A Test of Protection Algorithm Against Mass Data Harvesting

To what extent could new approaches to web-scraping bypass current data protection algorithms and allow LinkedIn's public profile data to be harvested?

Word Count: 3,988

Table of Content

Introduction

Privacy concerns have become one of the prominent issues of contemporary society. In light of recent events such as Facebook and Cambridge Analytica, where data of users are being harvested and shared without their knowledge, concerns were brought up about how the attack was possible in order to increase the capacity to defend against it. These type of attacks are sophisticated and harder to prevent; however, a more common, simpler attack involves gathering publicly available data through web-scraping.

To understand the privacy issue, the implications of data being harvested need to be explored. First, users' data that are collected could be used to steal their identity which would cause financial harm. Additionally, companies will be faulted if their data are being harvested which will lead to public distrust and hurt their revenue stream. The data collected could also help their competitors in the market. Thus, in hope of retaining users, sites such as LinkedIn claim to have the strongest protection algorithm against mass data harvest. Whilst I am tempted by the intellectual challenge of bypassing such algorithm, I sympathize with users that do not want information that they put online to appear in hundreds of other databases as I fear that my own data is at risk as well. Therefore, I decided to pursue this research with the question: ***To what extent could new approaches to web-scraping bypass current data protection algorithms and allow LinkedIn's public profile data to be harvested?***

Secondary Research

Web scraping is a term for various methods used to collect information on the Internet ("Web Scraping"). The Selenium library allows the automation of a web browser ("Browser Automation"). Utilizing this library enables a user to automate browser processes and collect data on different web pages.

Besides a human being, another type of user of the World Wide Web is a bot. Internet bots are software that performs automated tasks on the net, typically more efficiently than human (Dunham). Since web scraping scripts are essentially bots, to protect against them, one must be able to detect and stop it.

A basis for such a defense strategy could involve the use of weblog. Everytime a user access the internet, the web server logs the visit in a "web log file" which includes information on the user. There are four types of log files: "a) Access: Data of all incoming request and information about the client of the server. b) Error: list of internal errors. c) Agent: Information about user 's browser. d) Referrer: provides information about link and redirects [the] visitor to the site" (Patil and Patil). These files give the developer information in order to debug the site and in this case, can be used to increase the chance of detection of a bot.

Currently, one of the most effective methods of preventing harvesting by bots is CAPTCHA, which stands for Completely Automated Public Turing Test to Tell

Computers and Humans Apart (Caspio). This could create inconveniences in the user's experience, as the algorithm to detect bots is not advanced in its implementations.

More obvious defense techniques are used in layer four of the OSI model which is the TCP/IP layer. This method cannot, however, detect bots and may even do more harm with its mitigation stopping legitimate users from accessing the site content (Radware, Ltd.). Such layer four methods include detecting if a particular client sent multiple requests in a short period of time, for example, 1,000 requests in 1,000 seconds (Synack, Inc.). However, these defenses can be bypassed using a so-called "bot-net" which is essentially a distributed network of bots that are coordinated in its request in order to anonymize itself against such detection methods (Synack, Inc.).

Methods such as CAPTCHA are layer seven defense methods. These "attempt to verify that a transaction is initiated by a legitimate client application and is under the control of the user" (Radware, Ltd.). However, only CAPTCHA has been the most effective since JavaScript redirect challenges and SYN Cookies techniques for validating IP can be bypassed by attackers with either a parser for JavaScript to get the intended address or a machine with an actual IP address (Radware, Ltd.). As of December 2016, the company Radware, Ltd has created a system to detect headless browser bots that are normally used to attack such sites either for access of data or for disruptions of the usual operations of the sites. Radware, Ltd has not disclosed their techniques but as such a system could also be bypassed by an automated bot with a head i.e. using a browser application with a display on a desktop. However, if this was the case, then

Radware, Ltd would have at least succeeded in making the attacker reveal their own machine in the case that they do not have access to other non-headless bots.

Methodology

The objective of the following experiments is to demonstrate LinkedIn's ability to defend against mass data harvesting by identifying and stopping the bots. The defense will be considered successful in this essay if it stops the bots from reaching more than 400 profiles which are 40% of all the profiles that I will try to visit.

To conduct this research, I will run the self-written scripts that utilize Selenium webdriver library for Java. It takes in a randomized list of 1000 LinkedIn profile URLs as the input. The scripts will run until LinkedIn noticed that a bot is browsing their profiles and log the user out.

There will be 2 different scenarios for testing different strategies:

1. Random timing:

Different delays in hitting each profile in order to seem more human-like. This could provide an advantage for the bot assuming LinkedIn has a database hit-rate detection strategy. If this is unsuccessful, then

2. IP Addresses Rotation:

Changing access points could bypass the protection against autonomous scraping if the server does not think that the same machine is gathering data reducing the likeliness of being a bot.

A third method would be distributed scraping and it could be very effective. However, due to limited resources, I can only theoretically discuss the result and implication of such method later in the essay. There are 2 ways to approach this method: (a)With the same profile: This would allow for multiple IPs and timings with different platforms and identifiers. The server would not think that this is a bot since it has many unique identifiers but the profile could still be tracked. (b)With multiple profiles: This is the same as part (a) but with a higher likelihood of success due to the different profiles being used, therefore, harder to trace.

Results

The result collected were of the counts of profiles that have been visited before LinkedIn stop the bot i.e. logging the user account out. The details include the profiles count, time (experiment started, ended, and elapsed), and the rate at which the profiles were visited.

I first conducted a control experiment where a scraper bot is automated to visit every profile on the 1,000 anonymized LinkedIn users. As you can see in Appendix A, the profile visited includes details from their full name, job titles, and location to work experiences and education institutions which are publicly accessible from their LinkedIn profiles. In this first run, the bot visited 311 profiles in 36 minutes until the system detected it and log out the user. This means that approximately 8 different profiles are visited every minute by this bot and those individuals data had been collected.

On the second run, I used the timing script which has a delay between 4 to 9 seconds for each profile visit. As discussed this was to simulate the action of a human browsing the page. I could have embedded javascript to scroll the page making it behave even further like a human but for the purpose of this experiment I am going to keep to the landing page. Surprisingly, I managed to go through 1,000 profiles without ever being logged out in 4 hours and 23 minutes(See Appendix C). This means the bot visited approximately 3 profiles per minute.

For the last part of the experiment, I ran the IP Addresses Rotation script which uses 79 different IP addresses that rotates every 20 profiles or when the time to load the page takes over 20 seconds. The IP addresses used are randomly selected from a wide part of the world e.g. Brazil, Indonesia, etc as plotted in Figure 1. (See Appendix E) After I ran the script I was surprised to see that it only visited 133 profiles in 1 hour and 53 minutes before the system log it out. This puts the bot at the slowest rate: approximately 1 profile per minute.

Figure 1: Distribution of Proxies

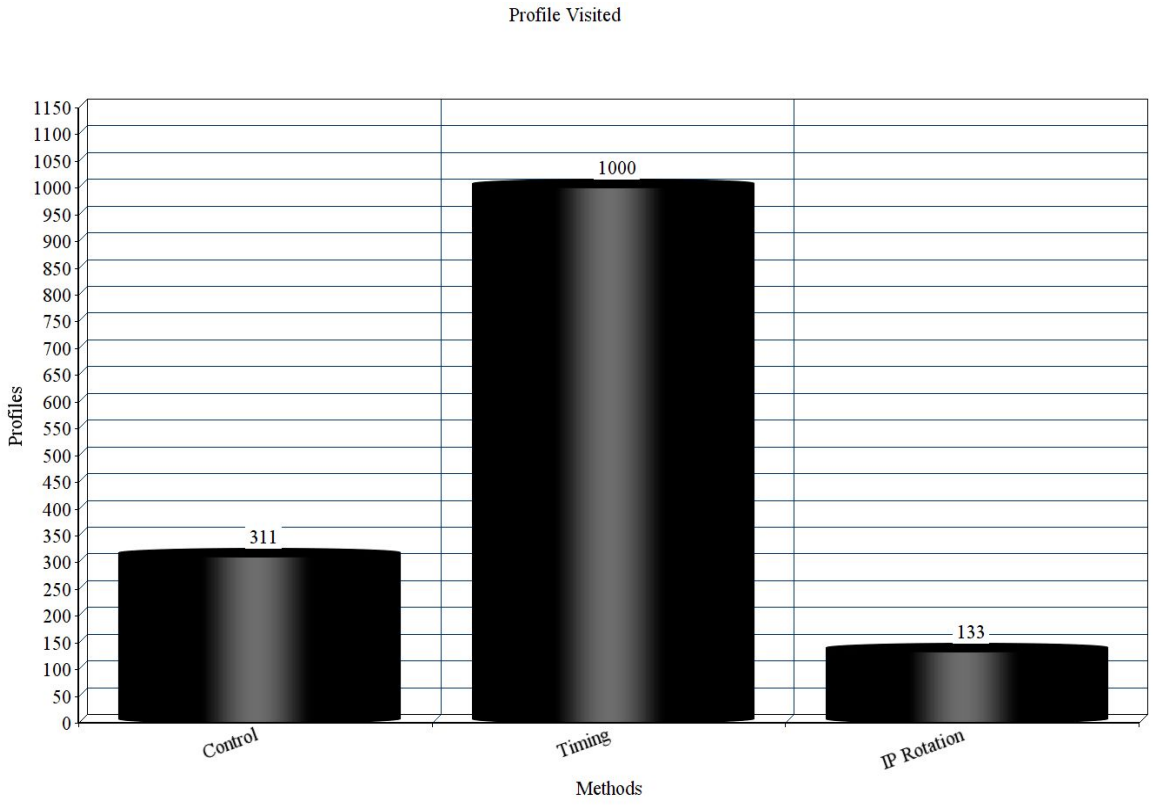


Figure 2: Barchart of Results

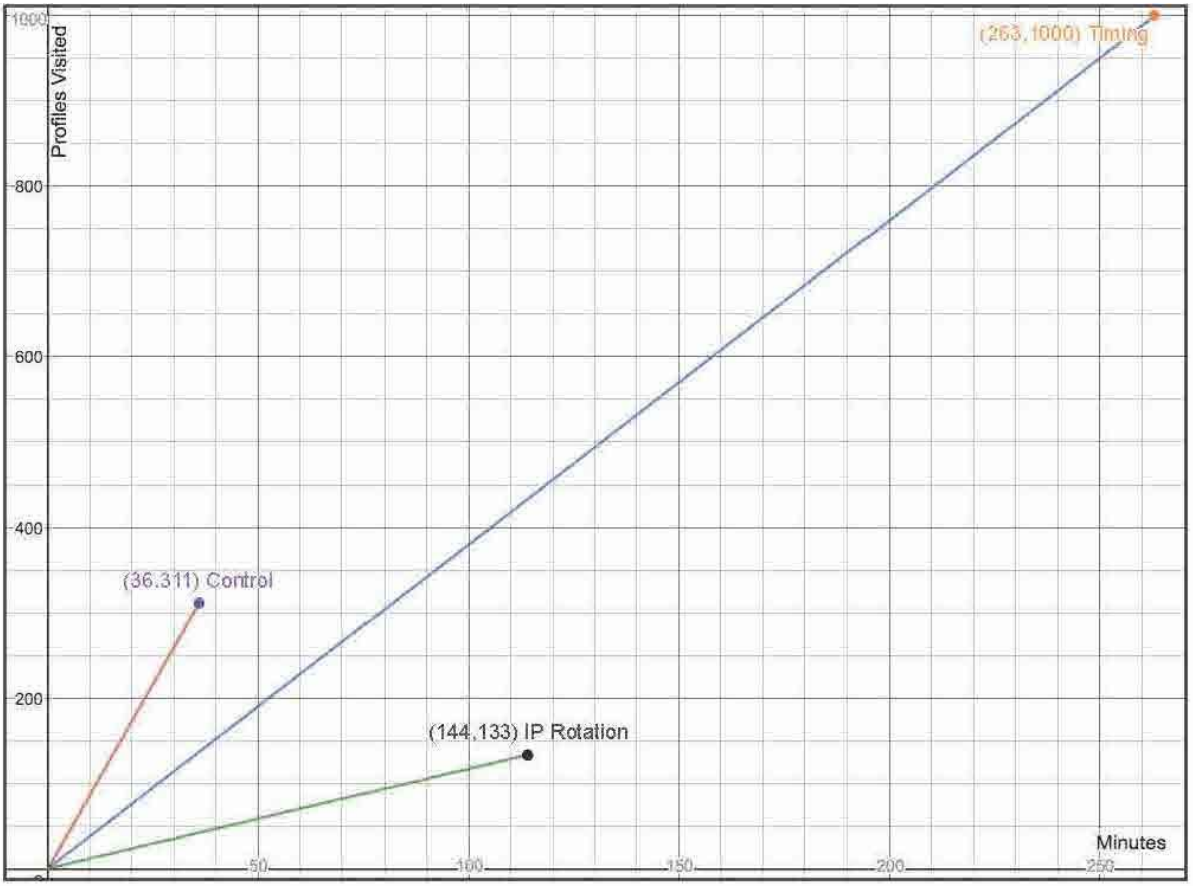


Figure 3: Line Graph of Results(Profile Visited Against Time)

Analysis

For the control experiment, the query rate for the database is at maximum the bot could work at. As it was stopped at 311 profiles, we can safely assume that LinkedIn can detect abnormally fast query rate to its database and therefore recognize that the user is not a human. There are two possible ways that LinkedIn can do this, one is by the account, the other is by the IP address.

However, to counter the hit rate detection, I have used the delay timing script to simulate human browsing behavior; this means that it will browse to the other profiles randomly between 4 to 9 seconds. It apparently succeeded as it achieved the 1,000

profiles with no ban in about 4 hours and a half. This might raise the question of whether the site has any other kind of defenses against data harvesting by bots. Another possible reason for this success might be because the time the bot remained inactive was longer than expected out time that LinkedIn has set, if that is the case then multiple more trials could be run in order to determine the breaking point of the defense algorithm and therefore find the delay timing of browsing that would be optimal for a certain amount of profiles that need to be collected i.e. using the smallest delay window to get the greatest amount of profiles possible.

Surprisingly, whilst testing the second part of the site's defense using the IP addresses rotation script, the site stopped the bot at just 133 profiles. This suggests that the site knows that even though the profiles are accessed from different locations (due to the alterations of IP addresses) it is still coming from the same person and the behavior of switching IP addresses must be a tell-tale sign for a bot. It seems that LinkedIn's IP addresses rotation defense works better than the timing one in detecting a bot.

However, this could also be the fact that the gap between rotation is too wide. For the above result, I used 20 profiles as the point for rotation, meaning that for every 20 profiles, I change the IP address that the device is browsing from. This means a significant amount of delay as well since it takes time for the browser to relaunch with the new configuration. Additionally, if the site takes more than 20 seconds to reach the device will rotate to a new IP address since either the proxy is down or too slow for operational purposes. This condition would put a wide gap between every 20 profiles or

when a proxy is not responding. Since the rotation only happens every 20 profiles while the hit rate remained at maximum, the same as the control experiment, it could be the reason that the site detected and stopped the bot at 133 profiles.

Nonetheless, that still does not explain why it only scanned through 133 profiles while the control experiment reached 311 profiles. This limitation could be the result of all the experiment being done on the same account, LinkedIn could have recognized that this user has been using a bot and put more limitation onto the person i.e. observe the user closely to see if even the slightest sign of abnormal activity is happening.

Therefore, in order to gather more information, I decided to run the IP rotation script again but with the gap of rotation reduced to only 10 profiles per rotation. This means that after every 10 profiles or if the connection takes more than 20 seconds to be established the browser will quit and reopen with a new proxy configuration. To my surprise the new settings let the bot collect up to 950 profiles (See Appendix F); however, it took more than 6 hours to do that. Putting the new results at:

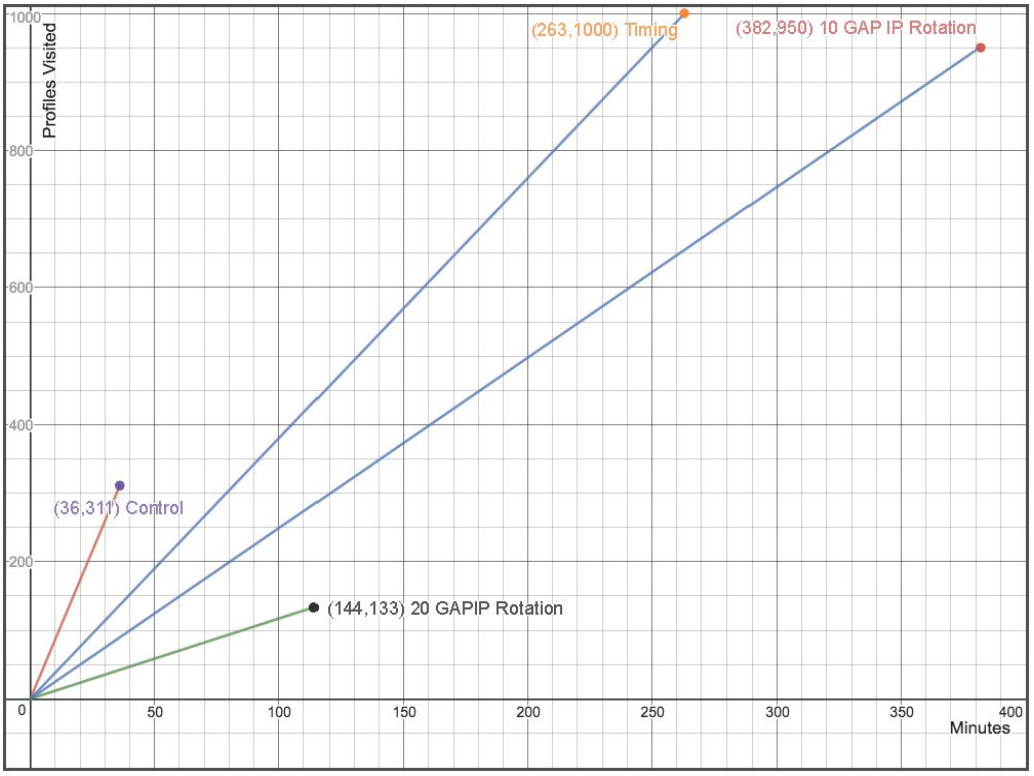


Figure 4: Updated Profile Visited against Time Graph

However, this result is still somewhat clouded since the delay between the 10 profiles rotation and the unexpected servers not responding contribute to the random delay that appears to be the factor of success in the Time Delay experiment. Nonetheless, the fact that this bot gathered over 900 profiles in a matter of 6 hours is still a very significant factor in this research. One possible reason why this could not reach the 1,000 profiles goal would be that the hit rate to the database within the 10 profiles gap would still be too high above the threshold set by the system. Additionally, the frequent IP addresses changes in the request by the same account along with the high hit rate might increase the chances of detection even further if LinkedIn were to have the ability to recognize requests from the same accounts as assumed above.

On top of that, LinkedIn might have other defenses that could include having an upper limit of all profiles visited by one account per day. If that were true then this

non-negotiable limit would be unbreakable and a strong defense against data harvesting to a certain extent. However, this approach would only slow the process of harvesting i.e. the bot would then be programmed to gather results under the limit and not get logged out then continue doing the same the next day. This detection avoidance would only work if there is a strict lifetime limit but that would impede a real-life human that are using LinkedIn for a very long time so it would push their loyalty away. Nevertheless, if that were to be implemented then attackers would simply create another account and continue until that one becomes invalid.

If we continue with the assumption that LinkedIn has an upper limit on overall profile visited by one account per day, one other hypothetical way to bypass that would be the one discussed in the methodology section of this essay. If a servlet was to be written so that multiple computers on the network could communicate with each other then the task of delegating say a list of 20,000 profiles would be split randomly between the machines (but not exceeding the limit of each one) then it would be randomly taken off the server one by one at random interval creating a network of information flowing into a single database. This influx of information would take significantly less time than any other method as it is not using one single account and would be very hard to detect. On top of that, each computer could be using different IP addresses in different countries to make it even harder to trace the hive. One possible way LinkedIn would be able to trace this if they were to find the pattern in the searches or profile visits. However, this would almost be undetectable unless it has a suspected group of accounts that it thinks are working together. Even if that were to happen the attacker could easily rotate accounts like the IP addresses rotations, I did in this study. This would mean

those different users from seemingly very different backgrounds working independently looking at random profiles at a random rate. There would be no distinguishing those types of bots from an actual human because they will behave almost exactly the same as a human. To optimize the performance of the network, IP addresses could be rotated less often while timings window could be decreased slightly and requests should be made at different times not all at the same time between each machine. An added layer of anonymity would also be good, if LinkedIn was to check for DNS(Domain Name Server) leakage which would mean they know that it is from the same local internet service provider which means the hive local area is recognized and could help LinkedIn in detecting these group, the attacker can easily use the publicly available DNS servers such as Google's public DNS: 8.8.8.8 or 8.8.4.4.

The only sure way, right now, to defeat a bot is a CAPTCHA challenge. There is no known way for a bot to defeat the challenge and would be the effective defense that LinkedIn can use. This would, however, inconvenience users but surely this will not discourage most people as they would not mind a little delay as long as they are sure that their profile is not being harvested. As of the time I am conducting this experiment I did not see any evidence of CAPTCHA being used on LinkedIn except for the signup page. However, to implement CAPTCHA effectively, LinkedIn needs to be able to detect the slightest hint that a bot is operating on its network. This means having to distinguish suspicious random activity to normal ones; otherwise, they would have to pose the challenge every time a user is loading a page which is a huge inconvenience and will lead the users to lose confidence in the system's ability to detect a bot. For example, if there were to be a database query rate at which the challenge was to be posed then the

attacker will just have to calibrate the bot to operate at the rate below that and if LinkedIn set the rate too low then a normal user that is visiting one profile to another a little too fast would also be posed the challenge; or if the attacker is using the random method described above then LinkedIn would need to go to extraordinary measures, which may include posing CAPTCHA at every loading screen, in order to detect and stop the bot. Therefore, if LinkedIn were to implement CAPTCHA, it would still require a very well rounded, sophisticated system of implementing to not push its users away and still defend its data.

Conclusion

In conclusion, we know that LinkedIn has at least basic protection against bots. Its network monitors for hit rates for its database that would exceed the normal human speed which only appears to come into effect after about 300 profiles. We know this from the control experiment, this means that if the profiles' data the attacker wants to gather on is less than 300 profiles then a simple bot would do the job perfectly without LinkedIn even notice.

With the above defense in mind, we can safely look at the second experiment with one of LinkedIn lenses. The second timing delay script is used to successfully bypass that exact hit rate filter by LinkedIn. By managing to pull data from 1,000 profiles in less than 5 hours, the result shows that LinkedIn has a gap in its defense that allows data to be harvested if between each database queries there is a pause of 4 to 9 seconds. Therefore, a patient attacker would be able to gather thousands of user profiles in a single day.

Even though at first it seemed LinkedIn was able to detect that a certain account has been making frequent database queries in inhuman speed from different IP addresses; however, the third experiment appears to have a flaw in its implementation that it triggers the hit rate detection that LinkedIn has, as proven in the control experiment, during the large 20 profiles gap where the IP address of the machine remained unchanged which prompts LinkedIn to detect and stop the bot after only 133 profiles data had been harvested. However, after adjusting for a small profile gap of 10 profiles where the IP address is changed, LinkedIn was not able to detect that there is a bot on their network until 950 user profiles had been harvested. This increase in data harvested could also be attributed to the delay that had been produced by the more frequent switches of IP addresses which required the shutting down and restarting the browser with a new proxy configuration. This suggests that LinkedIn cannot detect whether a certain user had been requesting information from multiple IP addresses and could mean that LinkedIn does not keep track of accounts for requests but only of IP addresses. Even though the rotation of IP addresses takes time, an optimal gap of profiles would let an attacker pass by unnoticed which means that if an attacker is patient enough and had done enough trials, he/she could potentially gather thousands of user profiles in a day.

Keeping these results in mind, the total number of LinkedIn users in 2016 was 467 million ("Number of LinkedIn Users."). Using 3.80 profiles/minute rate from the second experiment (See Appendix C), it would take an attacker on one computer an accumulative time of 234 years in order to gather 467 million people's public data from

LinkedIn without ever getting caught. Now imagine having a server farm using the hypothesized method of Distributive Scraping. Having just 10 machines would let the attacker do it in 23.4 years, 250 machines would reduce the time to less than a year, and 1000 machines would let an attacker accomplish that in a little over 2 months.

An overview of different defenses that LinkedIn could use was provided in the analysis section of the page; however, there are limitations to those defenses. Each of those defenses could be bypass with random delay timings and even a more advanced version with a network of computers which creates a hive of machines that gather information randomly mimicking a group of unrelated human being.

LinkedIn needs to provide a more sophisticated method of defense in order to keep bots from acting on their network. However, the most effective, but most likely will not be implemented, CAPTCHA challenge would be a significant challenge to put into practice as LinkedIn would struggle to maintain its client base.

Thus, I conclude that LinkedIn's data protection algorithm need significant improvements and whilst it is not implementing the best options, our data will be at risk of harvest.

Evaluation

The results of the experiments conducted were somewhat satisfactory as the component of the defenses could not be isolated completely to be tested against. As explained for the third experiment with the IP addresses rotation, the cause cannot be

singled down to the rotation alone but also to the timing created by the rotation. The research could have gone better if LinkedIn's defenses were known instead of trying to guess what it could be. However, this could actually be helpful as an outside attacker would have no idea what LinkedIn defenses could be either. This creates a more authentic experience for me and yields the most probable result an attack would yield.

In order to increase the quality of the methodology, more trials should be used with varying the time frame and IP addresses rotations gaps with different IP addresses group i.e. trying out only same regions IP addresses or mixed international ones. This would help give us an idea of how an attacker would calibrate their script in order to efficiently harvest data.

On top of that, many limitations were imposed on this research as I am a student with limited time and resources. The servlet described is hypothetical in this situation but it could definitely be engineered with enough knowledge. If an amateur researcher such as myself is able to produce this result, imagine what could a professional attacker with time and resources on their hand could do.

Even though I cannot single out the defenses responsible for blocking some of the bots, the very result of this experiment is evident that LinkedIn data protection algorithms have flaws. Therefore, I safely and confidently rely on this conclusion.

References

1. “Browser Automation.” *Selenium Blog Posts*, www.seleniumhq.org/.

2. Caspio. “Data Harvesting & How to Prevent It.” Caspio Blog, Caspio, 10 Nov. 2017, blog.caspio.com/what-you-need-to-know-about-data-harvesting-and-how-to-prevent-it/.

3. Dunham, Ken, and Jim Melnick. *Malicious Bots an inside Look into the Cyber-Criminal Underground of the Internet*. Auerbach Publications, 2009.

4. “Number of LinkedIn Users.” Statista, 2018, www.statista.com/statistics/274050/quarterly-numbers-of-linkedin-members/.

5. Patil, Priyanka V., and Ujawala M. Patil. "Survey on Preprocessing in Web Server Log Files." International Journal of Advanced Research in Computer Science, vol. 3, no. 1, 2012. ProQuest, https://search.proquest.com/docview/1443715868?accountid=74409.

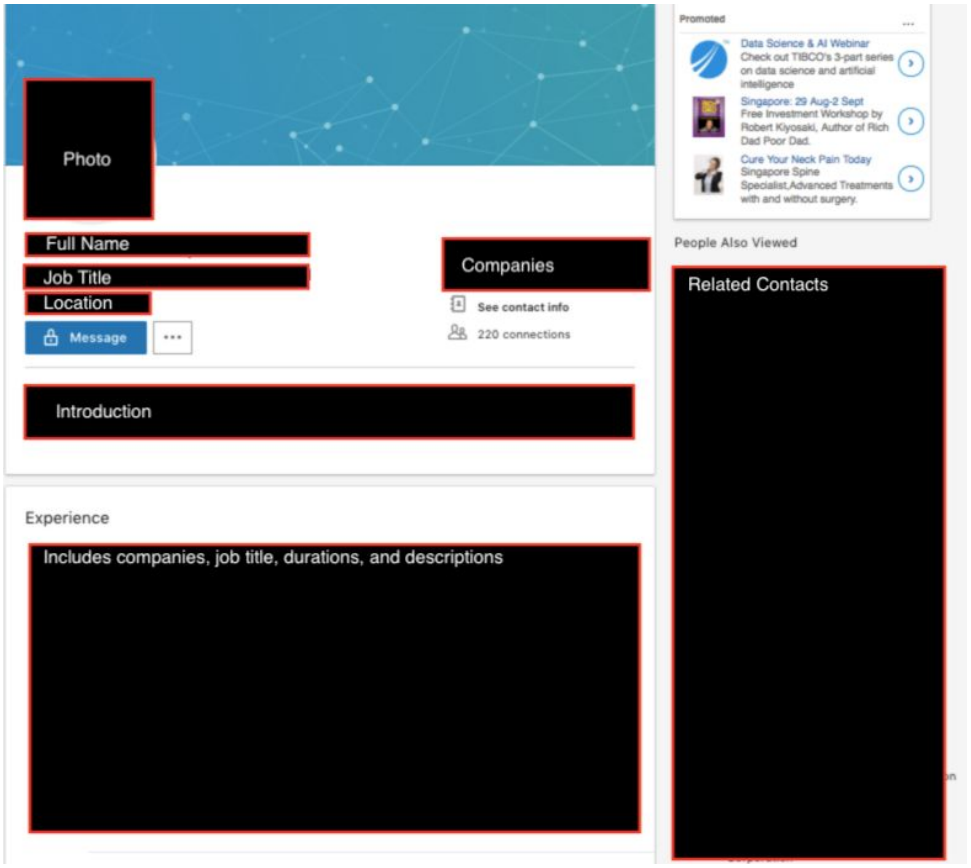
6. Radware, Ltd. "Radware, Ltd.; Patent Application Titled "Method and System for Detection of Headless Browser Bots" Published Online (USPTO 20160359904)." Computer Weekly News, Dec 28, 2016, pp. 1416. ProQuest, https://search.proquest.com/docview/1850928272?accountid=74409.

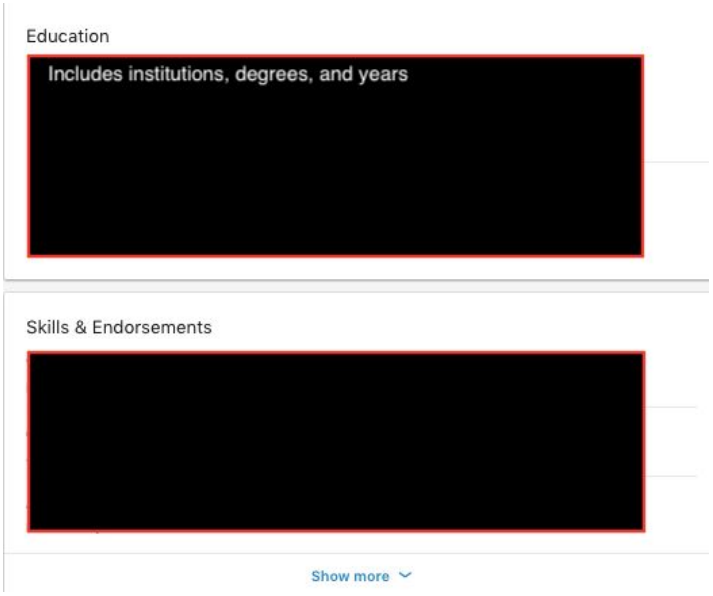
7. Synack, Inc. "Synack, Inc.; Patent Application Titled "Simulating a Bot-Net Spanning a Plurality of Geographic Regions" Published Online (USPTO 20160156657)." Computer Weekly News, Jun 22, 2016, pp. 1493. ProQuest, https://search.proquest.com/docview/1797352882?accountid=74409.

8. “Web Scraping” Techopedia.com, www.techopedia.com/definition/5212/web-scraping.

Appendix

A. Sample Profile Visited:





B. Experiment: Control Result

Time Started: 11:01 PM

Time Stopped: 11:37 PM

Time Elapsed: 36 minutes

Profile Visited: 311

Rate: 8.64 profiles/minute

C. Experiment: Timing Result

Time Started: 8:32 AM

Time Stopped: 12:55 PM

Time Elapsed: 4 hours 23 minutes

Profile Visited: 1,000

Rate: 3.80 profiles/minute

Time delay in the script between moving profiles: 4-9 seconds

D. Experiment: IP Addresses Rotation Result

Time Started: 1:40 PM

Time Stopped: 3:33 PM

Time Elapsed: 1 hour 53 minutes

Profile Visited: 133

Rate: 1.17 profiles/minute

IP addresses in rotation: 79

Profiles visited between each switch: 20

E. Experiment: IP Addresses Rotation's IP Addresses List

IP	PORT	IP	PORT	IP	PORT	IP	PORT
186.46.160.54	53281	177.72.228.138	53281	62.80.172.251	41258	41.169.56.80	53281
185.48.142.27	41258	202.91.69.4	53281	178.205.105.176	8080	43.239.75.54	8080
190.214.52.226	53281	36.67.29.235	53281	186.46.237.38	53281	93.80.50.75	8080
41.78.216.82	53281	177.180.152.94	53281	103.25.167.210	53281	95.0.176.198	8080
190.90.140.82	53281	47.75.48.149	80	41.161.63.58	8080	190.12.62.94	6666
190.214.16.230	53281	122.55.232.98	53281	36.65.152.198	53281	190.121.128.242	8181
79.142.202.109	8080	73.115.157.20	80	200.107.59.98	53281	185.16.102.186	41258
163.172.86.64	3128	176.118.131.136	53281	89.216.56.8	8080	115.77.191.180	53281
5.39.48.34	80	36.84.223.191	53281	190.121.231.243	53281	189.58.100.169	53281
195.138.93.1	8080	184.172.238.18	80	176.105.179.149	41258	153.146.163.197	8080
89.216.114.113	53281	189.112.92.139	53281	87.255.64.251	8080	190.152.182.150	53281
47.89.241.103	8080	212.118.37.103	8080	182.16.162.210	53281	46.0.198.9	3128
202.21.116.190	53281	46.10.157.115	8080	179.191.87.158	53281	195.66.157.49	41258
77.95.76.102	8080	213.33.224.82	8080	41.169.72.116	53281	46.37.192.29	8080
178.128.72.132	80	190.7.241.5	53281	82.208.111.196	8080	46.218.85.77	3129
63.249.67.70	53281	103.204.231.166	8080	190.12.48.158	52305	46.16.226.250	41258
85.200.245.220	8080	103.205.26.78	53281	181.113.130.150	53281	139.255.57.32	8080
41.33.84.1	53281	89.176.66.	8080	186.42.21	53281	46.181.42.	53281

36		134		4.186		242	
190.214.16.154	53281	96.9.79.140	55555	36.89.129.241	8080	200.54.194.12	53281
212.91.178.110	53281	182.48.78.122	53281	203.150.195.66	53281		

F. Experiment 2: IP Addresses Rotation Result

Time Started: 11:41 PM

Time Stopped: 6:03 PM

Time Elapsed: 6 hour 22 minutes

Profile Visited: 950

Rate: 2.49 profiles/minute

IP addresses in rotation: 79 (The same as the previous round)

Profiles visited between each switch: 10

G. Experiment: Control - Code Used

```
import static org.junit.Assert.*;


import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.util.ArrayList;

import java.util.concurrent.TimeUnit;
```

```
import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.Dimension;
import org.openqa.selenium.NoSuchElementException;
import org.openqa.selenium.Point;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;


public class Control {

	@Test
	public final void test() throws IOException, InterruptedException {
		System.setProperty("webdriver.chrome.driver", "chromedriver");
		// Creating chrome options instance:
		ChromeOptions options = new ChromeOptions();

		options.addArguments("user-data-dir=profile");

		// Initializing Chrome browser driver:
		WebDriver driver = new ChromeDriver(options);

		// increasing the dimension so next page can be clicked
```

```
driver.manage().window().setPosition(new Point(0, 0));
driver.manage().window().setSize(new Dimension(1440, 768));
driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
driver.manage().timeouts().pageLoadTimeout(5, TimeUnit.SECONDS);
driver.manage().timeouts().setScriptTimeout(5, TimeUnit.SECONDS);


BufferedReader br = new BufferedReader(new
FileReader("EE_DATA_COLLECTION.csv"));
String line = br.readLine();
ArrayList<String> profileURLs=new ArrayList<String>();
profileURLs.add(line);
while((line=br.readLine())!=null) profileURLs.add(line);
BufferedWriter bw=new BufferedWriter(new
FileWriter(“count_control.txt"));
for(int i=0;i<profileURLs.size();i++) {
driver.get(profileURLs.get(i));
Thread.sleep(200);
//The script will not collect any data but it will verify whether the page is loaded and
the information is found.
try {
//Verify information is found
driver.findElement(
By.xpath("//h1[starts-with(@class,'pv-top-card-section__name')]"));
```

```
                    String p="\n"+(i+1)+"";
                    System.out.println(p+" passed");
                    bw.write(p);
                    bw.flush();
                } catch(NoSuchElementException e) {
                    //It will be stopped when the information cannot be found
anymore i.e. when the bot is stopped.
                    System.out.println("broken");
                    break;
                }
            }
        }
}
```

H. Experiment: Timing - Code Used

```
import static org.junit.Assert.*;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Random;
```

```
import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.Dimension;
import org.openqa.selenium.NoSuchElementException;
import org.openqa.selenium.Point;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;

public class Timing {

	@Test
	public final void test() throws IOException, InterruptedException {
		System.setProperty("webdriver.chrome.driver", "chromedriver");
		// Creating chrome options instance:
		ChromeOptions options = new ChromeOptions();
options.addArguments("user-data-dir=profile");

		// Initializing Chrome browser driver:
		WebDriver driver = new ChromeDriver(options);

		// increasing the dimension so next page can be clicked
		driver.manage().window().setPosition(new Point(0, 0));
```

```
		driver.manage().window().setSize(new Dimension(1440, 768));


		BufferedReader br = new BufferedReader(new
FileReader("EE_DATA_COLLECTION.csv"));
		String line = br.readLine();
		ArrayList<String> profileURLs=new ArrayList<String>();
		profileURLs.add(line);
		while((line=br.readLine())!=null) profileURLs.add(line);
		Random gen=new Random();
		BufferedWriter bw=new BufferedWriter(new
FileWriter("count_timing.txt"));
		for(int i=0;i<profileURLs.size();i++) {
			driver.get(profileURLs.get(i));
			// TIME DELAY BELOW
			Thread.sleep(2000);
			Thread.sleep(2000+(gen.nextInt(5)*1000));
			//The script will not collect any data but it will verify whether the
page is loaded and the information is found.
			try {
				//Verify information is found
				driver.findElement(
By.xpath("//h1[starts-with(@class,'pv-top-card-section__name')]"));
				String p="\n"+(i+1)+"";
				System.out.println(p+" passed");
```

```
                    bw.write(p);
                    bw.flush();
                } catch(NoSuchElementException e) {
                    //It will be stopped when the information cannot be found
anymore i.e. when the bot is stopped.
                    System.out.println("broken at "+(i+1));
                    break;
                }
            }
        }
}
```

I. Experiment: IP Rotation - Code Used

```
import static org.junit.Assert.*;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Random;
import java.util.concurrent.TimeUnit;
```

```
import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.Dimension;
import org.openqa.selenium.NoSuchElementException;
import org.openqa.selenium.Point;
import org.openqa.selenium.TimeoutException;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;

public class IPRotation {
	HashMap<String,String> proxyServers=new HashMap<String,String>();
	ArrayList<String> ipList=new ArrayList<String>();

	@Test
	public final void test() throws InterruptedException, IOException {
		//Sort the proxies into a list of IP addresses and ports.
		BufferedReader br = new BufferedReader(new FileReader("IP_ROTATION
- Sheet2.csv"));
		String line=br.readLine();
		String key=line.split(",")[0];
		String value=line.split(",")[1];
		proxyServers.put(key, value);
```

```
while((line=br.readLine())!=null) {


        key=line.split(",")[0];
        value=line.split(",")[1];
        ipList.add(key);
        proxyServers.put(key, value);
}
br.close();
System.setProperty("webdriver.chrome.driver", "chromedriver");


br = new BufferedReader(new
FileReader("EE_DATA_COLLECTION.csv"));
line = br.readLine();
ArrayList<String> profileURLs=new ArrayList<String>();
profileURLs.add(line);
while((line=br.readLine())!=null) profileURLs.add(line);
Random gen=new Random();
BufferedWriter bw=new BufferedWriter(new
FileWriter("count_iprotation.txt"));


WebDriver driver = null;
int beg=0;
for(int i=beg;i<profileURLs.size();i++) {
        if(i==beg&&driver==null) {
```

```
                    try {
                        //This is setting the timeout for slower IP so that it
can automatically change to another IP if that one timeout.
                        System.out.println("1st: run at i:"+i);
                        driver=ipRotation();
                        driver.manage().timeouts().pageLoadTimeout(20,
TimeUnit.SECONDS);
                        driver.manage().timeouts().setScriptTimeout(20,
TimeUnit.SECONDS);
                    } catch(Exception e) {
                        System.out.println("5th: run at i:"+i);
                        driver.close();
                        i--;
                        continue;
                    }
                }
                //This if statement is to rotate the IP at every 10 profiles hit
                if(i!=beg&&i%10==0) {
                    System.out.println("2nd: run at i:"+i);
                    driver.close();
                    driver=ipRotation();
                    driver.manage().timeouts().pageLoadTimeout(20,
TimeUnit.SECONDS);
```

```
                    driver.manage().timeouts().setScriptTimeout(20,
TimeUnit.SECONDS);
                }
                try {
                    System.out.println("3rd: run at i:"+i);
                    driver.get(profileURLs.get(i));
                    Thread.sleep(1000);
                    try{

driver.findElement(By.xpath("//div[@id='main-frame-error']"));
                        throw new Exception();
                    } catch (NoSuchElementException e) {
                        try {
                            driver.findElement(

By.xpath("//h1[starts-with(@class,'pv-top-card-section__name')]"));
                                String p="\n"+(i+1)+"";
                                System.out.println(p+" passed");
                                bw.write(p);
                                bw.flush();
                        } catch(NoSuchElementException ne) {
                            System.out.println("broken at "+(i+1));
                            break;
                        }
```

```
                        }
                    } catch(Exception e) {
                        e.printStackTrace();
                        System.out.println("4th: run at i:"+i);
                        driver.close();
                        driver=ipRotation();
                        driver.manage().timeouts().pageLoadTimeout(20,
TimeUnit.SECONDS);
                        driver.manage().timeouts().setScriptTimeout(20,
TimeUnit.SECONDS);
                        i--;
                        continue;
                    }
                }
            }
            /**
This method creates a new webdriver with a different IP configuration that is chosen at
random.
*/
            public WebDriver ipRotation() {
                ChromeOptions options = new ChromeOptions();
                Random gen= new Random();
options.addArguments("user-data-dir=profile");
                String ip=ipList.get(gen.nextInt(proxyServers.size()-1));
```

```
            options.addArguments("--proxy-server="+ip+":"+proxyServers.get(ip));
            System.out.println("--proxy-server="+ip+":"+proxyServers.get(ip));
            return new ChromeDriver(options);

        }
}
```