Sherlock: A Browser Fingerprint-based Authentication System

Sullam Jeoung[1], Chunghun Kim[2], Myung-jong Kim[3], Han Park[4], JooSeok Song[5]

***Abstract*— Sherlock is a novel user authentication system. Sherlock allows users to log in to websites without typing their passwords. Sherlock utilizes the browser fingerprinting technique implemented by fingerprintjs2 to identify users' devices. Sherlock's benefits have been tested and backed up by our research.**

I. BACKGROUND AND RELATED WORK

A. Unusable and ineffective password policies

In 2014, Herley[1] highlighted a fatal flaw in how security is viewed by developers and service providers. Herley points out that most users make a deliberate and rational decision not to adopt security products and use less secure security measures. This is because the cost of improved security overweighs the benefits to an average user. In order to stay up-to-date with security best practices, users must learn and adopt ever-changing security measures, endure temporal and computational overhead, and tolerate the inconvenience caused by additional user input. Herley suggests that secure design should consider the usability of the system as much as security. We believe that this is particularly true for user authentication.

In 2010, Weir et al.[2] demonstrated that the use of Shannon's entropy, as defined in NIST SP800-63, is not an effective metric for password security. While enforcing a larger alphabet size increases theoretical entropy, users often employ coping mechanisms that make the probability distribution of characters in the alphabet non-uniform. Weir's work also showed that when Yahoo replaced their no-minimum-requirement password policy with a 6-character-minimum password policy and a password strength indicator, the average password entropy increased by only 1 bit.

In 2016, Wheeler[3] proposed a password strength estimator that calculates password strength by analyzing the password based on modern password guessing attacks rather than relying on the password's compliance to the LUDS (Lowercase, Uppercase, Digit, Symbol) policy. In the process, Wheeler conclusively showed that a LUDS policy is ineffective in improving password strength.

[1]S. Jeoung is an undergraduate at the Department of Computer Science, Yonsei University, Republic of Korea sullamjeoung at sherlock.com

[2]C. Kim is an undergraduate at the Department of Computer Science, Yonsei University, Republic of Korea chunghunkim at sherlock.com

[3]M. Kim is an undergraduate at the Department of Computer Science, Yonsei University, Republic of Korea myungjongkim at sherlock.com

[4]H. Park is a PhD candidate at the Department of Computer Science, Yonsei University, Republic of Korea p.hanpark at sherlock.com

[5]J. Song is with the Department of Computer Science, Yonsei University, Republic of Korea jooseoksong at sherlock.com

Yet, most web services still use variations of the LUDS password policy in their authentication methods. This is the "worst of both worlds" as a LUDS-enforced password policy makes passwords less memorable and nudges users into insecure coping strategies such as reusing of passwords and creation of dictionary-attackable passwords.[2], [4]*.

B. Password managers

Password managers alleviate issues of memorability introduced by LUDS policies. Because the user only needs to remember a single master password rather than multiple passwords for all of the user's accounts, the user can create secure and unique passwords for each of his/her account without the burden or memorization. However, there is a fundamental problem with password managers: a password manager creates a single point of failure for all of the user's passwords. An attacker only needs to gain access to the password manager to obtain all of the user's passwords.

This has been tackled in 2016 by Stobert and Biddle[5]. Stobert and Biddle proposed a system, *Versipass*, that combines features of a password manager and a cued graphical password. *Versipass* stores graphical cues instead of storing text passwords. When a user attempts to log in, *Versipass* prompts the user to select several boxes within the graphical cue. When the user selects the boxes, *Versipass* generates strong pseudo-random text passwords by hashing the information about the selected boxes. The boxes which the user selects effectively replaces the text password. While the proposal by Stobert and Biddle improves password memorability and partially solves the single point of failure problem, it creates new problems as well. The user must either install a browser extension or open a bookmarklet each time the user wants to log in to a web service. But most importantly, the method proposed by *Versipass* will probably* take longer to authenticate the user compared to a text-based password. According to Herley,[1] increasing the duration of authentication by 5 seconds a day results in 1,389 extra man years per day of human effort. Therefore, new authentication methods should strive to decrease the time used to authenticate, rather than increase it.

C. Generating secure and memorable passwords

In 2015 Huh et al.[6] proposed a password generation mechanism in which the system generates a random string and lets the user replace few characters in the string. This mechanism prevents common characteristics of user-generated passwords such as uppercase letters as the first character and symbols as the last character.

Although continued research in password policies has improved the usability of passwords, we believe that the password must ultimately be replaced. This is because the text password is an old system based on the traditional model of a personal computer. While the keyboard was a standard input device for the traditional computer, input methods in newer devices are evolving. For example, smart devices such as smart phones and smart watches do not have physical keyboards. With the advent of IoT, even more devices, such as vehicles and smart televisions, will not have the keyboard. Yet all of those devices must still support methods of authentication.

D. Browser fingerprinting

We wanted to find a completely new way of authenticating users. Instead of asking the user for more input and more work, we wanted to ask for less. We looked into ways of obtaining information from the user without explicitly asking for it.

Browser fingerprinting collects non-private information about a browser and its environment to evaluate the browser and differentiate it from others. Examples of information used in browser fingerprints include user agent, HTTP header information, and browser plugins. The work of Eckersley in 2010[7] demonstrated the effectiveness of browser fingerprinting by uniquely identifying 83.6 % of all visitors to the website panopticlick.eff.org. In 2016, Laperdrix et al.[8] used new features in HTML5, such as the Canvas API, to build a 17-attribute browser fingerprint which uniquely identified 89.4% of visitors to amiunique.org.

II. EVALUATION FRAMEWORK

An authentication system must satisfy several basic requirements to be considered useful and trustworthy.

A. Credentials

An authentication system must identify users based on credentials of the user. The credentials can take various forms and be composed of different combinations of what the user knows, has, and is. Most web-based user authentication takes the form of password authentication, which is an example of what the user knows. But if other forms of credentials are equally or more difficult to hijack, then they can replace the password.

B. Ease of use

Herley[1] suggests that the cost of increasing the duration of authentication by a mere 5 seconds can be upto 1,389 years of human effort. This number will only increase as more users join web services and create more accounts. Therefore new authentication methods must be intuitive, easy to use, and require less human effort.

C. Installation and set up

We believe that a new authentication method should not require the installation of third-party software or browser extension to the client machine. Client machines are varied. Some users may not have permission to install software on the machine. Some devices, such as IoT devices, may lack some input and output peripherals that make it difficult to implement new authentication systems. Therefore new authentication systems must minimize the difficulty in installation and set up, or avoid it altogether. If a new authentication system is compatible with pre-existing systems, integration and adoption will become easier.

D. Replaceability

Biometric information such as the fingerprint and the iris are authentication methods which have great usability promises. But they are "what the user is", which means compromised biometric data cannot be replaced. A better authentication system must provide a method for users to replace their old credentials in the unfortunate event that a database containing user authentication information is compromised.

E. Different levels of confidence

Different web services require different levels of confidence in user credentials.[1] For example, a banking system should have the highest level of confidence in user credentials, even if it means a reduction in usability of the overall system. On the other hand, websites that do not store critical private information about the user might want to trade confidence in the authenticity of user credentials for improved usability. A good authentication system should provide the service provider with a means to control the level of confidence in user credentials.

III. PROPOSED SCHEME

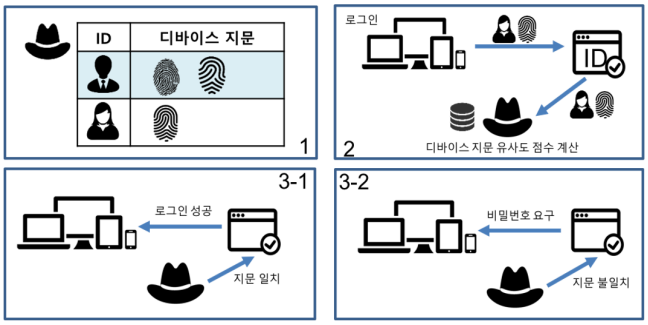

Fig. 1. Overview of Sherlock

A. Browser fingerprinting

According to Laperdrix et al.[8], a browser fingerprint composed of 17 different attributes uniquely identified 89.4% of 118,934 users. Sherlock uses an open source project fingerprintjs2 (v1.4.2)[9] which uses 23 fingerprint attributes which include some of the attributes used in AmIUnique.org, which was part of the work by Laperdrix et al.[8]. On top of the 23 attributes, Sherlock also uses the IP address as another feature.

TABLE I

FEATURES IMPLEMENTED BY FINGERPRINTJS2

Number	Features	Notes	Feature values of test account
1	User agent		Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36
2	Language		en-US
3	Color depth		24
4	Pixel ratio		1
5	Screen resolution		1600,900
6	Available screen resolution		1600,860
7	Timezone offset		-540
8	Session storage		1
9	Local storage		1
10	Indexed database		1
11	CPU class		unknown
12	Platform		Win32
13	'Do Not Track' enabled		1
14	List of plugins		Widevine Content Decryption Module::Enables Widevine licenses for playback of HTML audio/video content. (version: 1.4.8.903)::application/x-ppapi -widevine-cdm ,Shockwave Flash::Shockwave Flash 24.0 r0::application/x-shockwave-flash swf,application/futuresplash spl, Native Client::::application/x-nacl , application/x-pnacl
15	Canvas fingerprint		canvas winding:yes canvas fp: iVBORw0KGgoAAAANSUhEUgAAB 9AAAADICAYAAACwGnoBAAAgAE...
16	WebGL fingerprint		????
17	Ad blocker enabled		false
18	Has the user tampered with languages		false
19	Has the user tampered with screen resolution		false
20	Has the user tampered with operating system		false
21	Has the user tampered with browser		false
22	Touch screen detection and capabilities		0,false,false
23	List of fonts	Implemented with Flash, maintains order	Arial,Arial Black,Arial Narrow, Arial Unicode MS,Book Antiqua,Bookman Old Style,Calibri,Cambria,Cambria Math, Century,Century Gothic...
24	IPv4 address (1st octet)		121
25	IPv4 address (2nd octet)		141
26	IPv4 address (3rd octet)		156
27	IPv4 address (4th octet)		102

B. Signing up

To sign up to Sherlock, the user enters his/her email address, a password, and a PIN. Sherlock calculates a 27-attribute browser fingerprint of the client browser. Then, Sherlock stores the email address, password, PIN, and fingerprint into Sherlock's database. To improve security, all values except for the email address are hashed using the SHA-256 algorithm.

C. Signing in

To sign in to Sherlock, the user enters his/her email address. Sherlock calculates the client browser fingerprint and hashes it using the SHA-256 algorithm. Then, Sherlock compares the newly calculated browser fingerprint with the browser fingerprint in the database.

D. Calculating Feature Match Percentage (FMP) and Fingerprint Accuracy Score (FAS)

Sherlock weighs each fingerprint feature differently. The weights of each feature is based on the relative uniqueness of the feature. If a feature has more unique values, and if a feature has more chance for any two different browsers to have different values, the fingerprint feature is given a higher weight value.

We designed a simulation to measure the uniqueness of each fingerprint. We created a test account *try.sherlock@gmail.com* in the Sherlock service. We also asked volunteers to log their browser environment. Then, we compared the collected fingerprints against the test account. Using the 972 data points collected during the course of the simulation, we calculate the *Feature Match Percentage (FMP)*. We gave the highest weights to the fingerprint features with the lowest *FMP*.

$$FMP_i = \frac{n}{N} \quad (1)$$

n: Number of attempts that match the ith feature of test account

N: Total number of attempts

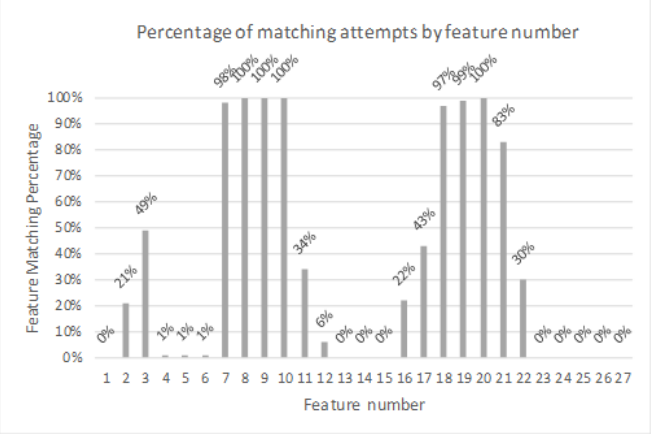

Fig. 2. Fingerprint Match Percentage of each fingerprint feature

E. Different levels of confidence

Sherlock either authenticates the user or prompts for more credentials depending on the *fingerprint accuracy score (FAS)*. *FAS* is calculated from the weights of each matching fingerprint attribute. A higher *FMP* value for a particular fingerprint feature will have a proportionately lower *FAS* value.

$$FAS_i = \frac{\alpha - FMP_i}{\sum_{j=1}^{n} a - FMP_j} \quad (2)$$

α: Normalizing coefficient

n: Number of fingerprint features

- If the user's *FAS* is greater than or equal to 75%, the user is authenticated without further requirements. The user is logged in to Sherlock.
- If the user's *FAS* is greater than or equal to 45% and less than 75%, authentication is unsuccessful and the user is prompted to enter his/her PIN.
- If the user's *FAS* is less than 45%, authentication is unsuccessful and the user is prompted to enter his/her password.

Sherlock uses the threshold values for PIN and password prompts at 75% and 45%. But the thresholds can be set at any arbitrary values. The service provider has precise control over the security-usability trade-off.

F. Registering more fingerprints

The user may have more than 1 device. Even if they are owned by the same person and is in the same general region, the browser fingerprints may vary greatly. But the user may still want to log into Sherlock without having to type the password for all devices. That is why each Sherlock user

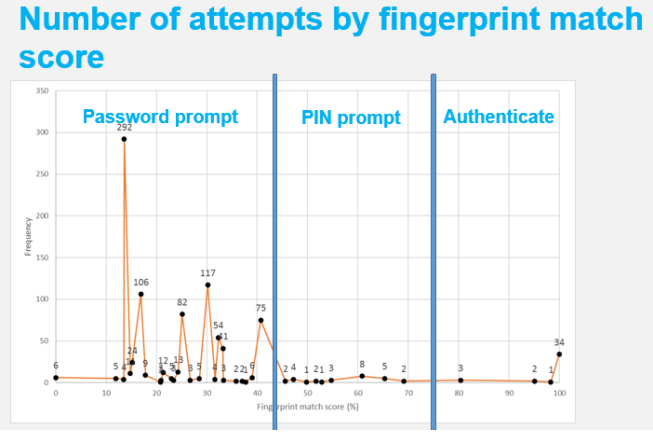

Fig. 3. FAS values of volunteer fingerprints against test account

is able to store up to 3 browser fingerprints. Then, when the user attempts to log in, Sherlock will compare the user's browser fingerprint with all browser fingerprints stored in the Sherlock database.

IV. DISCUSSION

A. Evaluation

- Credentials: Sherlock uses the browser fingerprint as credentials for authentication. Our preliminary testing shows that a browser fingerprint can uniquely identify users. Using multiple features allows enough entropy to prevent brute-force attacks. The process of salting and hashing the features allows defense against database breaches.
- Ease of use: If the user attempts to authenticate from one of the three browser environments that the user has registered, then Sherlock has is the fastest authentication method.

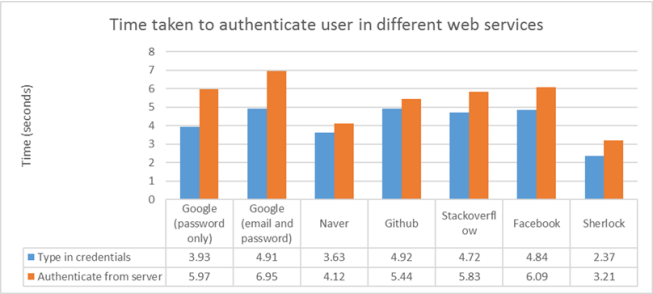

Fig. 4. Time taken to authenticate user in different web services

- Installation and set up: Sherlock requires no installation or user account set up from the user. Sherlock does not require any cognitive or temporal overhead from the user.
- Replaceability: Many of the features that Sherlock uses to create a browser fingerprint is easily replaceable. For example, the user agent can be changed from the browser, system fonts can be installed or uninstalled

from the device's system, and IP addresses can be easily spoofed.
- Different levels of confidence: Sherlock allows service provides control over the threshold values for authentication, PIN prompt, and password prompt.

B. Survey

We interviewed 11 individuals of various backgrounds. The interviewees were composed of high school students, university students majoring in engineering, university students majoring in social sciences, and IT professionals. The interviewees were composed of individuals from the 10 20 age group to the 40 50 age group. All 11 individuals agreed that the Sherlock system was more user-friendly and reported higher satisfaction level compared to previous authentication systems.

C. Limitations

- Our tests were done on a small sample of 972 and was skewed heavily towards devices owned by residents of the Republic of Korea (970 out of 972 people in the sample population were residents of the Republic of Korea).
- In previous authentication systems, the user needs to memorize 1 piece of information, the password. However, Sherlock requires that the user memorize an additional piece of information, the PIN, on top of a password.
- Sherlock does not require additional credentials from the user if the *FAS* value is above the threshold. But it is harder to remember a PIN or a password that the user uses only rarely.

D. Future work

- Adding more features to calculate the browser fingerprint will increase entropy and increase the security of Sherlock.
- Further study on True Positive, True Negative, False Positive, and False Negative entries is required to improve the usability of Sherlock.

V. CONCLUSION

We proposed a novel authentication system called Sherlock, which uses browser fingerprints in place of passwords. The fingerprint and its individual attributes can be changed unlike biometric fingerprints. We conducted a sample test of users and showed that Sherlock is as secure as a password-based authentication scheme. We conducted a focus group survey and showed that Sherlock is easier to use and more user-friendly compared to password-based authentication schemes.

REFERENCES

[1] C. Herley, "More is not the answer." *IEEE Security & Privacy*, vol. 12, no. 1, pp. 14–19, 2014.

[2] M. Weir, S. Aggarwal, M. Collins, and H. Stern, "Testing metrics for password creation policies by attacking large sets of revealed passwords," in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010, pp. 162–175.

[3] D. L. Wheeler, "zxcvbn: Low-budget password strength estimation," in *Proc. USENIX Security*, 2016.

[4] P. G. Inglesant and M. A. Sasse, "The true cost of unusable password policies: password use in the wild," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2010, pp. 383–392.

[5] E. Stobert and R. Biddle, "A password manager that doesn't remember passwords," in *Proceedings of the 2014 workshop on New Security Paradigms Workshop*. ACM, 2014, pp. 39–52.

[6] J. H. Huh, S. Oh, H. Kim, K. Beznosov, A. Mohan, and S. R. Rajagopalan, "Surpass: System-initiated user-replaceable passwords," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 170–181.

[7] P. Eckersley, "How unique is your web browser?" in *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2010, pp. 1–18.

[8] P. Laperdrix, W. Rudametkin, and B. Baudry, "Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints," in *37th IEEE Symposium on Security and Privacy (S&P 2016)*, 2016.

[9] V. Vasilyev, "Fingerprintjs2," https://github.com/Valve/fingerprintjs2, 2015.