A Method to Secure IoT Devices Against Botnet Attacks

Trusit Shah(✉) and Subbarayan Venkatesan(✉)

The University of Texas at Dallas, Richardson, TX 75080, USA
{trusit.shah, venky}@utdallas.edu

Abstract. An unsecured or weak authentication system between an IoT device and a user provides opportunities to attackers to manipulate and use the IoT device as botnet. The proliferation of IoT devices with an unsecured/weak authentication mechanism has increased the threat of using a huge number of IoT devices as botnets for large-scale DDoS attacks. Default credential pairs (like 'root-root' or 'admin-admin') for the Telnet or SSH connections are still part of a large group of IoT products, and many malwares have exploited this vulnerability to capture a large number of IoT devices and use them as botnets. In the recent past, Mirai malware had infected roughly a million IoT devices at its peak by brute-forcing just 62 pairs of default credentials. In this paper, we present a concept called 'login puzzle' to prevent capture of IoT devices in a large scale. Login puzzle is a variant of client puzzle, which presents a puzzle to the remote device during the login process to prevent unrestricted log-in attempts. Login puzzle is a set of multiple mini puzzles with a variable complexity, which the remote device is required to solve before logging into any IoT device. Every unsuccessful log-in attempt increases the complexity of solving the login puzzle for the next attempt. In this paper, we have introduced a novel mechanism to change the complexity of puzzle after every unsuccessful login attempt. If each IoT device had used login puzzle, Mirai attack would have required almost two months to acquire devices, while it acquired them in 20 h.

Keywords: IoT security · DDoS attacks · Client puzzles · Botnet

1 Introduction

In the last decade, the rapid growth of IoT products in the market have connected billions of small-scale devices to the internet. To survive in such a fast-growing market, developers often neglects security aspects and release vulnerable products in the market. The attackers use such vulnerabilities to perform large-scale DDoS attacks and steal personal information of the user. Mirai is a well-known DDoS attack, which represents the first type of attack using vulnerable IoT devices [21]. A casino at Las Vegas, which lost some of its critical information due to a cyber-attack enabled by a vulnerable internet enabled fish tank[1], is an example of another attack.

[1] https://www.businessinsider.com/hackers-stole-a-casinos-database-through-a-thermometer-in-the-lobby-fish-tank-2018-4.

IoT devices with either default or easy to guess credentials allow the attackers to access these devices remotely with root permissions. Attacker use a pre-defined set of credentials to log in open Telnet or SSH ports. Most of the time, around 100 credentials are sufficient to capture a massive number of IoT devices and only a few seconds are needed to brute-force these credentials. Thus, a self-replicating attacker script can brute-force a few hundred thousand devices in less than a week. The ability to brute force devices without any restriction enables attackers to capture a large number of IoT botnets.

A potential solution for this problem is to restrict a single IP from trying a large number of login attempts. This solution is vulnerable to IP address hopping [22]: an attacker can change the IP address after every unsuccessful login. Another solution is to implement a blocking mechanism, which allows a small number of login attempts before it blocks further login attempts. This method certainly reduces the number of devices being captured, but the attacker script can get an instantaneous access till it reaches the blocking number. A selective approach with different login credentials for different kind of devices can capture a large number of devices. A timeout-based method is also useful in preventing multiple abusive login attempts. The device enters into a timeout mode after every unsuccessful attempt and every unsuccessful attempt increases the timeout period. The downside of the timeout method is, the attacker script can capture other devices in parallel.

In this paper, we present a method called ‘login puzzle’ to prevent the unrestricted login attempts performed by a malicious script. Login puzzle is a collection of mini puzzles, which requires a fixed amount of time and computation power to solve. When a malicious script tries to brute-force an IoT device, it needs to allocate a fixed amount of time and resource to solve the login puzzle, and every unsuccessful attempt increases the complexity of the login puzzle, hence increasing the time and computation power required to solve the next login puzzle. At a certain point, the login puzzle is hard enough that it is not practical for the script to solve. This method not only limits the number of attempts but also slows down the rate of attempts. It is a combination of both blocking and timeout method and provides better performance than these individual methods.

Another approach is to send a single puzzle to the login entity instead of multiple mini puzzles, it leads to a problem in changing the complexity of the puzzle. If we change the complexity every time a wrong attempt happens, an adversary can request multiple parallel sessions with a same complexity. On the other side, a malicious script can perform multiple attempts just to increase the complexity to block a benign user. Use of login puzzle solves these problems.

Login entity must solve all the mini puzzles of the login puzzle to get the login access to the IoT device. All of these puzzles are sent sequentially; login entity will get the next mini puzzle, once it solves the previous one. In case of parallel requests, once a request solves all the puzzle and attempts an unsuccessful login attempt, number of mini puzzles for the other parallel requests will increase, consequently increasing the complexity of the subsequent login attempts. If an adversary just requests multiple sessions without actually solving mini puzzles, the complexity of the puzzle won’t increase, hence benign user won’t be block by any malicious script.

We have calculated the improvement achieved by using login puzzle. We have measured this scheme with Mirai and computed the amount of additional time and resource required to capture the IoT devices. The main contribution of our work includes:

- A practical mechanism to extend the capture time of an IoT device as botnet
- Introduction to the login puzzle and provide calculations for its performance efficiency
- A novel complexity changing mechanism, which ensures constant resource allocation from the client side and provides minimum penalty to benign users

This paper is organized in the following way. We discuss about previous work in Sect. 2. Section 3 describes our security mechanism followed by Sect. 4 explaining the performance analysis and comparison with other authentication mechanisms.

1.1 DDoS Attack Overview

A denial of service is characterized as ‘attempt to prevent the use of a network service to the legitimate users. In a DDoS, an attacker uses multiple devices to achieve denial of service. As more IoT devices are being manufactured with limited security considerations, new methods are being introduced to use IoT devices as bots for DDoS attacks. A study by Dragan Peraković et al. shows that the number of DDoS attacks has increased with the growth in numbers of IoT devices [20].

A DDoS attack using IoT botnets typically consists of three phases: host scanning, device acquisition and denial of service attack. In the host scanning phase, the attacker scans for the IoT devices with open Telnet or SSH ports. After discovering a vulnerable device, the attacker script tries to log into the device using a set of pre-defined credentials. If the attacker gets the access to the IoT device using one of the pre-defined credentials, the attacker executes the second phase. In the device acquisition phase, a self-replicating script is injected inside the IoT device by the attacker.

The self-replicating script scans through the network to infect more IoT device using the same host scanning mechanism and injects the same script into the newly infected device. Meanwhile, this script continuously communicates with the attacking server and waits for the commands to perform a DDoS attack. After acquiring enough IoT devices, the attacking server commands all the infected IoT devices to attack a specific service. A large volume of internet traffic is generated for the victim with the potential to disrupt the service.

Figure 1 shows an underlying architecture for the DDoS attack using IoT botnets. The four main components for a DDoS attack using IoT devices are an attacker, a malware server, a command and control (CnC) server and IoT botnets. The attacker initiates the host scanning phase and acquires an ample amount of IoT devices. Those discovered IoT devices download the self-replicating script from the malware server. In some cases, the malware server is also known as report server. The script registers the IoT device to the CnC server and waits for the command from the CnC server. Finally, after acquiring adequate bots, the attacker commands the CnC server to send a request to botnets to attack the victim service.

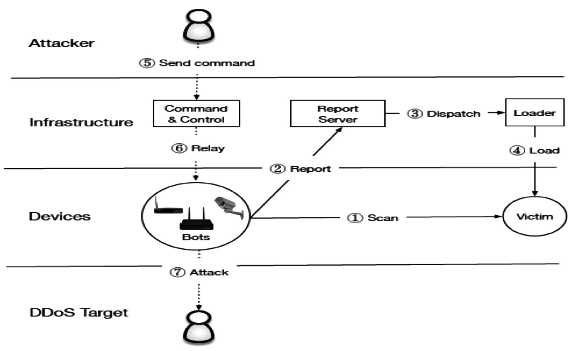

Fig. 1. Mirai architecture

2 Previous Work

Researchers have proposed few methods to use digital problems to avoid DDoS attacks. Most of these works secure server or web services from a DDoS attack. The very first work was introduced by Aura et al. as client puzzle [1]. In this paper, the authors introduced the concept of resource commitment at the client side. The client solves a puzzle given by the server with the authentication, before the server allocates any heavy resource for the client. While this method is very effective in stopping the DDoS attack at server side, it doesn't provide significant improvement at the host scanning phase because it uses problems of same complexity every time. If the complexity is too high, the benign user suffers from getting access of the IoT device and if the complexity is too low, the attacker can solve it easily and can brute force credentials to get the access of the IoT device. A varying complexity method helps benign user to access IoT device with low computation and restricts an attacker from brute forcing the credentials on the IoT devices. Michalas et al. presented a similar concept in client puzzle to prevent DoS attack in an ad-hoc network [3]. Another client puzzle was introduced by Abliz et al. which protects a server from DDoS attack based on traffic created by the clients [14]. Suriadi and et al. have presented a similar approach to prevent a DoS attack for web services [2].

Many approaches for creating client puzzle have been proposed by researchers. Chen et al. have proposed a generic way to represent a client puzzle [4]. In the paper authors have defined client puzzle as a tuple algorithm, which provides a formal setup to generate puzzle, solve puzzle and formal verification of the puzzle. Groza et al. have presented a formal verification for the puzzle hardness and bounds [5]. Jing et al. have used repeated squaring and hash reversal client-puzzle with the leaky bucket algorithm [6].

Other than client puzzle, various security strategies at different stages of DDoS attack have been proposed by researchers to prevent the DDoS attacks [12, 13]. At the discovery stage, a DDoS attack can be prevented by closing the unused ports. At the network level, construction of a firewall restricts the spread of malware. IoT devices are made for performing specific tasks and require communication with specific domains only. Continuous monitoring of running processes and network traffic of an IoT device provides enhanced security. This method constrains device activities and network traffic and causes false alarms when benign but unknown activities are happening.

Various methods have been introduced by different researchers to prevent the DDoS attack on the victim side. There are methods called intrusion detection to detect a potential DDoS attack. There are other signature-based methods to detect the intrusion [16, 19]. Recently, machine learning based detection methods have been proposed to check the spread of the DDoS attack [17, 18].

3 Our Methodology

We present a mechanism to provide security against the DDoS attack at the device scan phase. The malware could acquire a large number of IoT devices because there is no restriction on trying multiple guessed credentials using a brute force attack. Our method called 'Login puzzle', prevents an attacker from brute-forcing credentials. Login puzzle provides an extra layer of a challenge-response mechanism by generating NP-hard puzzles and asks the login entity to solve it before being logged in. The difficulty of question gets harder with the number of false attempts, hence stopping the intruder from using the brute-force method to capture an IoT device.

The login attempts to an IoT device can be divided into three main categories: log in using user interface having username-password as security credentials, log in using script having username-password as security credentials and log in using script using the digital certificate as security credentials. The third category is not vulnerable to brute force attack as it is computationally impossible to create a brute-force attack on a digital certificate, hence no additional security required for this category. For the first and second categories, an additional mechanism is required to prevent the unrestricted brute-force login attempt. In our methodology, we use regular captcha and login puzzle for the first and second category, respectively.

Both first and second category use: username-password as security credential: To differentiate them, a flag bit indicating the type is sent along with the login request. In this paper, we assume that a computer program is not capable of solving the regular captcha used for login. Hence, if a script tries to fake itself as a human to avoid the login puzzle, it won't be able to login to the system.

3.1 Login Puzzle

A login puzzle is a collection of mini puzzles with a varying complexity, which require a fixed amount of time and resource to solve depending on the complexity. A login puzzle is defined as P (n), where n defines the complexity of the puzzle. Complexity of a login puzzle is addition of all of its mini puzzles. Mini puzzle is expressed as M (x, n), where x is the set of puzzle parameters, n is the complexity of the mini puzzle. A login puzzle can have mini puzzles with different complexities. The time required to solve the login puzzle P and mini puzzle M at complexity n is t_p (n) and t_m (n) respectively.

A login puzzle P has following properties:

- Based on the complexity n, it requires a fixed amount of time and computation power to solve.
- For a puzzle function P, if $n_1 > n_2$ then $t_p(n_1) \geq t_p(n_2)$.
- For every device with a finite computational capacity C, $\exists n$ at which computational capacity C requires unreasonably more time and resource to solve the login puzzle. This value of complexity is called critical complexity for the computational capacity C. The time require to solve it called critical time $t_c(n)$.

Complexity Increment of the Login Puzzle

After every unsuccessful attempt, the complexity of the login puzzle is increased. There are two ways the complexity can be increased: linear increment and exponential increment. In the linear increment, the complexity of login puzzle is increased exponentially. This can be achieved by either increasing the number of mini puzzles, without changing the complexity of mini puzzles or increasing the complexity of only one mini puzzle. In exponential increment, the complexity of each mini puzzle is incremented exponentially, consequently increasing the overall complexity exponentially. In general, after every unsuccessful attempt complexity of the puzzle is incremented exponentially and in case client stops attempting the puzzle, the complexity is increased linearly.

Login Puzzle in Login Mechanism

At each login attempt by a remote device, a login puzzle is sent to the device. The initial complexity of the login puzzle is 1, and each unsuccessful login attempt to the IoT device exponentially increases the complexity. As per the third property of the login puzzle, at some point, the login attempt requires immense resources, which cannot be handled by the remote login-device. The complexity increment mechanism is as follow:

- Normal attempt: When a remote login entity solves all the puzzles, it is considered as a normal attempt. Once the remote device provides solutions for all the puzzles, it has access to the login and if it performs an unsuccessful login attempt, the complexity of login puzzle is increased exponentially. On the other side, for successful login attempt the complexity will reset to 1. The first part of Fig. 2 provides illustration to this case.
- Midway give up attempt: When a login entity stops solving mini puzzles in the middle and doesn't arrive at login attempt, the login puzzle complexity is increased linearly. This can be performed by either increasing number of mini puzzles or increasing complexity of single mini puzzle. The second part of Fig. 2 represents this case.
- Parallel login attempts: This situation arises when a login entity requests for the login access during an existing login entity solving login puzzle. At the initial stage the new login entity will have same complexity as the previous one. In case the previous one does the unsuccessful login attempt, the complexity of login puzzle is doubled. Figure 3 shows the message exchange representation of this case.

The proposed method increases the complexity of the login puzzle irrespective of the IP address of the requestor. If an attacker script changes its IP address at each login attempt, it still needs to solve login puzzle with a higher complexity after every unsuccessful attempt. Thus, login puzzle is not vulnerable to IP hopping attack.

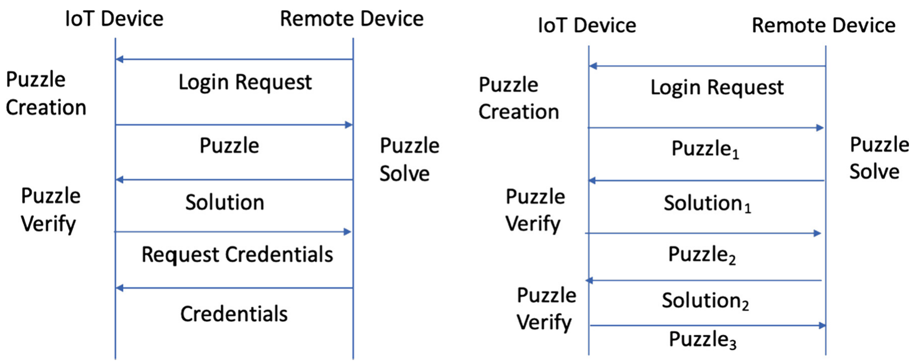

Fig. 2. The first image represents the normal login request, where the puzzle is provided to the remote device and after providing the solution remote device is able to login. The second image represents the case when remote device stops solving the puzzle after certain login attempts.

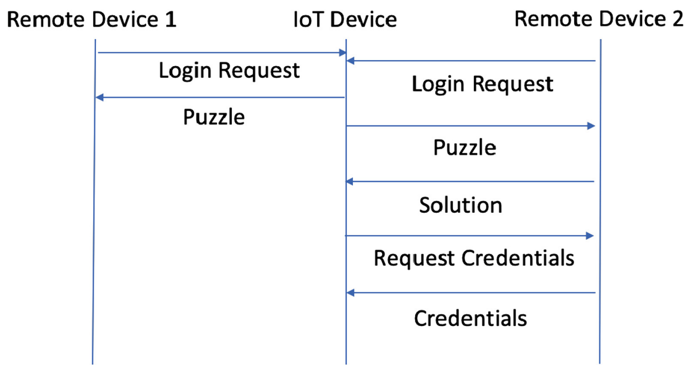

Fig. 3. This figure shows the parallel request case, in this particular case, if the remote device 2 provides wrong credentials, the complexity of remote device 1 will increase exponentially.

3.2 Resource and Timing Analysis

The resources and time required to solve a login puzzle is addition of time and resource required to solve all mini puzzles, hence we consider complexity of login puzzle for the resource and timing analysis. Every device needs to allocate a fixed computational capacity to solve a login puzzle. The required computational capacity varies based on the type and complexity of the login puzzle and, as per the 3rd property of the login puzzle, after m tries, the login puzzle reaches a certain complexity that it is not feasible for the device to solve it. Thus, a device with computational capacity C is limited to only m tries. All these m tries are not instantaneous; for every try, the device needs to solve a login puzzle, which requires a fixed amount of time to solve. In this section, we will measure the effectiveness of login puzzle on a botnet capture script, which uses x number of credentials to capture an IoT device.

The time required to try one credential without login puzzle be t_d. This time includes propagation and transmission delay caused by the network and processing delay due to login procedure; it is few milliseconds in duration. If we consider a uniform distribution of the credentials, it requires on an average *x*/2 attempts for an attacker script to capture an IoT device as a botnet if the device's credential is from one of the x credentials. The device needs to perform x failed attempts if the device's credential is not from the list. Assuming that the ratio of number of devices whose credentials are from x credentials to the total number of devices scanned is p, the average time required to capture one device is:

$$t_d * \frac{x}{2} * p + t_d * x * (1 - p)$$

This can be further simplified as:

$$x * t_d\left(1 - \frac{p}{2}\right) \tag{1}$$

Here,

$$p = \frac{\textit{number of devices whose credentials are from x credentials}}{\textit{total number of devices scanned}}$$

If we consider an exponentially increasing login puzzle, where the complexity of $t_p(n)$ increases 'a' times with the increment of complexity of login puzzle by 1. Considering, the time required to solve the login puzzle at complexity 1 is t_c. As per the 3rd property of login puzzle, after m attempts the problem becomes incomputable for the device. The total time required to solve login puzzle for first m attempts is,

$$t_m = \sum_{i=0}^{m-1} t_c * a^i$$

If the device's credential is from one of the pre-defined credentials, it takes x/2 attempts to capture the device, otherwise the attacker script tries x credentials. Considering the ratio of number of devices whose credentials are from x credentials to the total number of devices scanned is p and there are k mini puzzles in one login puzzle, average time required by a device to capture one device after using login puzzle is,

$$p * \left(t_{x/2} + \frac{x * k * t_d}{2}\right) + (1 - p) * (t_x + x * k * t_d)$$

Where,

$$t_{x/2} = \sum_{i=0}^{x/2-1} t_c * a^i \text{ and } t_x = \sum_{i=0}^{x-1} t_c * a^i$$

For the value of $x/2 < m \leq x$, the t_x term in above equation becomes t_m, because after m attempts the login puzzle is incomputable.

$$p * (t_{\frac{x}{2}} + \frac{x * t_d}{2}) + (1 - p) * (t_m + x * t_d)$$

For simplicity, if we consider the time complexity of login puzzle increases with a factor of 2. The above values can be simplified as below:

$$t_m = t_c * (2^m - 1)$$

$$t_{x/2} = t_c * \left(2^{x/2} - 1\right)$$

If we assume the factor of $\mathrm{t_c}$ to $\mathrm{t_d}$ is f.
The average required time to capture a device using login puzzle is,

$$t_c * \left(2^{x/2} - 1 + \frac{\mathrm{x} * \mathrm{k}}{2 * f}\right) * p + t_c * \left(2^m - 1 + \frac{\mathrm{m} * \mathrm{k}}{f}\right) * (1 - p) \tag{2}$$

If $m \leq x/2$, $x/2$ in the first term of the above equation becomes t_m and the average time required to capture a device using login puzzle is,

$$t_c * \left(2^m - 1 + \frac{\mathrm{m} * \mathrm{k}}{f}\right) * p + t_c * \left(2^m - 1 + \frac{\mathrm{m} * \mathrm{k}}{f}\right) * (1 - p)$$

The above equation can be simplified to,

$$t_c * \left(2^m - 1 + \frac{\mathrm{m} * \mathrm{k}}{f}\right) \tag{3}$$

Also, the probability of the device getting captured for all above cases is,

$$\frac{p * m}{x}$$

4 Evaluation and Performance Measure

We evaluated our approach on the different environments with different login puzzle conditions for Mirai DDoS attack. We have used three different environments with completely different capabilities to measure the effectiveness of our approach. The first environment is a small scale IoT device and we have used Raspberry Pi 3. The second environment is a MacBook Pro laptop with quad-core Intel i7 processor and 16 GB of RAM. The final environment is a cloud server hosted on AWS cloud service. The AWS instance contains 36 cores with 60 GB RAM. The first environment represents the normal spread of botnets, where infected IoT devices scan other IoT devices and solve the login puzzles themselves. The second and third environments provide the insight for the case, when IoT devices don't solve the login puzzles but send it over to the server to solve them. The primary difference between the second and third environment is the processing power of the server. The login puzzle we are using for the evaluation is as follow:

D (x, n): find a number m, for which SHA512(x $\|$ m) has all last n bits are 0.

Table 1 shows the typical time required for all three environments to solve the login puzzle for complexity n.

4.1 Mirai Evaluation

Mirai was a well-known malware, which acquired more than 600k devices during its lifespan. Mirai infected 11k hosts in first 10 min of bootstrap phase and 64,500 devices in 20 h [8]. The bootstrap phase lasted for around two hours and initial 834 devices started scanning. MIRAI had comparatively a smaller number of credentials compare to other massive DDoS attacks Blaster [9] and Code Red [10]. This contributed to a slow doubling rate 75 min compare to Code Red (37 min) and Blaster (9 min) [8].

We use the following method to simplify the performance evaluation of login puzzle. First, we calculate the average time required to capture a single device with and without using login puzzle. Using these values, we measure the performance enhancement achieved by login puzzle. Finally, we multiply the performance enhancement ratio and the actual time taken by Mirai to spread over the network and compute the time required to spread Mirai when all the IoT devices are using login puzzle. We also made another assumption that the number of mini puzzles, k = 16.

Table 1. Time required to solve login puzzle for different type of devices

Environment	n	t(n)
Raspberry Pi	16	~10 s
	24	200 s
	32	Incomputable
Macbook Pro	16	Less than a second
	24	20 s
	32	~85 min
	40	Incomputable
AWS Server	16	Milliseconds
	24	Less than a second
	32	~1 min
	40	~250 min
	48	Incomputable

Number of credentials used in Mirai was 62 (x is 62 in the Eq. 1). Thus, the average time to capture one device is:

$$62 * t_d\left(1 - \frac{p}{2}\right) = 6.2 * \left(1 - \frac{p}{2}\right) \text{ in seconds} \tag{4}$$

We have tested the values of t_c and t_d for raspberry pi, and the values are 10 µs and 100 ms respectively. The factor $f = \frac{t_c}{t_d} = 10^{-4}$.

To calculate value of m, we assume that the attacker script stops solving login puzzle, once the time taken for solving captcha reaches 100 s. value of $m = \log_2(\frac{100\,\mathrm{S}}{t_c})$ $= 23.26 \sim 24$.

As value of the $x/2$ is greater than m, the average time required to capture one IoT device is calculated using Eq. 3,

$$t_c * \left(2^m - 1 + \frac{\mathrm{m} * \mathrm{k}}{f}\right) = 10^{-5} * \left(2^{24} - 1 + 24 * 16 * 10^4\right) \sim 200\,\mathrm{s}$$

Also due to limitation of only m tries can be performed, not all devices with the default credentials can be captured.

The probability of device being captured is: $\frac{m}{x} = \frac{24}{62} = 0.39$.

Thus, with the login puzzle the number of devices captured by Mirai malware is: 64,500 * 0.39 = 24967.

Time required to capture one device using login puzzle with probability of device being captured is 1:

$$t_{(p=1)} = \frac{\textit{time required to capture one device using login puzzle}}{\textit{probability of device being captured}} = 512.82\ \mathrm{s}$$

In conclusion, use of login puzzle increases the time required to capture a device by a factor of,

$$\frac{t_{(p=1)}}{\textit{time required to capture single device without login puzzle}} = 82.72$$

Here, the time required to capture single device without digital captcha is calculated from Eq. 4 by setting p = 0, which represents the maximum value for the equation. This implies that the total time required to cover the spread of 64,500 devices with login puzzle will be = 20 h * 82.72 ∼ 69 days.

Similar calculation can be performed on 2nd and 3rd type of environment and we can find time required to capture 64,500 devices if all the devices were using login puzzle and the login puzzles are solved on cloud with capabilities mentioned in Table 1. Table 2 mentions these times.

Table 2. Time required to capture 64,500 devices with login puzzle using different environment

Environment	Time required
Raspberry pi	59 days
Macbook pro	42 days
AWS Server (96 GB RAM, 64 cores)	26 days

4.2 Comparison with Blocking Method

In the blocking method, after a certain amount of attempts the IoT device blocks further login attempts until the user approves those attempts. At first, it looks like a blocking method with a small number of login attempts performs better than the login puzzle. In fact, based on our calculations, our method performs 70 times better than the blocking mechanism.

Let's assume that the IoT device blocks login attempts after b attempts. The average time required to capture one device is,

$$t_d * \frac{b}{2} * \left(\frac{b * p}{x}\right) + t_d * b * \left(1 - \frac{b * p}{x}\right)$$

The first term in the equation represents that one of the randomly guessed credentials in first b tries is the credential of the device. On an average it requires b/2 tries to achieve it with the probability b/x. This probability is finally multiplied with probability p, which represents the number of devices with the credentials used by the malware. The second term represents the scenario when the malware is not able to guess the correct credentials in first b login attempts. The above equation can be further simply to 300–7.25p milli-seconds for the value of b = 3 and x = 62. The ratio of number of devices captured to total number of devices with given credential is $\frac{b}{x} = \frac{3}{62} = 0.05$.

The value of p varies from 0 to 1 doesn't have a big impact on time required to capture the device. Thus, we can assume that the time required to capture one device is 300 ms. So, the average time to capture one device is = 300/0.05 ms = 6 s. The average time to capture one device using login puzzle is = 170/0.39 s = 436.34 s. This implies that the login puzzle method is roughly 70 times better than blocking method.

4.3 Comparison with Existing Client Puzzles

In this section, we will compare our approach with existing client puzzles and showcase its suitability to prevent the IoT device capture on a large scale. We compare our scheme with simple client puzzle and adaptive client puzzle. A simple client puzzle is the most basic form of client puzzle and adaptive puzzle is a type of puzzle which changes its complexity based on the situation. We used three parameters to compare these client puzzles: login delay, resource assurance, benign penalty.

Table 3 shows a comparison between these three client puzzles. As normal client puzzle doesn't change its complexity over the time, it is not providing any significant delay in login process. At the same time, it is not creating any high waiting time for the benign user, so benign penalty is minimum in this case. For the case of adaptive puzzle, the login delay can go really high, as the complexity go higher. As discussed in the previous sections, a malicious user can increase the complexity for the benign user by sending multiple fake requests. Just like adaptive puzzle, login puzzle has high login delay and increases with every wrong login attempt. In login puzzle, as discussed in the previous sections, puzzle ensures the resource assurance and has different policies to increase complexity for benign and malicious request, hence benign penalty is minimal in this case.

Table 3. Client puzzles comparisons

	Client puzzle	Adaptive puzzle	Login puzzle
Login delay	Very low	High	High
Resource assurance	No	No	Yes
Benign penalty	low	Very high	Minimal

5 Discussion and Future Work

Unlike normal adaptive puzzle, login puzzle doesn't have problem of benign penalty. Login puzzle provides minimal penalties to benign users. These penalties can be further reduced, if someway a benign request is differentiated from malicious request. As a future work to reduce the benign penalties, a method to provide the login puzzle only to the suspicious scripts can be implemented. There are researches explain multiple ways to differentiate a bot attack from a regular user login [7, 11, 15].

This paper assumes the distribution of the passwords is uniform, but in a real-world scenario, it is not the case. A more detailed study with a non-uniform password distribution provides a more accurate performance analysis of the login puzzle.

6 Conclusion

A large number of IoT devices with default credentials pairs or easy to guess passwords has increased the threat of Acquisition of IoT devices as botnets. In the recent past, Mirai malware has exploited this vulnerability and infected around 600 k IoT devices.

In this paper, we demonstrated how adding an extra layer of login puzzle during an authentication phase can reduce the number of IoT devices being captured as botnets. A login puzzle is an NP-hard problem, whose complexity increases at every unsuccessful login attempt. In this paper, we used 'searching a bitstream of zeros at the end of the inverse of a one-way hash function' as the login puzzle problem.

However, this method does not prevent capturing of every IoT device, it reduces the number of devices being captured and increases the time and resource required to capture those devices. Based on the calculations in Sect. 4, the additional time required to capture IoT devices is enough to detect the spread of the malware through the network traffic and prevent it from acquiring more devices.

Acknowledgements. This research was conducted at Distributed Systems Lab of The University of Texas at Dallas. We would like to thank all the anonymous reviewers for their valuable feedback to improve this paper.

References

1. Aura, T., Nikander, P., Leiwo, J.: DOS-resistant authentication with client puzzles. In: Christianson, B., Malcolm, J.A., Crispo, B., Roe, M. (eds.) Security Protocols 2000. LNCS, vol. 2133, pp. 170–177. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44810-1_22
2. Suriadi, S., Stebila, D., Clark, A., Liu, H.: Defending web services against denial of service attacks using client puzzles. In: 2011 IEEE 9th International Conference on Web Services 2011, pp. 25–32. IEEE Computer Society (2011)
3. Michalas, A., Komninos, N., Prasad, N.R., Oleshchuk, V.A.: New client puzzle approach for dos resistance in ad hoc networks. In: 2010 IEEE International Conference on Information Theory and Information Security (ICITIS), pp. 568–573. IEEE, December 2010
4. Chen, L., Morrissey, P., Smart, N.P., Warinschi, B.: Security notions and generic constructions for client puzzles. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 505–523. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_30
5. Groza, B., Warinschi, B.: Revisiting difficulty notions for client puzzles and DoS resilience. In: Gollmann, D., Freiling, F.C. (eds.) ISC 2012. LNCS, vol. 7483, pp. 39–54. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33383-5_3
6. Jing, Y.K., Ming, J.T.C., Niyato, D.: Rate limiting client puzzle schemes for denial-of-service mitigation. In: 2013 IEEE Wireless Communications and Networking Conference (WCNC), pp. 1848–1853 (2013)
7. Walid, A., Mostafa, A., Salama, M.: MalNoD: malicous node discovery in internet-of-things through fingerprints. In: 2017 European Conference on Electrical Engineering and Computer Science (EECS). IEEE (2017)
8. Antonakakis, M., et al.: Understanding the Mirai botnet. In: USENIX Security Symposium (2017)
9. Bailey, M., Cooke, E., Jahanian, F., Watson, D.: The blaster worm: then and now. IEEE Secur. Priv. **3**, 26–31 (2005)
10. Moore, D., Shannon, C., Claffy, K.: Code-red: a case study on the spread and victims of an internet worm. In: 2nd ACM Internet Measurement Workshop (2002)
11. Binkley, J.R., Singh, S.: An algorithm for anomaly-based botnet detection. SRUTI **6**, 7 (2006)
12. Mirkovic, J., Dietrich, S., et al.: Internet denial of service: attack and defense mechanisms. University of Pittsburgh (2005)
13. Jiang, L., et al.: Analysis and comparison of the network security protocol with DoS/DDoS attack resistance performance. In: 2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conference on Embedded Software and Systems (ICESS). IEEE (2015)
14. Abliz, M., Znati, T.F.: Defeating DDoS using productive puzzles. In: 2015 International Conference on Information Systems Security and Privacy (ICISSP), pp. 114–123. IEEE, February 2015

15. Gu, G., et al.: Botminer: clustering analysis of network traffic for protocol-and structure-independent botnet detection, p. 139 (2008)
16. Gu, G., Porras, P., Yegneswaran, V., Fong, M., Lee, W.: BotHunter: detecting malware infection through ids-driven dialog correlation. In: Proceedings of the 16th USENIX Security Symposium (Security 2007) (2007)
17. Saad, S., et al.: Detecting P2P botnets through network behavior analysis and machine learning. In: 2011 Ninth Annual International Conference on Privacy, Security and Trust (PST). IEEE (2011)
18. Ranjan, S., Robinson, J., Chen, F.: Machine learning based botnet detection using real-time connectivity graph based traffic features. U.S. Patent No. 8,762,298, 24 June 2014
19. Masud, M.M., Al-khateeb, T., Khan, L., Thuraisingham, B., Hamlen, K.W.: Flow-based identification of Botnet traffic by mining multiple. In: Proceedings of International Conference on Distributed Framework & Application (2008)
20. Peraković, D., et al.: Analysis of the IoT impact on volume of DDoS attacks. In: XXXIII Simpozijum o novim tehnologijama u poštanskom i telekomunikacionom saobraćaju – PosTel 2015, Beograd, 1. i 2. December 2015
21. Kolias, C., Kambourakis, G., Stavrou, A., Voas, J.: DDoS in the IoT: Mirai and other botnets. Computer **50**(7), 80–84 (2017)
22. Sifalakis, M., Schmid, S., Hutchison, D.: Network address hopping: a mechanism to enhance data protection for packet communications. In: IEEE International Conference on Communications, ICC 2005, pp. 1518–1523 (2005)