**Minor Project Report**

**On**

***Detection of Advance Persistence Threat in Online Social Network using Honeypots***

**Submitted in partial fulfillment of the requirements**

**for the award of degree of**

**Bachelor of Technology**

**In**

**Computer Science and Engineering**

|  |  |  |
| --- | --- | --- |
| Guide: | Submitted By: |  |
| Ms. Leena Singh | Rishab Kumar | 00110407216 |
|  | Shreya Gupta | 10110402715 |
| (Assistant Prof.) | Yash Kaushik | 00210407216 |
|  | Vibhor Kumar | 03110402715 |



**Department of Computer Science and Engineering**

**Amity School of Engineering and Technology,**

**GGS Indraprastha University, New Delhi (2015)**

**CERTIFICATE**

This is to certify that the minor project report entitled “Detection of Advance Persistence Threat in Online Social Network using Honeypots”

submitted by

Rishab Kumar 00110407216

Shreya Gupta 10110402715

Yash Kaushik 00210407216

Vibhor Kumar 03110402715

has been accomplished under my guidance.

This report is submitted for partial fulfillment of award of degree of Bachelor of Technology in Computer Science and Technology at Amity School of Engineering and Technology, affiliated to Guru Gobind Singh Indraprastha University, Delhi.

Date:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ms. Leena Singh

Assistant Professor

Computer Science and Engineering

Amity School of Engineering and Technology

Bijwasan, New Delhi

**ACKNOWLEDGEMENT**

For the accomplishment of this project, I would like to take this opportunity to express gratitude to the people who have been a part of this project right from its inception and helped and supported me. It is to them I owe my deepest gratitude.

I express my sincere gratitude to **Prof. Dr. Rekha Agarwal (Director)**, Amity School of Engineering and Technology, and **Dr. Pinky Nair,Head of Department (Department of CS/IT),** for their constant encouragement and guidance. Also, I would like to thanks **Ms. Leena Singh (Assistant Professor, CSE.)**, for her constant support and guidance. Without their insights and support this project would not have been possible.

**ABSTRACT**

Reconnaissance is the initial and essential phase of a successful advanced persistent threat (APT). In many cases, attackers collect information from social media, such as professional social networks. This information is used to select members that can be exploited to penetrate the organization. Detecting such reconnaissance activity is extremely hard because it is performed outside the organization premises. In this paper, we propose a framework for management of social network honeypots to aid in detection of APTs at the reconnaissance phase. We discuss the challenges that such a framework faces, describe its main components, and present a case study based on the results of a field trial conducted with the cooperation of a large European organization. In the case study, we analyze the deployment process of the social network honeypots and their maintenance in real social networks. The honeypot profiles were successfully assimilated into the organizational social network and received suspicious friend requests and mail messages that revealed basic indications of a potential forthcoming attack. In addition, we explore the behavior of employees in professional social networks, and their resilience and vulnerability toward social network infiltration.

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| **TABLE NO.** | **CAPTION** | **PAGE** |
|  |  |  |
|  |  | **I** |
|  |  | **II** |
|  |  | **III** |
|  |  | **IV** |
|  |  | **V** |
|  |  | **VI** |
|  |  |  |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE NO.** | **CAPTION** | **PAGE** |
| **1.** | **The procedure the attack on Honeyfarm.** |  |
| **2.** | **Number of incidents captured during the period.** |  |
| **3.** |  |  |
| **4.** |  |  |
| **5.** |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**LIST OF SYMBOLS**

|  |  |  |
| --- | --- | --- |
| **SYMBOL NO.** | **CAPTION** | **PAGE** |
|  |  |  |
|  |  | **I** |
|  |  | **II** |
|  |  | **III** |
|  |  | **IV** |
|  |  | **V** |
|  |  | **VI** |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | **PAGE** |
|  |  |  |
|  | **CERTIFICATE** | **I** |
|  | **ACKNOWLEDGEMENT** | **II** |
|  | **ABSTRACT** | **III** |
|  | **LIST OF TABLES** | **IV** |
|  | **LIST OF FIGURES** | **V** |
|  | **LIST OF SYMBOLS** | **VI** |
|  |  |  |
|  |  |  |
| **1** | **Introduction** | **1** |
|  | **1.1 APT(Advanced Persistent Threat)** |  |
|  | **1.2 Lifecycle APT** |  |
|  | **1.3 APT Honeypots** |  |
|  | **1.4 Honeypot agent** |  |
|  | **1.5 The activity of attacks** |  |
|  | **1.6 Basic Types of Security Solutions** |  |
| **2** | **Existing System Disadvantage** |  |
|  | **2.1** **The communication and transport component.** |  |
|  | **2.2 All the various classical kinds of security attacks may be adapted and applied to online social networks.** |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**TABLE OF CONTENTS**

**Chapter - 1**

**Introduction**

Institutions and businesses always face new threats. One of the biggest problems lately is type of APT threats, which are sophisticated, multiple attacks at a specific organization. Threats type of APT (Advanced Persistent Threat) belongs to the category of cyber-attacks, their goals most often as commercial entities, political and state institution and the individuals. These types of threats require long-term high secrecy. They carried a group of attackers who are well privy to the problem. They use more types of vulnerabilities to break the key security systems. In the initial stage of the APT focus on getting information about the network configuration and server operating systems. Later, focus on installing rootkits and other malware to gain control and communication with C&C (Command & Control Server) attackers. The contested objects are long compromised to steal intellectual property, copying of confidential and sensitive data, or financial gain. Individual systems are often long infected, and the achievement of the objectives striker ever taken out of service.

**1.1 APT (Advanced Persistent Threat)**

Definitions of precisely what an APT is can vary, but can be summarized by their named requirements below:

**Advanced** - Operators behind the threat have a full spectrum of intelligence-gathering techniques at their disposal. These may include computer intrusion technologies and techniques, but also extend to conventional intelligence-gathering techniques. While individual components of the attack may not be classed as particularly "advanced" (e.g. malware components generated from commonly available do-it-yourself malware construction kits, or the use of easily procured exploit materials), their operators can typically access and develop more advanced tools as required. They often combine multiple targeting methods, tools, and techniques in order to reach and compromise their target and maintain access to it.

**Persistent -** Operators behind the threat have a full spectrum of intelligence-gathering techniques at their disposal. These may include computer intrusion technologies and techniques, but also extend to conventional intelligence-gathering techniques. While individual components of the attack may not be classed as particularly "advanced" (e.g. malware components generated from commonly available do-it-yourself malware construction kits, or the use of easily procured exploit materials), their operators can typically access and develop more advanced tools as required. They often combine multiple targeting methods, tools, and techniques in order to reach and compromise their target and maintain access to it.

**Threat -** APTs are a threat because they have both capability and intent. APT attacks are executed by coordinated human actions, rather than by mindless and automated pieces of code. The operators have a specific objective and are skilled, motivated, organized and well-funded.

**1.2 Lifecycle of APT**

APT has been firmly defined methodology that has been proven in recent years. It begins phishing and social engineering ends and export large volumes of stolen data to the attacker's server. Attackers use techniques and methods are constantly evolving and have a great ability to adapt effectively. They keep their tools a step ahead than the current status of infected systems. Attackers can have multiple campaigns running in parallel. Every consists of one or more operations. These operations are usually distributed into phases. For example, in the initial phase, the aim is to provide a striker initial entry point to the target system. The following phases are then usually parallelized and distributed among individual cells due to more efficient attacks. The subsequent section describes the basic operation phases within a single APT intrusion. The following section describes the details of these phases and their possible detection.

**Initial compromise -** This is done using conventional practices of social engineering, spear phishing emails, and with zero-day virus. Next option is to infections websites, and forced the victim to visit them. Operators behind the threat have a full spectrum of intelligence-gathering techniques at their disposal. These may include computer intrusion technologies and techniques, but also extend to conventional intelligence-gathering techniques. While individual components of the attack may not be classed as particularly "advanced" (e.g. malware components generated from commonly available do-it-yourself malware construction kits, or the use of easily procured exploit materials), their operators can typically access and develop more advanced tools as required. They often combine multiple targeting methods, tools, and techniques in order to reach and compromise their target and maintain access to it.

**Establish Foothold** **–** Install remote administration software in victim's network, create network backdoors and tunnels allowing stealth access to its infrastructure. Connection communication with the Command & Control server the attacker and as he controls remotely contested keeps updating machines and used malware.

**Escalate Privileges -** Use exploits and password cracking to acquire administrator privileges over victim's computer and possibly expand it to Windows domain administrator accounts.

**Internal Reconnaissance** **-** Collects information on surrounding infrastructure, trust relationships, Windows domain structure. Move Laterally — expand control to other workstations, servers and infrastructure elements and perform data harvesting on them.

**Maintain Presence -** Ensure continued control over access channels and credentials acquired in previous steps.

**Complete Mission -** Exfiltration stole data from victim's network.

Furthermore, in this article we will focus in detail on the stage Move laterally. Previous phase is detectable by standard quality tools. But if the attacker gets up to the current stage, it means that standard security techniques have failed. This phase is a standard security technique almost undetectable. The attacker behaves as a normal user and using common tools. One of the methods to detect the attacker is using the honeypots.

**1.3 APT Honeypots**

While there are many solutions to detect APT, are not all 100% effective. With the honeypot are able to some extent combat APT attackers. In this section we will discuss this problem and propose practical solutions that would form part of a system to detect APT. The concept of the honeynet first began in 1999 when Lance Spitzner, founder of the Honeynet Project, published the paper "To Build a Honeypot": "A honeynet is a network of high interaction honeypots that simulates a production network and configured such that all activity is monitored, recorded and in a degree, discreetly regulated."

Honeypot is an information system whose purpose is to attract potential attackers and record their activities. Honeypot is used to detect and analyse attacks on computer networks and systems. Honeypots servers are dedicated servers, workstations and the network collects information about attackers and intruders who attack systems. Honeypots are most often used for the early detection of malware and subsequent analysis of its behaviour. Malware is constantly changing its strategy of attack and different ways to hide and avoid finding. For these reasons, the malware somehow lure and then analyses their behaviour. It is important to remember that the honeypot does not replace traditional security systems, but only complements it. Based on design criteria, honeypots can be classified as pure honeypots, High interaction honeypots and Low-interaction honeypots.

Two or more honeypots on a network form a honeynet. Typically, a honeynet is used for monitoring a larger and/or more diverse network in which one honeypot may not be sufficient. Honeynets and honeypots are usually implemented as parts of larger network intrusion detection systems. A honeyfarm is a centralized collection of honeypots and analysis tools For detection system using APT with Highinteraction honeypots, Low-interaction honeypots and Honeypot on production systems.

**High-interaction honeypots** **-** Honeypot with a high degree of interaction shows a complete real system, with all services and functions. Unfortunately, this method of implementation allows the attack the whole system, including the honeypot

**Low-interaction honeypot -** These honeypots simulate only a few features transport layer operating system. In these systems, it is easy to identify the mapped threats, unfortunately detection of new types of attacks is impossible in most cases.

**Honeypot on production systems -** It is a special version of honeypots, implanted in a production system. If the user does not have access to production systems, allow him to produce the system log. After verification, but is not admitted to the productive version, but in the sandbox, with imaginary data. The attacker feels that operates within the contested system, but is found only in the sandbox, which is monitored. All information about the activities striker transferred to the control system. Depending on the system administrator if this will be a honeypot to inform the user. It can also serve as an opportunity to capture unauthorized access to authorized systems.

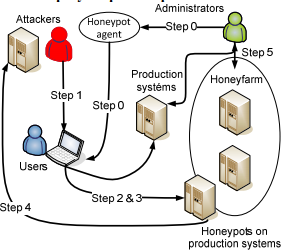
Monitoring APT attacks honeyfarm used with any number of High-interaction honeypots, Lowinteraction honeypots and Honeypots on production systems, according to the current situation.

**1.4 Honeypot agent**

Next complement the above solution is a honeypot agent. The original design of honeypots has one major limitation. Honeypots are waiting for the attacker. Role honeypot is passive. The design of this solution becomes the attacker honeypots notice and carries out its activity without being detected by the system.

Therefore, this solution we extended the agent who directs the attacker to the system honeypots. As these types of attacks simulate the behaviour of users, the attacker slip agendas and users little trap. The essence trap lies in the difference between continuous user behaviour and bot.

The user of the system is using the agent set a trap. The average user is hidden at first sight, or not interesting for his work. For example, a typical user ignores file system, various TMP directories, and the like. Bot trying to do the contrary, collecting information about invaded system, it searches every corner of systems. This is the stage where they come onto the scene Honeypot systems that offer interesting information for bots. The next chapter will present all the steps of how the system works.

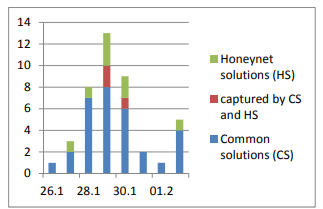


**Figure 1.1:** **The procedure the attack on Honeyfarm.**

**1.5 The activity of attacks**

The following chart recorded a number of anti-virus detection systems and antimalware a number of incidents captured by honeypots running in the selected time period for a non-homogeneous network. The environment consists of 400 systems under the control of the administrator, as well as about an average of 300 to 400 devices on private property without the possibility of influencing their management. Honeypot agent was installed about 15% of the stations.

Incidents, are marked as blue, captured by conventional anti-virus and anti-solutions. The green marked attacks are detected only by the honeypots. Red is marked by the intersection of the two types of detection. The attack was detected using the Common solutions and Honeynet solutions. More successful Common solutions is expected, an attack captured in the beginning. These attacks are mostly in documented and there is a defence for them. Unfortunately, some new types can bypass this protection, and then it can only be detected using the honeypots. These intersections are the most targeted, more destructive and more dangerous.



**Figure 1.2:** **Number of incidents captured during the period.**

**1.6 Basic Types of Security Solutions**

We present a basic division and subsequently introduced as a type of honeypot technology IDS.

The following discussion deals with the threat detection capabilities in a virtual environment. This includes a basic overview of the classification of intrusion detection systems, and discusses some of the basic concepts.

1. **Host and network-based systems**

Detection systems and intrusion prevention systems are divided into intrusion detection IDS (intrusion detection system) and intrusion prevention systems IPS (intrusion prevention system). It is also possible detection systems and intrusion prevention divided into host-based (host based IDS - HIDS) and network-based (NIDS). For both categories is common continuous monitoring system, the ability to alert the administrator to the attack revealed a record during the attack. HIDS systems are deployed on individual servers and user workstations. It is a software product, which suggests that the possibility of their use is limited support for operating systems used on the monitored computers. These products monitor system calls, logs, error messages, and the like. They protect against attacks on the operating system and applications running on the computer. They can evaluate the success of any attack. A comprehensive NIDS that use information obtained from the local network segment.

1. **Intrusion Detection System**

Intrusion Detection System IDS is used to detect intrusion attempts integrity, confidentiality and availability of data in the protected network. It is a set of tools, methods and resources that help us identify, disclose and report unauthorized and unapproved activities. It is a passive system which only draws attention to it and makes active countermeasures.

Through the warnings and statistics gives the operator information about the recorded attacks. It's just one part of the overall protection of the protection system.

It also detects operating activities, which do not necessarily represent a threat to the system. Some traditional IDS can also actively responding to the detected attack. In this case, mostly to work with a firewall that dynamically changing part of its policy to avoid the communication assessed as offensive.

**Chapter - 2**

**Existing System Disadvantage**

Advanced attackers make use of online social networks (OSNs) in order to extract useful information and establish contact with company employees as potential entry points into the organization.

Nowadays, online social networks involve people from the entire world, of any age and with any kind of education. They also helped to increase computer usage among categories that previously showed little interest for it (Stroud, 2008). The users of information systems have various types of security requirements, including: confidentiality, integrity, accountability, availability and anonymity. The same security requirements can be applied to social networking platforms, as well. Unfortunately, while most users are aware that their profile and the information they publish is essentially public, they usually strengthen their privacy settings only after problems arise and tend to overlook the actual impact of the information they disclose (Stroud, 2008). Apparently harmless information can be exploited, and the more information the attacker has, the more severe and sophisticated the attack can be. For example, name, location and age can be used to connect a profile to a real-world identity for more than half of the residents in the USA (Irani et al., 2011). In fact, social networking platforms are susceptible to different types of attacks, targeting different components, conducted from different domains, using different techniques. For better analysing these attacks, it is useful to identify the main abstract components of a generic social networking platform, corresponding to different functional aspects of those systems. Attackers can target each of the different components, or they can target different levels, possibly with roughly the same logic.

We identify four main components:

1. The social networking component. It manages and protects access to the users' personal profiles and the social relationships among users.

2. The content management component. It manages and protects access to all user generated content, including personal status updates, comments, links to other content, photos and multimedia galleries.

3. The infrastructure services component. It provides the basic infrastructure services needed to run the social networking platform, including storage and replication services for content and profiles, information indexing and routing, management of users' online presence.

**2.1** **The communication and transport component.**

It encapsulates basic inter-networking and adhoc networking functionalities. Moreover, we can distinguish two different kinds of attackers:

* **Intruders** - An attack can be conducted by users accessing the system without proper authorization, or with accounts created for conducting the attack, purposely.
* **Insiders** - Also legitimate users or entities participating in the systems operations can assume malicious behaviours. From the users point of view, malicious behaviour can also be attributed to the service provider.

**2.2 All the various classical kinds of security attacks may be adapted and applied to online social networks.**

The most typical threats against OSNs include:

1. **Unauthorized Access -** Users who have not been granted adequate permissions for accessing some services and resources, may attempt to circumvent the security mechanisms and policies of the system and gain unauthorised access. In a social networking platform, any user who has access to some profiles and messages can harm their legitimate owners. The collection of existing data is the basis of profiling attacks. These data may also supply some knowledge for secondary data collection from a wide range of different sources, including other OSNs. Remote access can also occur at system level. In this case the attacker may directly gain control of all resources.
2. **Social engineering -** In a social networking application, a common attack is to psychologically manipulate a user into performing misguided actions. It is similar to a confidence trick or a traditional fraud, but by means of computer-based communications and online social networking, typically to gain access to confidential information. In a phishing scheme, the attacker masquerades as a trustworthy entity to obtain the desired information. In most cases the victim and the attacker never acknowledged each other directly in real life.
3. **Masquerading -** When a rogue user disguises his identity and claims the identity of another user, the former is said to be masquerading. Masquerading may be attempted by an attacker either during a conversation or while registering his own profile, for deceiving other users or the whole social networking platform. Sometimes, masquerading is the first step to gain access to infrastructure services and resources to which the attacker is not entitled. Simple impersonation, by cloning the victim's profile from the same platform or by porting profile data from a different platform, may easily lead the attacker to gain trust from the victim's contacts. This way, it can damage other users eventually deceived. Especially in communities where reputation is valued, masquerading can also damage the user whose identity has been stolen. In fact, the attacker may pretend to be another user in order to shift the blame for any liable action.

**2.3 We will provide some examples and a discussion of these threats in the following subsections.**

However, also other traditional security attacks can be conducted against OSNs, including:

Denial of Service (DoS). The services and communications at the infrastructure level can be disrupted by common denial of service attacks. Social networking platforms are also susceptible to all the conventional denial of service attacks aimed at the underlying operating system or communication protocols. In addition to attacking the whole infrastructure of a social networking platform, users can also launch denial of service attacks against specific users, especially in a distributed platform. For example, repeatedly sending messages or other spam may place undue burden on the recipient users and their systems. Malicious users can also intentionally distribute false or useless information to prevent other users from completing their social activities.

• **Repudiation** - In general, repudiation occurs when a user, after having performed some action, later denies that action having happened (at least under his responsibility). Repudiation can be intentional or even accidental. It can also be the result of a misunderstanding, when users have a different view of events. In any case it can generate important disputes. In a sense, nothing can prevent a user from repudiating one of his actions. But a social networking platform can eventually help resolving disputes by providing needed evidence, if it maintains a sufficiently detailed log of events. For users who value their reputation, the availability of such evidence may constitute a valid deterrent. • Eavesdropping. The attempt to observe the flow and possibly the content of confidential messages is one of the most classical security threats. Apart from reading the content of messages, which may require cryptanalysis, an eavesdropper may gather useful information by simply observing the pattern of messages and their recipients, for example inferring the type of services being requested. To eavesdrop on other users, an attacker may also exploit the infrastructure and communication services of the platform, e.g. through unauthorized access.

• **Alteration -** When a user signs up a social networking service, he starts exposing his profile and content to the platform. An attacker may tamper with the profile and content data published by the victim, with all the messages he communicates to other users and all data used on the infrastructure services. Alteration can also be conducted by the service operator, which provides the facilities for online social networking and may take control of published data. Alteration may take the particular form of filtering, or censorship, when applied systematically for removing undesired content from the OSN.

• **Copy and Replay** - Each action in a social network may be subject to copy and reply. In this type of security threat, an attacker attempts to intercept some data and clone it, for retransmitting it later. The interceptor may successfully copy and replay a message, a complete profile or any other data. If those data are not associated with a signature and a timestamp, the repeated reception of such copies may pass unnoticed and accepted as a legitimate action.

Attackers are drawn to these channels because they make finding and engaging targets trivial, are easy and cost effective to use, are simple to create fraudulent accounts and allow the spread of malicious content at an unprecedented scale and efficiency.

**From the recent [Vevo breach](https://www.zerofox.com/blog/vevo-hacked-via-linkedin-phishing-campaign/" \t "_self) stemming from a LinkedIn phishing attack to**[**Russian operatives**](https://www.zerofox.com/blog/russia-just-used-trumps-favorite-social-network-hack-us-government/)**using Twitter to spearphish and distribute malware to the United States Department of Defense, advanced, large-scale cybercrime on social media has become mainstream. In light of National Cybersecurity Awareness Month.**

**2.4 The ZeroFOX team compiled a list of the ten worst social media attacks of all time to demonstrate the growing need for safeguarding these platforms. In no specific order:**

[**10k US Government Employees Spearphished with Malware-Laced Posts**](http://time.com/4783932/inside-russia-social-media-war-america/)

 In early 2017, Russian operatives sent over 10,000 custom phishing messages via social media, each link laced with malware enabling the attacker to access and control the victim’s device. This attack represents a major advancement in cyber capabilities and an escalation in Russia’s cyberwar against the US. This is the most well-organized, coordinated attack at the nation-state level we’ve ever seen.

[**Fake Social Media Persona Sends Malware to Employees Via Social Media**](https://www.secureworks.com/research/the-curious-case-of-mia-ash)

Attackers created an incredibly compelling fake persona, a London-based photographer named Mia Ash, and connected with corporate employees. The attacker disseminated a Remote Access Trojan (RAT), called PupyRAT, via these social media honeypot accounts to hijack the controls of victims’ devices. The persona had accounts across several popular social networks.

[**3rd Party App Leads to Hundreds of High-Profile Account Compromises**](https://techcrunch.com/2017/03/15/twitter-counter-hacked/)

 A vulnerability in a 3rd-party app called TwitterCounter allowed Turkish-language attackers to hijack controls of hundreds of high-profile accounts. They posted aggressive messages against the Netherlands after a contentious week of deteriorating relations between the Netherlands and Turkey and pivotal elections in both countries. The posts used swastikas and called the Dutch “nazis.” The breached accounts included a number of global brands and well-followed, verified accounts, including Forbes, the official Bitcoin Blockchain account, Starbucks, the European Parliament, UNICEF, Nike and Amnesty International.

[**HAMMERTOSS Malware Uses Social Media for Command & Control**](https://www2.fireeye.com/rs/848-DID-242/images/rpt-apt29-hammertoss.pdf)

The HAMMERTOSS malware automatically searches social networks for commands posted by attacker profiles, allowing cybercriminals to control the malware via social media posts. The attacker group behind this malware is also responsible for attacks against the White House, the Joint Chiefs of Staff, the State Department and other nation-state governments, such as Norway. This novel approach to weaponizing social media shows the need to analyze social media as a full lifecycle attack vector.

[**Financial Crime Runs Rampant on Social Networks**](https://techcrunch.com/2016/08/26/tracking-instagrams-money-flipping-scammers/)

ZeroFOX researchers revealed the vast underground world of financial crime on social media, in which scammers prey on the followers of verified banks with fraudulent financial services offerings, including card cracking and money flipping. The scale of the problem is massive, with nearly a quarter-million posts for a single type of scam on a single social network. The problem was found on every major social media channel and results in hundreds of millions of dollars in losses annually.

[**AP’s Social Accounts Hijacked, $136 Billion Lost in Stock Market Value**](https://www.washingtonpost.com/news/worldviews/wp/2013/04/23/syrian-hackers-claim-ap-hack-that-tipped-stock-market-by-136-billion-is-it-terrorism/)

Attackers compromised the account of the Associated Press, posting fake breaking news that bombs had gone off in the White House. The Dow subsequently dropped 150 points before rebounding; an economic value of $136 billion. Attackers have done this on a less public scale, in which they subtly erode the value of a company by compromising accounts and posting slanderous news about their target.

[**LinkedIn Hacked, Exposing 117 Million Credentials**](https://motherboard.vice.com/en_us/article/78kk4z/another-day-another-hack-117-million-linkedin-emails-and-password)

 The networks themselves get breached as well. The 2016 LinkedIn data dump was the 7th largest in history by sheer number of compromised credentials, according to HaveIBeenPwned.com. The breach, which originally occurred in 2012, resulted in an eventually 117 million exposed email and password combinations, which were then sold on the dark web.

[**Enigma’s Slack and website hacked, a half million in Ether coin stolen**](https://techcrunch.com/2017/08/21/hack-enigma-500000-ico/)

 Social collaboration tools are an often overlooked genre of social platforms that pose a new security risk. In 2017, the Slack community channel of Enigma, a startup exchange for the cryptocurrency Ethereum, were breached by attackers. The attackers impersonated the executives of the company and instructed the community members to send their Ethereum coin to a specific coin wallet, stealing roughly a half million worth of the cryptocurrency.

[**Phishing Direct Message Sent to Customers from Compromised Brand Account**](http://news.softpedia.com/news/Bank-of-Melbourne-Twitter-Account-Hacked-222511.shtml)

 In September of 2011, an Australian bank suffered the worst-case scenario for an account takeover, in which attackers didn’t immediately vandalize the account or post inflammatory messages, but instead sent direct messages to followers asking them to disclose sensitive financial institutions. While most account hacks are merely embarrassing and costly from a brand and public relations perspective, they can also be used for large scale cyberattack against a brand’s most loyal and engaged followers.

[**Vevo Hacked Via Targeted LinkedIn Phishing Attack, 3.12TB Exfiltrated**](http://gizmodo.com/welp-vevo-just-got-hacked-1813390834)

**Streaming service Vevo suffered a breach when one of its employees was phished via LinkedIn. Hackers were able to obtain and publicly release 3.12TB worth of the company’s sensitive internal data. The professional social network allows attackers to rapidly identify their target at a specific organization and send them a personalized message, all under the auspices of professional networking or recruitment.**

To help people understand just how hackers get your passwords – secure or otherwise – we've put together a list of the top ten password-cracking techniques used by hackers. Some of the below methods are certainly outdated, but that doesn't mean they aren't still being used.

## The top ten password-cracking techniques used by hackers:

### 1. Dictionary attack

The dictionary attack uses a simple file containing words that can be found in a dictionary, hence its rather straightforward name. In other words, this attack uses exactly the kind of words that many people use as their password.

Cleverly grouping words together such as "letmein" or "superadministratorguy" will not prevent your password from being cracked this way – well, not for more than a few extra seconds.

### 2. Brute force attack

Similar to the dictionary attack, the brute force attack comes with an added bonus for the hacker. Instead of simply using words, a brute force attack lets them detect non-dictionary words by working through all possible alpha-numeric combinations from aaa1 to zzz10.

It’s not quick, provided your password is over a handful of characters long, but it will uncover your password eventually. Brute force attacks can be shortened by throwing additional computing horsepower, in terms of both processing power – including harnessing the power of your video card GPU – and machine numbers, such as using distributed computing models like [online bitcoin miners](http://www.alphr.com/security/1008460/government-websites-bitcoin-cryptocurrency-mining).

### 3. Rainbow table attack

Rainbow tables aren't as colourful as their name may imply but, for a hacker, your password could well be at the end of it. In the most straightforward way possible, you can boil a rainbow table down into a list of pre-computed hashes – the numerical value used when encrypting a password. This table contains hashes of all possible password combinations for any given hashing algorithm. Rainbow tables are attractive as it reduces the time needed to crack a password hash to simply just looking something up in a list.

However, rainbow tables are huge, unwieldy things. They require serious computing power to run and a table becomes useless if the hash it's trying to find has been "salted" by the addition of random characters to its password ahead of hashing the algorithm.

There is talk of salted rainbow tables existing, but these would be so large as to be difficult to use in practice. They would likely only work with a predefined "random character" set and password strings below 12 characters as the size of the table would be prohibitive to even state-level hackers otherwise.

### 4. Phishing

There's an easy way to hack: ask the user for his or her password. A phishing email leads the unsuspecting reader to a faked log in page associated with whatever service it is the hacker wants to access, requesting the user to put right some terrible problem with their security. That page then skims their password and the hacker can go use it for their own purpose.

Why bother going to the trouble of cracking the password when the user will happily give it you anyway?

### 5. Social engineering

Social engineering takes the whole "ask the user" concept outside of the inbox that phishing tends to stick with and into the real world.

A favourite of the social engineer is to call an office posing as an IT security tech guy and simply ask for the network access password. You’d be amazed at how often this works. Some even have the necessary gonads to don a suit and name badge before walking into a business to ask the receptionist the same question face to face.

### 6. Malware

A keylogger, or screen scraper, can be installed by malware which records everything you type or takes screenshots during a login process, and then forwards a copy of this file to hacker central.

Some malware will look for the existence of a web browser client password file and copy this which, unless properly encrypted, will contain easily accessible saved passwords from the user's browsing history.

### 7. Offline cracking

It’s easy to imagine that passwords are safe when the systems they protect lock out users after three or four wrong guesses, blocking automated guessing applications. Well, that would be true if it were not for the fact that most password hacking takes place offline, using a set of hashes in a password file that has been ‘obtained’ from a compromised system.

Often the target in question has been compromised via a hack on a third party, which then provides access to the system servers and those all-important user password hash files. The password cracker can then take as long as they need to try and crack the code without alerting the target system or individual user.

### 8. Shoulder surfing

The most confident of hackers will take the guise of a parcel courier, aircon service technician or anything else that gets them access to an office building.

Once they are in, the service personnel "uniform" provides a kind of free pass to wander around unhindered, and make note of passwords being entered by genuine members of staff. It also provides an excellent opportunity to eyeball all those post-it notes stuck to the front of LCD screens with logins scribbled upon them.

### 9. Spidering

Savvy hackers have realised that many corporate passwords are made up of words that are connected to the business itself. Studying corporate literature, website sales material and even the websites of competitors and listed customers can provide the ammunition to build a custom word list to use in a brute force attack.

Really savvy hackers have automated the process and let a spidering application, similar to those employed by leading search engines to identify keywords, collect and collate the lists for them.

### 10. Guess

The password crackers best friend, of course, is the predictability of the user. Unless a truly random password has been created using software dedicated to the task, a user-generated ‘random’ password is unlikely to be anything of the sort.

Instead, thanks to our brains' emotional attachment to things we like, the chances are those random passwords are based upon our interests, hobbies, pets, family and so on. In fact, passwords tend to be based on all the things we like to chat about on social networks and even include in our profiles. Password crackers are very likely to look at this information and make a few - often correct - educated guesses when attempting to crack a consumer-level password without resorting to dictionary or brute force attacks.

**Chapter – 3**

**Proposed System Objective**

The objective of this case study was to analyze the suggested method of using artiﬁcial proﬁles as honeypots. In addition, we derive useful insight which relate to the operation and deployment of such social network honeypots. Speciﬁcally, we attempt to answer the following questions:

1) How can we create a genuine and attractive honeypot?

2) What should be the honeypot wiring strategy?

3) How often are proﬁles subject to attacks (suspicious emails or suspicious friend requests)?

4) How can malicious contact attempts be identiﬁed?

5) How easily do employees trust and connect with unknown people?

6) Can the proposed framework be executed and operated on real OSNs?

# **Honeypots are primarily deployed for exactly the reasons you describe -- to distract would-be attackers from the real servers. Using a honeypot may give you the opportunity to detect and respond to an attack (on the bogus system) before the attackers are able to do any real harm. Whether or not to deploy a [honeypot](https://searchsecurity.techtarget.com/definition/honey-pot)is something you should consider carefully.**

# **In order to be effective, a honeypot must appear real enough to attackers to attract their interest, but must (of course) not contain any data or information of real value. This can be a difficult balance to obtain. And the honeypot itself must be specially secured, so that it could not be used to compromise other hosts on your network.**

# **You should consult with your legal department to determine whether there are any legal issues within your organization, state, or country, relating to the use of a honeypot. This may include your own liability if your honeypot is compromised and used to attack others, as well as what you can and can't do. For example, you may be able to passively monitor and record all activity, but may NOT be able to take any direct or retaliatory action against your attackers.**

# **You would also need to consider what is and is not acceptable as evidence, should you ever intend to prosecute any attackers. Given all of the above, you should consider whether your time**and security efforts are better spent configuring, monitoring, and maintaining a honeypot, or configuring and deploying additional security and monitoring on your "real" servers. Security is always an equation that balances cost, risk, and business need; the answer for each organization is different.

# We propose social network honeypot for acquiring indications of forthcoming attacks. The general concept is presented. The artiﬁcial proﬁles are created, integrated into the OSN, and monitored. An attacker operates several OSN proﬁles to search for relevant employees and connect with them. During this process, the attackerattempts to contactthe honeypots, for example, by sending a friend request or an email message with a malicious payload. Suspicious friend requests and emails sent to the honeypot’s email account are analyzed. The goal of the social network honeypot is to trap the attacker’s activity as soon as possible.

# Using the proposed framework, we strive to provide the following beneﬁts to the organization.

# 1) Understanding the extent to which the organization is a target of attacks via OSNs.

# 2) Understanding which functions in the organization that attackers are interested in (e.g., secretaries versus senior technical personnel).

# 3) Detecting APT attacks at early phases in the APT life cycle.

# 4) Providing detection with a minimal false positive rate.

# 5) Understanding to what extent attackers use the email addresses of employees or the OSNs as an entry point to the organization (e.g., for injecting malicious code).

# 6) Providingthe organizationwith additionalforensicinformation.

# 3.1 The proposed framework, supports the creation, maintenance, and monitoring of artiﬁcial proﬁles (i.e., honeypots) in OSNs.

# We elaborate on its main components in the following sections.

# Social Network Acquisition

# The ﬁrst component focuses on acquisition of the informal social network of organization employees. It includes a crawler whose objective is extracting user information from proﬁles of members of the target organization from various OSNs. Such information is utilized by the system for creating reliable artiﬁcial proﬁles. Two main methods can be used for crawling: using (developer) API and Web Scrapping. Acquiring OSN proﬁles is a challenging task. Social network services detect and block unsolicited crawling activities and ofﬁcial data acquisition channels are not well established yet. Other technical challenges include varied API and page structures of the different OSNs where employees may have their proﬁles. Additionally, it is important to normalize the data and mark the missing pieces appropriately. Finally, identiﬁcation of the employee proﬁles in the various OSNs is a nontrivial task, especially when the employees are kept uninformed regarding the honeypot deployment in order to minimize the threat of insider data leakage. In general, the data collection required to create genuine honeypots is very similar to the reconnaissance activity performed by attackers. For example, both parties can employ targeted crawling or homing social bots to acquire the data. A third approach, suitable for very few organizations,is to oblige the employees to expose their personal OSN proﬁles to some organizational web application. In current implementation, we employ targeted crawling.

# Proﬁle Manager

# This is the main component of the framework which controls the proﬁle after its creation and supports: accepting/sending friend requests, sending posts and messages, completing the “like” action, and more, depending on the API provided by the speciﬁc OSN. Its primary task is “wiring” the honeypot (i.e., connecting it with other proﬁles in the OSN in order to increase its reliability). The framework provides a method for identifying proﬁles that should be approached with a friend request. This is performed based on the “social-bot organization intrusion” strategy , which is based on the following assumptions:

# 1) The more friends a user has, the more likely he or she is to approve the friend request.

# 2) The more mutual friends a user has with the requester the more likely he or she will approvethe friend request.

# The wiring algorithm includes these main phases:

# Phase 1 (Connect to Collaborating Employees): Typically, a group of employees is aware of the honeypot deployment process in order to support it from the IT, Human Resources (HR), and security perspectives. Connecting thehoneypots to a subset of these employees, with their consent, will increase the proﬁle credibility and help with further assimilation in the OSNs.

# Phase 2 (Send Requests to External, Highly Connected, Proﬁles): This phase helps to further increase the credibility of the honeypot by connecting to proﬁles with a high probability of approving friend requests.

# Phase 3 ( Send requests to insider proﬁles): Phase 3 consists of sending friend requests to the employees having the highest probability of accepting the friend requests according to the number of friends a proﬁle has and the number of mutual friends.

# Each friendship request proposed by the algorithm should be approvedby the system operatorbeforesending the request.

**Chapter - 4**

**Design and Implementation**

The system has been designed to detect various attacks on social media like Tweeter, Facebook etc. and to also secure the user’s account by analyzing their authentication record.

Interaction with the various OSN(Online Social Network) has been done by using their API’s.

Then any user’s account has been logged in and analyze their account by analyzing their tweets, counting of no. of friends to know that how many people are there in his network.

In the implementation of security part of the project, the security has been done by trying to login into user’s friend’s account and by passing the combination of very general password that most of the people keep which can be easily detected by hacker.

By passing various general combination of password if one is able to login into other’s account means that the person whose account has been logged in has kept a very weak password.

**Tools used**

**4.1 Tweepy**

Tweepy supports accessing Twitter via Basic Authentication and the newer method, OAuth. Twitter has stopped accepting Basic Authentication so OAuth is now the only way to use the Twitter API.

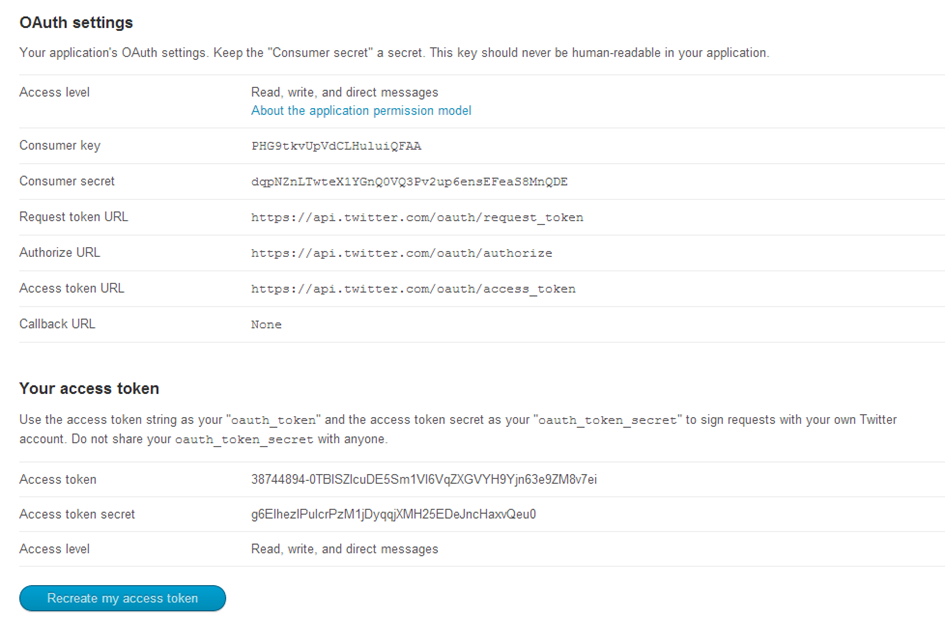
Tweepy is a great open-source library which provides access to the Twitter API for Python. Although the documentation for Tweepy is a bit scarce and doesn't have many examples, the fact that it heavily relies on the Twitter API, which has excellent documentation, makes it probably the best Twitter library for Python, especially when considering the Streaming API support, which is where Tweepy excels.

The main difference between Basic and OAuth authentication are the consumer and access keys. With Basic Authentication, it was possible to provide a username and password and access the API, but since 2010 when the Twitter started requiring OAuth, the process is a bit more complicated. An app has to be created at [dev.twitter.com](https://dev.twitter.com/).

OAuth is a bit more complicated initially than Basic Auth, since it requires more effort, but the benefits it offers are very lucrative:

* Tweets can be customized to have a string which identifies the app which was used.
* It doesn’t reveal user password, making it more secure.
* It's easier to manage the permissions, for example a set of tokens and keys can be generated that only allows reading from the timelines, so in case someone obtains those credentials, he/she won’t be able to write or send direct messages, minimizing the risk.
* The application doesn't reply on a password, so even if the user changes it, the application will still work.

After logging in to the portal, and going to "Applications", a new application can be created which will provide the needed data for communicating with Twitter API.



**Figure 4.1: Tweeter app setting**

This is a screen which has all of the data needed to talk to Twitter network. It is important to note that by default, the app has no access to direct messages, so by going to the settings and changing the appropriate option to “Read, write and direct messages”, you can enable your app to have access to every Twitter feature.

Tweepy provides access to the well documented Twitter API. With Tweepy, it's possible to get any object and use any method that the official Twitter API offers. Main Model classes in the Twitter API are Tweets, Users, Entities and Places. Access to each returns a JSON-formatted response and traversing through information is very easy in Python.

One of the main usage cases of Tweepy is monitoring for tweets and doing actions when some event happens. Key component of that is the StreamListener object, which monitors tweets in real time and catches them.

The Stream object is created, which uses that listener as output. Stream, being another important object in Tweepy also has many methods, in this case filter() is used with parameters passed. "follow" is a list of followers whose tweets are monitored, and "track" is a list of hashtags which will trigger the StreamListener.

**4.2 TextBlob**

TextBlob is a Python (2 and 3) library for processing textual data. It provides a simple API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more.

The basic steps for using TextBlob are:

* Create a TextBlob object, passing a string with the text we want to work with.
* Use various methods and attributes of the resulting object to get at various parts of the text.

For example, here’s how to get a list of sentences from a string:

The .sentences attribute of the resulting object is a list of sentences in the text. (Much easier than trying to split on punctuation, right?)

Each sentence object also has an attribute .words that has a list of words in that sentence.

The TextBlob object also has a “.noun\_phrases” attribute that simply returns the text of all noun phrases found in the original text:

TextBlob can also tell us what part of speech each word in a text corresponds to. It can tell us if a word in a sentence is functioning as a noun, an adjective, a verb, etc. In NLP, associating a word with a part of speech is called “tagging.” Correspondingly, the attribute of the TextBlob object we’ll use to access this information is .tags.

This for loop is a little weird, because it has two temporary loop variables instead of one. (The underlying reason for this is that .tags evaluates to a list of two-item tuples, which we can automatically unpack by specifying two items in the for loop. Don’t worry about this if it doesn’t make sense. Just know that when we’re using the .tagsattribute, you need two loop variables instead of one.) The first variable, which we’ve called word here, contains the word; the second variable, called pos here, contains the part of speech.

**4.3 Django**

With Django, you can take Web applications from concept to launch in a matter of hours. Django takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source.

Django was designed to help developers take applications from concept to completion as quickly as possible.

Django includes dozens of extras you can use to handle common Web development tasks. Django takes care of user authentication, content administration, site maps, RSS feeds, and many more tasks - right out of the box.

Django takes security seriously and helps developers avoid many common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery and clickjacking. Its user authentication system provides a secure way to manage user accounts and passwords.

Some of the busiest sites on the planet use Django’s ability to quickly and flexibly scale to meet the heaviest traffic demands.

**What does Django code look like?**

When a request is received the application works out what is needed based on the URL and possibly information in POST data or GETdata. Depending on what is required it may then read or write information from a database or perform other tasks required to satisfy the request. The application will then return a response to the web browser, often dynamically creating an HTML page for the browser to display by inserting the retrieved data into placeholders in an HTML template.Django web applications typically group the code that handles each of these steps into separate files:

**Figure 4.2: Django Architecture.**

* **URLs:**While it is possible to process requests from every single URL via a single function, it is much more maintainable to write a separate view function to handle each resource. A URL mapper is used to redirect HTTP requests to the appropriate view based on the request URL. The URL mapper can also match particular patterns of strings or digits that appear in an URL, and pass these to a view function as data.
* **View:** A view is a request handler function, which receives HTTP requests and returns HTTP responses. Views access the data needed to satisfy requests via *models*, and delegate the formatting of the response to *templates*.
* **Models:** Models are Python objects that define the structure of an application's data, and provide mechanisms to manage (add, modify, delete) and query records in the database.
* **Templates:** A template is a text file defining the structure or layout of a file (such as an HTML page), with placeholders used to represent actual content. A *view* can dynamically create an HTML page using an HTML template, populating it with data from a *model*. A template can be used to define the structure of any type of file; it doesn't have to be HTML!

[**Sending the request to the right view (urls.py)**](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction#Sending_the_request_to_the_right_view_(urls.py))

A URL mapper is typically stored in a file named **urls.py**. In the example below, the mapper (urlpatterns) defines a list of mappings between routes (specific URL patterns) and corresponding view functions. If an HTTP Request is received that has a URL matching a specified pattern then the associated view function will be called and passed the request.

urlpatterns = [

path('admin/', admin.site.urls),

path('book/<int:id>/', views.book\_detail, name='book\_detail'),

path('catalog/', include('catalog.urls')),

re\_path(r'^([0-9]+)/$', views.best),

]

**Figure 4.3: URLS.PY**

The urlpatterns object is a list of path() and/or re\_path() functions (Python lists are defined using square brackets, where items are separated by commas and may have an optional trailing comma. For example: [item1, item2, item3,]).

The first argument to both methods is a route (pattern) that will be matched. The path() method uses angle brackets to define parts of a URL that will be captured and passed through to the view function as named arguments. The re\_path() function uses a flexible pattern matching approach known as a regular expression. We'll talk about these in a later article!

The second argument is another function that will be called when the pattern is matched. The notation views.book\_detail indicates that the function is called book\_detail() and can be found in a module called views (i.e. inside a file named views.py.

**Handling the request (views.py**)

Views are the heart of the web application, receiving HTTP requests from web clients and returning HTTP responses. In between, they marshall the other resources of the framework to access databases, render templates, etc.

The minimal view function index(), which could have been called by our URL mapper in the previous section.  Like all view functions it receives an HttpRequest object as a parameter (request) and returns an HttpResponse object. In this case we don't do anything with the request, and our response simply returns a hard-coded string. We'll show you a request that does something more interesting in a later section.

Views are usually stored in a file called **views.py**

.

### 4.4 Defining data models (models.py)

Django web applications manage and query data through Python objects referred to as models. Models define the structure of stored data, including the field types and possibly also their maximum size, default values, selection list options, helps text for documentation, label text for forms, etc. The definition of the model is independent of the underlying database — you can choose one of several as part of your project settings. Once you've chosen what database you want to use, you don't need to talk to it directly at all — you just write your model structure and other code, and Django handles all the dirty work of communicating with the database for you.

The very simple Django model for a Team object. The Team class is derived from the django class models.Model. It defines the team name and team level as character fields and specifies a maximum number of characters to be stored for each record. The teamlevel can be one of several values, so we define it as a choice field and provide a mapping between choices to be displayed and data to be stored, along with a default value.

### 4.5 Querying data (views.py)

The Django model provides a simple query API for searching the database. This can match against a number of fields at a time using different criteria (e.g. exact, case-insensitive, greater than, etc.), and can support complex statements (for example, you can specify a search on U11 teams that have a team name that starts with "Fr" or ends with "al").

The view function (resource handler) for displaying all of our U09 teams. The line in bold shows how we can use the model query API to filter for all records where the team\_level field has exactly the text 'U09' (note how this criteria is passed to the filter() function as an argument with the field name and match type separated by a double underscore: **team\_level\_\_exact**.

This function uses the render() function to create the HttpResponse that is sent back to the browser. This function is a shortcut; it creates an HTML file by combining a specified HTML template and some data to insert in the template (provided in the variable named "context"). In the next section we show how the template has the data inserted in it to create the HTML.

### 4.6 Rendering data (HTML templates)

Template systems allow you to specify the structure of an output document, using placeholders for data that will be filled in when a page is generated. Templates are often used to create HTML, but can also create other types of document. Django supports both its native templating system and another popular Python library called Jinja2 out of the box (it can also be made to support other systems if needed).

The HTML template called by the render() function in the previous section might look like. This template has been written under the assumption that it will have access to a list variable called youngest\_teams when it is rendered (contained in the context variable inside the render() function above). Inside the HTML skeleton we have an expression that first checks if the youngest\_teams variable exists, and then iterates it in a for loop. On each iteration the template displays each team's team\_name value in an [<li>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/li) element.

## What else can you do?

The preceding sections show the main features that you'll use in almost every web application: URL mapping, views, models and templates. Just a few of the other things provided by Django include:

* **Forms**: HTML Forms are used to collect user data for processing on the server. Django simplifies form creation, validation, and processing.
* **User authentication and permissions**: Django includes a robust user authentication and permission system that has been built with security in mind.
* **Caching**: Creating content dynamically is much more computationally intensive (and slow) than serving static content. Django provides flexible caching so that you can store all or part of a rendered page so that it doesn't get re-rendered except when necessary.
* **Administration site**: The Django adminstration site is included by default when you create an app using the basic skeleton. It makes it trivially easy to provide an admin page for site administrators to create, edit, and view any data models in your site.
* **Serialising data**: Django makes it easy to serialise and serve your data as XML or JSON. This can be useful when creating a web service (a web site that purely serves data to be consumed by other applications or sites, and doesn't display anything itself), or when creating a website in which the client-side code handles all the rendering of data.

**4.7 Selenium**

Selenium is a tool to test your web application. You can do this in various ways, for instance

* Permit it to tap on buttons
* Enter content in structures
* Skim your site to check whether everything is "OK" and so on.

Web UI Automation means the automatic execution of the actions performed in a web browser window like navigating to a website, filling forms that include dealing with text boxes, radio buttons and drop downs, submitting the forms, browsing through web pages, handling pop-ups and so on. Selenium WebDriver is the one that can automate all these tasks. It can interact with all types of Web browsers available till date like Firefox, Internet Explorer, Safari, and Chrome, etc.

Selenium is an open source, and its library is available in different programming languages to perform the Web UI Automation testing, and Python is one of them.

Selenium WebDriver Client Library for Python enables us to utilize all the features available with Selenium WebDriver and interact with Selenium Standalone Server to perform Automated testing (both remote and distributed testing) of browser-based applications.

**CHAPTER - 5**

**RESULTS**

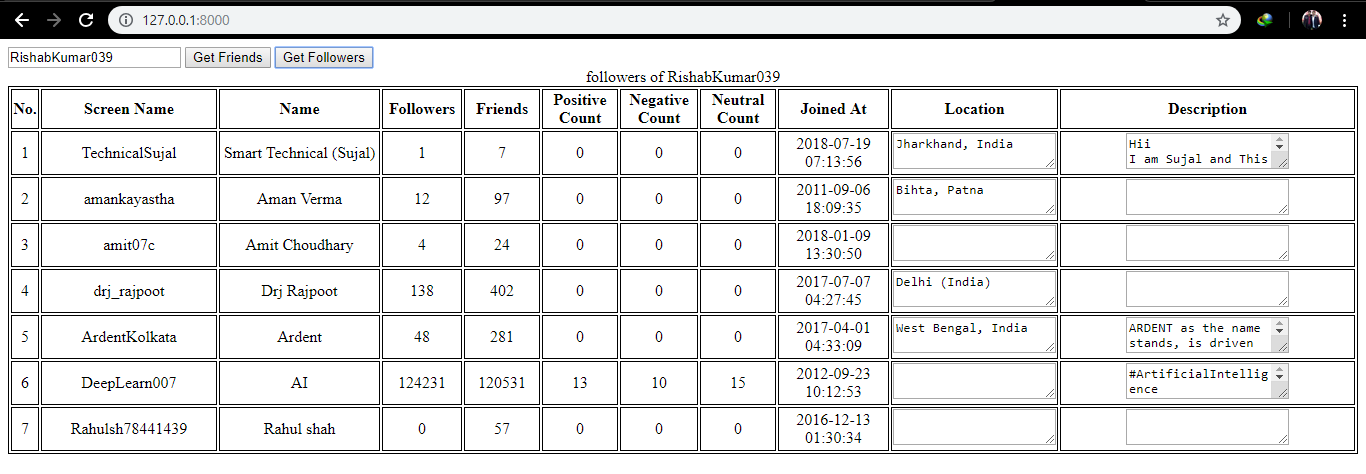
Creation and management of honeypot has been completed with the proper monitoring of social account.

First the artificial profiles have been created and the admin has monitored the user’s account and extracted their friend’s information and found the positive,negative and neutral count of each friend by analyzing his tweets which was tweeted by him and the user who has higher negative count has been kept under the list of suspicious and the dashboard has been created which shows the friend’s data and their sentiments as shown in figure 5.1.

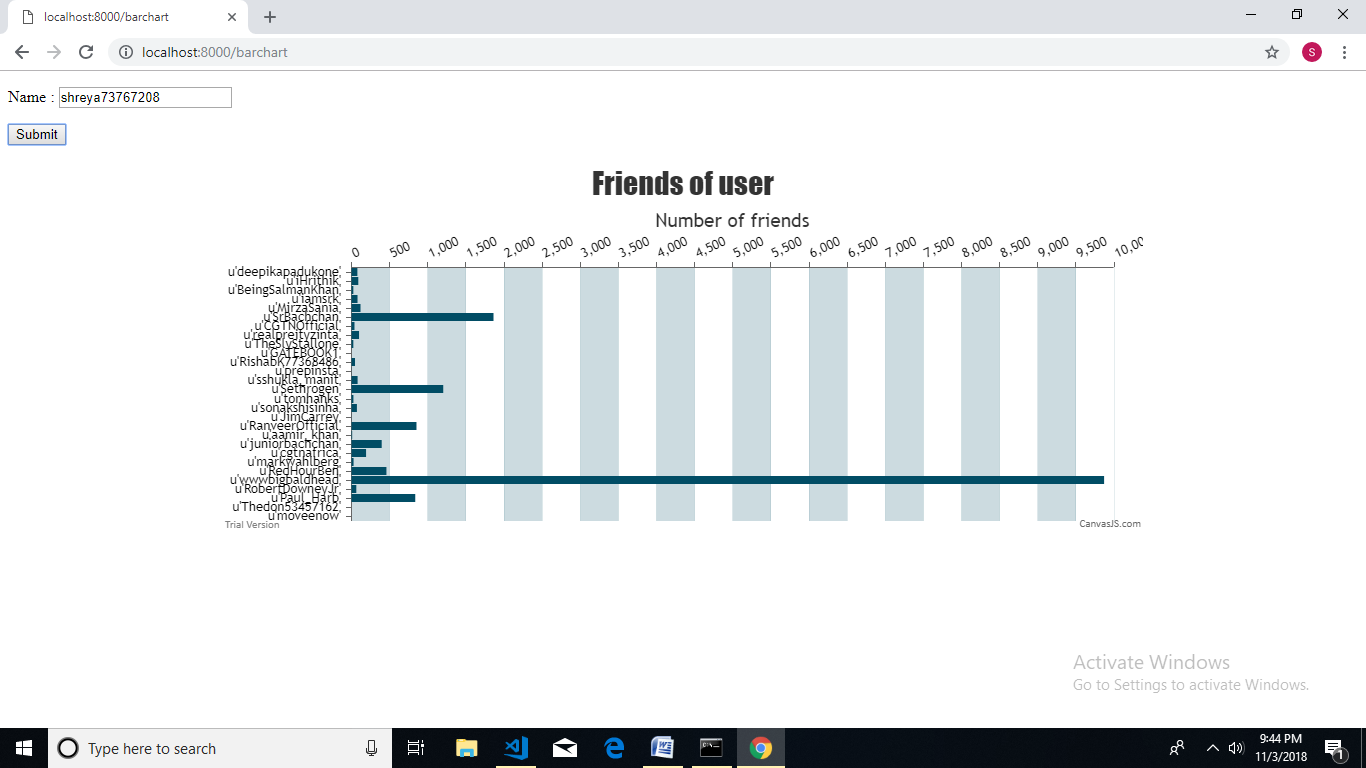
The Graph has been plotted which shows the name of each friends and their no. of friends which shows how many people are in his network as shown in figure 5.2.

The next part of project i.e security part has been implemented in which we check the password strength of each user by trying to login into their account by passing different password combination which most of the people generally keep like their date of birth,phone no. etc. and the user whose account has been logged in successfully it means their password strength is not good and can be easily hacked.

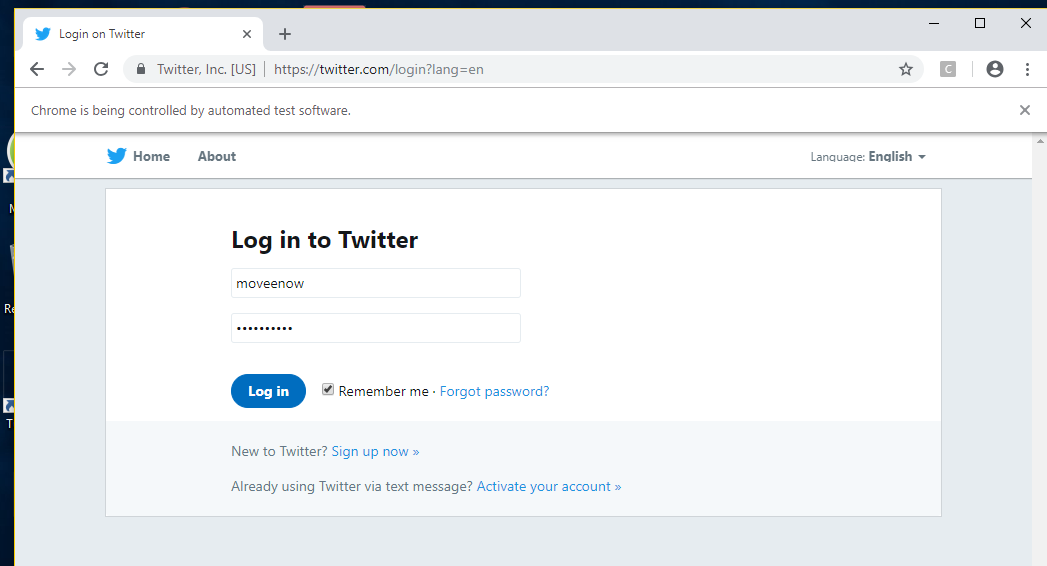
Here in this case tweeter account has been taken into consideration to check the security of one’s account on tweeter.The figure 5.3 shows the that the attempt is being made to login into user’s account.



**Figure 5.1:-dashboard of friens data.**

****

**Figure 5.2:- Graph of user’s friends.**

****

**Figure 5.3:-Login to tweeter account.**

**CHAPTER – 6**

**CONCLUSION AND FUTURE SCOPE**

The system has been designed to detect various attacks on social media like Tweeter, Facebook etc. and to also secure the user’s account by analyzing their authentication record.

Interaction with the various OSN (Online Social Network) has been done by using their API’s.

Then any user’s account has been logged in and analyze their account by analyzing their tweets, counting of no. of friends to know that how many people are there in his network.

In the implementation of security part of the project, the security has been done by trying to login into user’s friend’s account and by passing the combination of very general password that most of the people keep which can be easily detected by hacker.

By passing various general combination of password if one is able to login into other’s account means that the person whose account has been logged in has kept a very weak password.

**REFRENCES**

1. Abigail Paradise, Asaf Shabti,Rami Puzis, Aviad Elyashar, (2017) “Creation and Detection of Social Network Honeypots for Detecting Targeted Cyberattack”, IEEE Transactions on Computational Social Systems,pp: 0.
2. R. Jasek, M. Kolarik, and T. Vymola,(2013) “APT detection system using honeypots,” in *Proc. 13th Int. Conf. Appl. Informat. Commun. (AIC)*,pp:0.
3. A. Paradise, R. Puzis, and A. Shabtai, (2014) “Anti-reconnaissance tools: Detecting targeted socialbots,” IEEE Internet Comput., vol. 18, no. 5, pp. 11-19.
4. M. Ask, P. Bondarenko, J. E. Rekdal, A. Nordbo, Ruthven, and P. B. Nordbo (2013) “Advanced persistent threat (APT) beyond the hype,” Presented at the IMT4582 Netw. Secur. GjoviN Univ. College, pp- 0.

**Appendices**

Tweeterlogin.py

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import Select

from selenium.common.exceptions import NoSuchElementException,StaleElementReferenceException

import unittest, time, re

import os

from selenium.webdriver.chrome.options import Options

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.common.keys import Keys

from selenium.webdriver.support.ui import WebDriverWait

import threading

import time

import tweepy

from tweepy import OAuthHandler

from tweepy import Cursor

import re # regular expression

from textblob import TextBlob #text/tweet parse

from itertools import permutations

'''

options='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890@!#$%^&\*'

for w in permutations(options,8):

print w

'''

ck="Jm6WDXZuiwlwUnT9mFbPdSpcg"

cs="Wp4Gbf74R6MZWjXvj0ifKrCobwWbchhn53Mv8L7VMLbIUi6Wnd"

at="919434545924935681-wFjVVTbs0pmyB2VwSoj4VwGb7tBYCyr"

ats ="RaPfxU0rSMjkS3MSI9N0ztXu4I2iLMecXg79OerNHw4Ly"

auth = OAuthHandler(ck, cs)

# set access token and secret

auth.set\_access\_token(at, ats)

# create tweepy API object to fetch tweets

api = tweepy.API(auth)

fr=[]

user=api.get\_user('shreya73767208')

for friend in user.friends(count=200):

fr.append(friend.screen\_name)

#api.send\_direct\_message('1053918664112500741L',text='message text here')

#####################################################################################################################################################

browser = webdriver.Chrome("C:/Users/mihir/Desktop/chromedriver\_win32/chromedriver.exe")

url = 'https://twitter.com/login?lang=en'

browser.get(url)

browser.implicitly\_wait(30)

trap=[]

for f in fr:

pwd=[]

first=""

for i in f:

if not i.isdigit():

first=first+i

pwd.append(f+"@1")

for i in range(0,2):

pwd.append(first.lower()+str(i))

for p in pwd:

print p

browser.find\_element\_by\_class\_name('js-username-field').send\_keys(f)

browser.find\_element\_by\_class\_name('js-password-field').send\_keys(p)

browser.find\_element\_by\_css\_selector('button.submit.EdgeButton.EdgeButton--primary.EdgeButtom--medium').click()

browser.implicitly\_wait(30)

if browser.current\_url==("https://twitter.com/login/error?username\_or\_email="+f):

browser.refresh()

browser.get(url)

else:

trap.append(f)

browser.get('https://twitter.com/settings/devices')

print browser.find\_element\_by\_css\_selector('span.device\_number\_with\_country\_code').text

browser.find\_element\_by\_css\_selector('a#user-dropdown-toggle.btn.js-tooltip.settings.dropdown-toggle.js-dropdown-toggle').click()

browser.find\_element\_by\_css\_selector('li#signout-button.js-signout-button').click()

#browser.get('https://twitter.com/settings/devices')

#print browser.find\_element\_by\_css\_selector('span.device\_number\_with\_country\_code')

browser.get(url)

browser.implicitly\_wait(30)

break

print trap

Barchart.html

<!DOCTYPE HTML>

<html>

<head>

<meta charset="UTF-8">

<script src="http://code.jquery.com/jquery-latest.min.js"></script>

<body>

<p>

Name : <input type="text" name="name" id="name" />

</p>

<p>

<input type="button" value="Submit" onclick="testSave()" />

</p></body>

{% load static %}

<script>

function testSave()

{

var name;

name = document.getElementById('name').value;

//alert("hi")

//

$.get("saveTest1?name="+name, function(responseText) {

// Execute Ajax GET request on URL of "someservlet" and execute the following function with Ajax response text...

alert(responseText);

d=responseText.split("]");

var out = new Array();

for(var i=0; i<d.length-1;i++)

{

s = d[i].split(",");

lab = s[0].replace("[","");

value =s[1].replace(" ","");

value =parseInt( value.replace("'",""));

//alert(lab+"|"+value);

out.push({label:lab,y:value});

}

var chart = new CanvasJS.Chart("chartContainer", {

animationEnabled: true,

title:{

text:"Friends of user"

},

axisX:{

interval: 1

},

axisY2:{

interlacedColor: "rgba(1,77,101,.2)",

gridColor: "rgba(1,77,101,.1)",

title: "Number of friends"

},

data: [{

type: "bar",

name: "companies",

axisYType: "secondary",

color: "#014D65",

dataPoints:out

}]

});

chart.render();

});

}

/\*

\*/

</script>

</head>

<body>

<div id="chartContainer" style="height: 370px; max-width: 920px; margin: 0px auto;"></div>

<script src="{% static 'canvasjs.min.js' %}"></script>

</body>

</html>

**FIRSTPAGE.HTML**

<html>

<head>

<script src="http://code.jquery.com/jquery-latest.min.js"></script>

</head>

<ul>

<li><a href="barchart">See Graph</a></li>

<li><a href="trap">See trapped users</a></li>

<body>

</html>

**TRAP.HTML**

<html>

<head></head>

<body>

<form action="savetest2" method="get" >

<input type="submit" value="Get" />

</form>

<h1>List of trapped users</h1>

{% for row in data %}

{% for col in row %}

<h2 align="center">{{ col }}</h2>

{% endfor %}

{% endfor %}

</body>

</html>

**FRIENDS.PY**

import tweepy

from tweepy import OAuthHandler

from tweepy import Cursor

import re # regular expression

from textblob import TextBlob #text/tweet parse

ck="Jm6WDXZuiwlwUnT9mFbPdSpcg"

cs="Wp4Gbf74R6MZWjXvj0ifKrCobwWbchhn53Mv8L7VMLbIUi6Wnd"

at="919434545924935681-wFjVVTbs0pmyB2VwSoj4VwGb7tBYCyr"

ats ="RaPfxU0rSMjkS3MSI9N0ztXu4I2iLMecXg79OerNHw4Ly"

auth = OAuthHandler(ck, cs)

# set access token and secret

auth.set\_access\_token(at, ats)

# create tweepy API object to fetch tweets

api = tweepy.API(auth)

#print(api,' login success ')

def getSentiment(tweet):

analysis = TextBlob(tweet)

#print(analysis)

#print(analysis.sentiment.polarity)

# set sentiment

if analysis.sentiment.polarity > 0:

return 'positive'

elif analysis.sentiment.polarity == 0:

return 'neutral'

else:

return 'negative'

def cleanData(tweet):

return ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t]) |(\w+:\/\/\S+)", " ", tweet).split())

def getTweets(p,coun):

tweets = api.search(q=p,count=coun)

pc = 0

nc = 0

netc = 0

for t in tweets:

tweetlist= cleanData(t.text)

#print(tweetlist)

data = getSentiment(tweetlist)

tweetdata = []

tweetdata.append(data)

for t in tweetdata:

if t == 'positive':

pc =pc+1

elif t == 'negative':

nc = nc+1

else:

netc = netc+1

#print('user name :',p)

#print ('positive count ',pc)

#print ('negative count ',nc)

#print ('neutral count ',netc)

sent=[]

sent.append(pc)

sent.append(nc)

sent.append(netc)

return sent

user=api.get\_user('shreya73767208')

#print user.screen\_name

def frndz1(n):

user=api.get\_user(n)

ct=0

person=[]

list2=[]

for friend in user.friends(count=200):

#print friend.screen\_name

ct=ct+1

#list2=[]

list2.append(friend.screen\_name)

list1=list(list2)

list2=[]

for l in list1:

list3=[]

data1=getTweets(l,40)

#print data1

user = api.get\_user(id = l)

sc = user.screen\_name

user\_profile = api.get\_user(screen\_name=sc)

data = []

data.append(['ID ',user\_profile.id]);

data.append(['Screen Name',user\_profile.screen\_name]);

data.append(['Name ',user\_profile.name]);

data.append(['Followers ',user\_profile.followers\_count]);

data.append(['Location ',user\_profile.location])

data.append(['Description',user\_profile.description])

data.append(['FriendCount',user\_profile.friends\_count])

data.append(['Joined At ',user\_profile.created\_at])

#print(str(data[0]) + ' : ' + str(data[1]) + ' : ' + str(data[6]))

l3=str(data[6]).split(',')

list3.append(l)

list3.append((l3[1].replace(']','')).replace(' ',''))

list2.append(list3)

return(list2)

#frndz1('shreya73767208')

**VIEWS.PY**

# -\*- coding: utf-8 -\*-

from django.shortcuts import render

from django.shortcuts import render\_to\_response

from django.http import HttpResponse

from . import connectionex

from . import friends

from . import tweeterloginselenium

#from . import friends

def index(request):

#return render\_to\_response("blog/bar-chart.html")#initially index.html

return render\_to\_response("blog/firstpage.html")

def about(request):

return render\_to\_response("blog/about.html")

return render\_to\_response("blog/contact.html")

return render\_to\_response("blog/signup.html")

return render\_to\_response("blog/login.html")

return render\_to\_response("blog/index.html")

# Create your views here.

def savetest2(request):

o=tweeterloginselenium.get()

return render\_to\_response("blog/trap.html",{'data':o})

#def showfriends(request):

# o=friends.frndz()

# t=friends.frndz1()

# return render\_to\_response("blog/bar-chart.html",{'data':o,'data1':t})

def home(request):

return render\_to\_response("blog/home.html")

return render\_to\_response("blog/about.html")

return render\_to\_response("blog/contact.html")

return render\_to\_response("blog/index.html")

def saveTest1(request):

n = request.GET['name']

s = friends.frndz1(n)

#s = [[u'user1','100'],[u'user2','120'],[u'user3','40'],[u'user4','60']]

return HttpResponse(s)

def firstpage(request):

return render\_to\_response("blog/firstpage.html")

#return render\_to\_response("blog/barchart.html")#also work if present

return render\_to\_response("blog/trap.html")

def barchart(request):

return render\_to\_response("blog/barchart.html")

def trap(request):

return render\_to\_response("blog/trap.html")

**URLS.PY**

from django.contrib import admin

from django.conf.urls import include, url

from . import views

#contain all functions and html

urlpatterns = [

url('about$',views.about,name="about"),

#url('saveTest$',views.saveTest,name="saveTest"),

url('showTest$',views.showTest,name="showTest"),

url('saveTest1$',views.saveTest1,name="saveTest1"),

url('savetest2$',views.savetest2,name="savetest2"),

#url('savetest$',views.savetest,name="savetest"),

url('flipkart$',views.flipkart,name="flipkart"),

url('search$',views.search,name="search"),

url('firstpage$',views.firstpage,name="firstpage"),

url('contact$',views.contact,name="contact"),

url('signup$',views.signup,name="signup"),

url('showcrawldata$',views.showcrawldata,name="showcrawldata"),

url('login$',views.login,name="login"),

url('barchart$',views.barchart,name="barchart"),

url('trap$',views.trap,name="trap"),

url('home$',views.home,name="home"),

url('',views.index,name="index"), # landing / default page

]