Password Cracking and Malicious USB Devices

Michael Hehmann[1]; Evan Jacobs[2]; Kristiana Rendon[3]; Chad Spensky[4]; Bao Huynh[5]
[1]Mathematics, Cal Poly San Luis Obispo; [2]Electrical Engineering, Ventura College; [3]Computer Science, Pasadena City College; [4]Computer Science, UCSB; [5]NAVSEA ETO 207

Abstract

Many organizations enforce strict security policies in order to prevent adversaries from compromising their information-sensitive systems. These policies include creating complex passwords and prohibiting the use of USB devices on machines. Due to the inconvenience of these policies, many users create passwords based on certain mnemonic devices, such as visual keyboard patterns, and adopt a lenient attitude on USB device usage. It is important to demonstrate to users that these shortcuts lead to a significant weakening of the organization's security. To this end, we created an interactive web-based platform to quickly crack certain visual keyboard pattern passwords.

Our website allows a user to input pattern-based passwords which are cracked in real-time using the password cracking tool hashcat. Additionally, we demonstrate how an attacker can steal the password file from a Linux machine with a USB device and send the hashes to a website to be cracked.

Introduction

Keyboard Patterns

- Visual keyboard pattern passwords are used to meet complexity requirements.
- These passwords may look strong, for example "%TGB6yhn&UJM".

Figure 1: Examples of parallel passwords.

- However, they are weak and easy to crack, compromising security.

USB Rubber Ducky

- The USB Rubber Ducky acts like a keyboard when plugged into a computer.
- This device holds a script that will steal the password/hash files and send them to our server to be cracked.

Method

- Prince Processor was used to combine short sequences of adjacent keys into length 12 passwords.
  - This dictionary was supplied to hashcat to crack the password hashes.
- The USB Rubber Ducky utilizes its own scripting language, called DuckyScript, to execute Bash commands on the target machine.
- The website was developed with a full-stack design. It uses a Flask server, Python backend, SQLite3 database, and CSS/HTML/Javascript frontend.

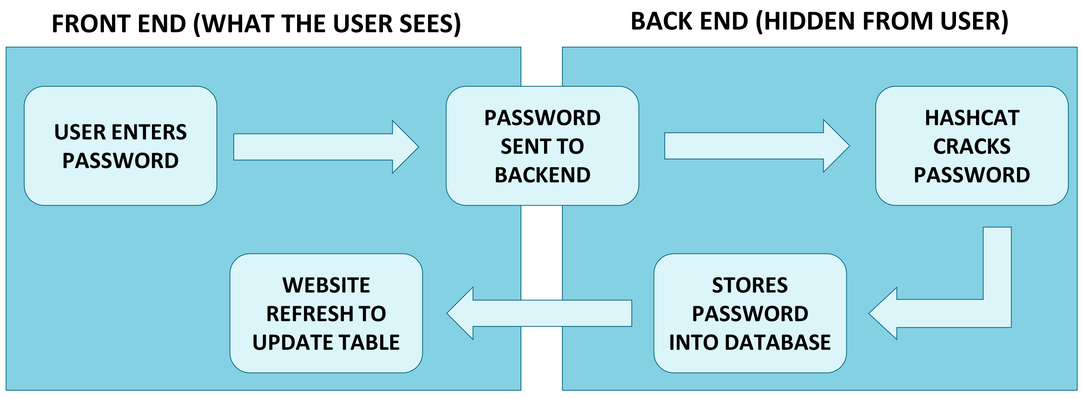

Figure 2: Web Application Flow Chart

Figure 3: Attack vector for the USB Rubber Ducky.

Results

- Hashcat checked 10 million hashes per second on a laptop with a dedicated GPU.
- The remote server hosts all the program files for the website. A user will be able to connect to this server and input passwords to be cracked.
- The USB Rubber Ducky is capable of stealing a password file from another machine where it will be sent to the server to be cracked and displayed onto the website.

Figure 4: Estimated cracking time vs. actual cracking time on our website (in milliseconds).

Conclusion

While certain keyboard pattern passwords fulfill complex password requirements (character set, length, forced resets, etc.), these passwords can be easily and quickly cracked on weak hardware. Our web application demonstrates this weakness and provides an interface for users to test parallel pattern passwords. Users should opt for stronger passwords, even at the expense of convenience. Additionally, personnel should be aware of the risks that USB devices pose.

References

[1] Bonneau, J., Herley, C., Oorschot, P. and Stajano, F. (2012). The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. *IEEE Symposium on Security and Privacy*.
[2] Chou, H., Lee, H., Hsueh, C. and Lai, F. (2012). Password Cracking Based on Special Keyboard Patterns. *International Journal of Innovative Computing, Information and Control*, 8(1(A), pp.387-402.
[3] Herley, C. and Oorschot, P. (2011). A Research Agenda Acknowledging the Persistence of Passwords. *IEEE Security & Privacy Magazine*.
[4] Schweitzer, D., Boleng, J., Hughes, C. and Murphy, L. (2009). Visualizing Keyboard Pattern Passwords. *6th International Workshop on Visualization for Cyber Security*, pp.69-73.
[5] hashcat. 2017. *hashcat.net*
[6] princeprocessor. 2016. *github.com/hashcat/princeprocessor*
[7] Hill, R. pyhashcat. 2016. *github.com/rich5/pyhashcat*
[8] Grassi, P., Fenton, J., Newton, E., Perlner, R., Regenscheid, A., Burr, W., Richer, J., Lefkovitz, N., Danker, J., Choong, Y., Greene, K. and Theofanos, M. (2017). Digital Identity Guidelines: Authentication and Lifecycle Management.

Acknowledgements

We would like to thank our mentors Bao Huynh and Chad Spensky for their invaluable guidance and patience during this project. We would also like to thank Brian Meadows, Sara Melvin and Socrates Frangis at NSWC PHD for their help and advice on various technical issues. Additionally, Francesco Disperati and Sam Green at UCSB helped us with web development and presentation skills and we thank them for their time and expertise.