for the implemented attacks.

ARP poisoning There are a few measures that can be taken by end-users to defend themselves against ARP poisoning attacks or reduce the likelihood of them happening. The first measures consist of relying on *Virtual Private Networks* (VPNs) when one frequently makes use of public WiFi hotspots. If there is communication between two hosts on a regular basis, it is possible for both of these hosts to set up a static ARP entry of the other host. This static ARP entry will then act as a permanent entry in the ARP cache of both of these hosts, which adds a layer of protection against ARP poisoning attacks. If none of these additional measures is applicable, it is still possible to get third-party tools that are used for the detection of suspicious activity. These tools can help detect if an ARP poisoning attack is taking place. These kinds of tools can be further supported by using sound monitoring tools. These kinds of tools go hand-in-hand with a packet filtering set up, which can be used when to stop follow-ups of ARP poisoning attacks. Furthermore, it is possible for end-users to try to execute ARP poisoning attacks on themselves and see if they can defend themselves against those or at least detect those attacks. From the results of such a test, they learn if they need to take additional measures or if they are good to go (for the most part).

DNS cache poisoning Some of the measures, such as monitoring tools and packet filtering, mentioned in the previous paragraph also work against DNS cache poisoning attacks. Additionally, one should ensure that the DNS software, such as Bind [3], is at the latest version in order to have the latest security patches. Moreover, one should consider enabling auditing on all DNS servers. Furthermore, one could forbid recursive queries to prevent spoofing. Setting up multiple external and internal DNS servers is also a possibility for the more technically inclined users. One can also think about enabling the *Domain Name System Security Extensions* (DNSSEC), which adds a layer of security to regular DNS as the original design of DNS itself did not include any security measures. Bind9 considers a defense strategy that consists of UDP port randomisation, which strongly decreases the probability to guess the right field values when forging a fake authoritative DNS response. This is then an especially useful defense strategy against the Birthday Attack, which is now deprecated. Furthermore, the attack proposed in YAADA only works when the authoritative DNS server is located on the local network. This results in this attack not being able to be executed in any setting, but only in a particular setting with this requirement. Therefore, the attack proposed by YAADA is rather outdated and would not work in most modern environments.

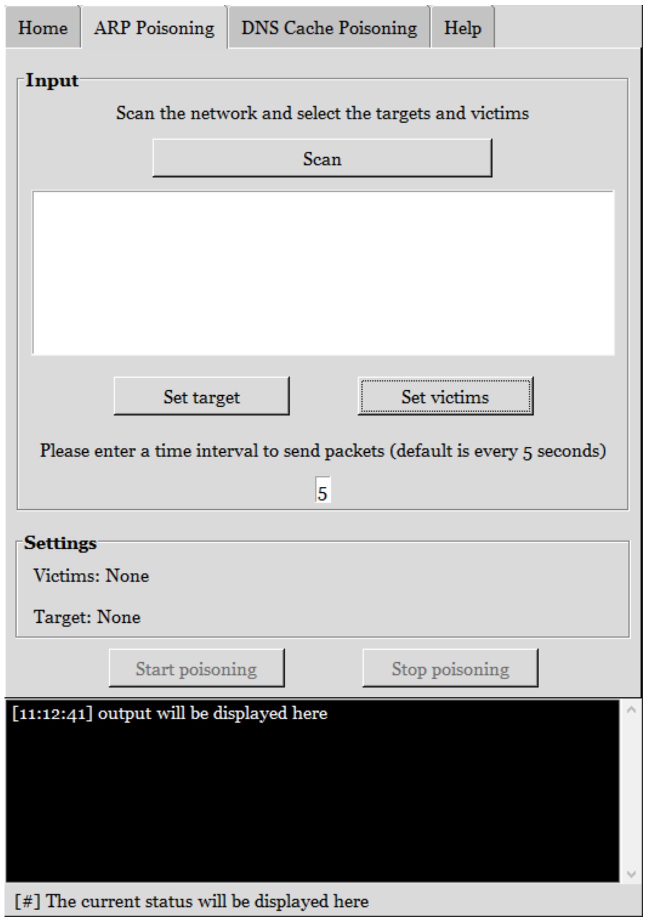
4. Attack Engineering

4.1 ARP poisoning

The application is easy to use, which stems from the linear flow of steps that have to be taken in order to execute attacks and its other capabilities. That is, the end-users cannot execute an attack before they set the victims and the target of the attack. Furthermore, they cannot stop an attack when no attack is active. Additionally, before an end-user can set a target and victims for ARP poisoning, they have to first scan the network as these options are not available otherwise. What this means is that the users have to perform a certain number of steps in order to execute the attacks, which makes the process rather linear and user-friendly. The tab that the user is presented with when wanting to execute an ARP poisoning attack has been depicted in Figure 4. In this figure, one can see the local network scan, the victims, target, the output logs, and the current status of the application. The ARP poisoning attack is done between each pair of victims and the MAC address of those victims for one another is set at the MAC address of the target. This is done on a separate thread where every x (default 10) seconds a forged packet is sent.

The chain of steps for executing an ARP poisoning attack after starting the application is are shown below. Note that we assume that all inputs are sound.

1. Navigate to the ‘ARP poisoning’ tab.
2. Scan the network by clicking on the ‘Scan’ button.



Scan the network and select the targets and victims

Please enter a time interval to send packets (default is every 5 seconds)

5

Victims: None

Target: None

```
[11:12:41] output will be displayed here
```

[#] The current status will be displayed here

Figure 4. The tab used for ARP poisoning attacks.

3. Select the victims (at least two) by clicking on items in the list.
4. Set the victims by clicking on the 'Set victims' button.
5. Repeat the previous two steps for the target (this can be done before the previous steps or between the previous steps as well).
6. Set a time interval that is used for sending forged packets or let it sit at a default of 10 seconds.
7. Start ARP poisoning by clicking on the 'Start Poisoning' button.

4.2 DNS cache poisoning

The DNS cache poisoning tab is disabled by default and gets enabled after executing an ARP poisoning attack when at least the DNS name server is in the group of victims and when the target is the attacker ('self'). This is done as an ARP poisoning attacks needs to be executed on the DNS name server and the authoritative server with the attacker's MAC address as the attacker can then sniff the traffic sent between the name server and the authoritative server. It is then possible to check for a certain domain to be queried, that is to be specified by the attacker, by sniffing the communication between the servers and modifying the location of the website to a fake IP. During the ARP poisoning attack, the attacker forwards all packets from the authoritative server to the name server if and only if the packets are not responses to the DNS query for this certain website. Then, the attacker can forge a response similarly to how the authoritative server would respond to the name server but with a different IP address for that certain domain. This way, the name server will cache the 'fake' website and return this for the specific domain to the victims.

The chain of steps for executing a DNS cache poisoning attack after starting the application are described below. Note that we assume that all inputs are sound.

1. Navigate to the 'ARP poisoning' tab.
2. Scan the network by clicking on the 'Scan' button.
3. Select the victims (at least two) by clicking on items in the list. Note that the 'DNS NS' victim has to be included in this selection of victims.
4. Set the victims by clicking on the 'Set victims' button.
5. Repeat the previous two steps for the target (this can be done before the previous steps or between the previous steps as well). Note that the target has to be the 'self' victim, which represents the attacker themselves.
6. Set a time interval that is used for sending forged packets or let it sit at a default of 10 seconds.
7. Start ARP poisoning by clicking on the 'Start Poisoning' button.
8. Navigate to the 'DNS cache poisoning' tab.
9. Fill in a domain that is accessible from the network.
10. Set the domain by clicking on the 'Set domain' button.
11. Fill in an IP that represents a fake domain that the victims should be redirected to.
12. Click on the 'Start poisoning' button.

5. Conclusion

YAADA offers its users an easy way to execute ARP poisoning and DNS cache poisoning attacks. There is, however, still much room for improvement. The primary improvements that can be made fall in two distinct categories, namely the functionalities offered by the application and the required knowledge to use the application effectively. Let us consider how these aspects can be improved.

Functionality As of now, the application only considers two attacks. This is not attractive for end-users as they are not looking for multiple tools for various attacks. Instead, they desire a single application that offers a dozen of various attacks. What this means when it comes to improvements for our application is that more attacks and other features could be added. Fortunately, the code of the application has been designed such that it is easy to extend the features provided by the application by adding new 'Frames' to the tab navigation bar. These frames can be found under the `/src/frames/` folder. Here, each frame considers a single attack. Furthermore, each of the frames consists of two parts; one part that considers the GUI and the other part that considers the functionality. The GUI part is implemented in the actual frame, whereas the attack is implemented under the `/src/attacks/` folder. This creates high perfective maintenance. Furthermore, the attacks implemented are already quite 'old' when we consider the world of information security. Therefore, these attack are not guaranteed to work on modern environments.

Required knowledge Even though a GUI has been used to make the application easy in use, there is still a barrier of technical knowledge the end-user should posses of in order to use YAADA comfortably. This minimum amount of knowledge is due to the entire application introducing technical terms, which can not be avoided at all times. Furthermore, in some cases, the end-user can set parameters themselves even though the default parameter would be perfectly fine. This complicates the process for less technically inclined users. Thus, it would be possible to implement a 'beginner' setting that provides a very minimalistic user interface where only the bare minimum has to be filled in by the end-user.

All things considered, the application provides the capabilities that it was intended to provide during the design of the application, but there is still much room for improvement. The ARP poisoning and DNS cache poisoning attacks can be powerful as they allow to affect a large number of users through single attacks. For the DNS cache poisoning attack, this is mainly due to many clients referring to the same DNS name server. There exist defensive measures that can be taken against these attacks. Some of these defenes do, however, rely on an attacker not having enough network capacity or computation power. This is something that could change in the future and that is why these attacks should still be kept in mind.

References

[1] Alberto Ornaghi and Marco Valleri. Ettercap, 2005.

[2] Gerald Combs et al. Wireshark-network protocol analyzer. *Version 0.99*, 5, 2008.

[3] Cricket Liu and Paul Albitz. *DNS and Bind.* " O'Reilly Media, Inc.", 2006.