A Practical Approach to Distributed WPA/WPA2 Cracking with CUDA

Sunjay Dhama, Cal Poly San Luis Obispo

sdhama@calpoly.edu

February 20, 2014

Abstract

This project builds upon the work presented by security research Thomas Roth at Defcon 20 by providing a practical implementation to cracking Wi-Fi Protected Access (WPA) and Wi-Fi Protected Access II (WPA2) passwords with Compute Unified Device Architecture (CUDA) [17]. It does this by incorporating a password database that allows an attacker to easily store their word list in a database, which the cracking nodes query. This paper describes the implementation of the attack and discusses countermeasures to help mitigate it.

Contents

1 Introduction

The first section of this report will go over background information, so the reader can fully comprehend the work presented. The next section will give an overview of related work. The third section will detail why this implementation is important. The fourth section will discuss the technical details of the work. The fifth section will go over results. The sixth section will speak about ways to mitigate this attack. The second to last section will draw some conclusions and the final section will make suggestions for future work.

2 Background

This work utilizes three important concepts: WPA, CUDA, and Amazon Web Services (AWS) Elastic Compute Cloud (EC2). Each of these three pieces were integral in this project.

2.1 WPA

WPA/WPA2 are security protocols released in 2003 and 2006 for wireless LAN communication, and are meant to provide confidentiality and integrity [11] . WPA/WPA2 was designed in order to fix flaws in Wired Equivalent Privacy (WEP) [16], which has been shown to be easily exploitable [20]. Neither WPA nor WPA2 are completely secure anymore either. Researchers Martin Beck and Erik Tews showed how to exploit some of their flaws[19]. However, their work is beyond the scope of this project, which does not utilize those discovered flaws. Simply put, this project assumes WPA/WPA2, which are essentially a modified version of SHA-1, are unflawed crytographic hashing algorithms.

By definition, SHA-1 is a one-way function, which means given the output of a text encoded by SHA-1, the original text cannot be derived without the key[8]. However, there are several methods to obtaining a WPA/WPA2 password. The most predominant method is a dictionary-based attack where the attacker uses a word list of possible passwords and the Access Point (AP) name to generate Pairwise Master Keys (PMKs) [24]. A PMK is used to generate a Private Transient Key (PTK), which is used to generate a Message Identity Code (MIC) [24]. The generated MIC and the MIC captured by an attacker from the four-way handshake, the process of a computer authenticating itself to the router when attempting to join the network, are then compared to determine if they match [24]. If there is a match, the password used to generate the PMK is the password for the network [24]. This is the type of

attack examined.

When WPA/WPA2 was first developed, it was infeasible to crack a password over 7 characters in length due to amount of time required to generate each PMK [22]. However, since the release of the CUDA SDK, which allows programmers to utilize the General-Purpose Computing on Graphics Processing Units (GPGPU) for computationally intense operations other than graphics processing, the feasibility of cracking WPA/WPA2 has increased exponentially [22].

2.2 CUDA

The release of the CUDA SDK drastically changed the landscape of encryption. Roth noted this when he said:

> Modern high-end GPUs feature more than 400 cores which can be used to accelerate a lot of applications in the science sector, in the medical sector, in the financial sector and in the *encryption sector* - practically everywhere where the same program needs to run over a large amount of independent data elements.

Running the same program "over a large amount of independent data elements" is exactly what happens during a dictionary-based attack, so it is logical to incorportate the CUDA SDK [17]. Leveraging the CUDA SDK to speed up the cracking process has been prevalent for several years and a multitude of other cracking applications exist that utilize it. Three of the most popular applications are Pyrit, oclHashcat, and Cryptohaze Multiforcer [14, 1, 5].

However, all three have drawbacks. While oclHashcat is arguably the fastest hash cracking software available to date, it is not open-source and until very recently, did not support running in a distributed environment [3]. The only currently available open-source software to utilize CUDA to crack WPA/WPA2 is Pyrit, which has *limited* support for running in a distributed environment [14]. However, Pyrit is not maintained and has a number of bugs that hamper adoption of it [14]. Cryptohaze Multiforcer is very similar to this project [5]. It too supports easily cracking password hashes in a distributed environment, but presently does not support cracking WPA/WPA2 [5]. This project seeks to overcome all of those issues by being completely open-source and easy for the non-technical user to run when crack WPA/WPA2 in a distributed environment. However, obtaining access to a large number of physical computers with both Nvidia GPUs and the CUDA SDK installed can be a challenge.

2.3 AWS EC2

If a user does not have access to physical computers with Nvidia GPUs, EC2 offers customers the option of paying per hour for the usage of virtual computers, referred to as instances [4]. It is no longer necessary to spend thousands of dollars on a *cracking rig* [10]. Instead, one pays for the amount of time they need [4]. EC2 is of interest in this application due to the introduction of GPU instances, which were used to benchmark this application. They have the following specifications:

Cluster GPU Quadruple Extra Large Instance
22 GB of memory
2 x Intel Xeon X5570, quad-core Nehalem architecture
2 x NVIDIA Tesla Fermi M2050 GPUs
1690 GB of instance storage
64-bit platform
I/O Performance: Very High (10 Gigabit Ethernet)
API name: cg1.4xlarge

Table 1: EC2 Instance cg1.4xlarge Specifications

3 Existing work

Before going into detail about this project, it is necessary to summarize previous work. Although Roth has not publically released the source code for his application, others following the steps he laid out did [24]. In 2012, Chaodong Zheng, Chengwen Luo, and Kartik Sankaran (ZLS) from the University of Singapore released their implementation of Roth's code, which is continued by this project [24].

3.1 Importance

ZLS's work was important for the following reasons:

(I) ZLS removed almost all monetary barriers to entry into advanced password cracking [24]. Essentially anyone could partition EC2 GPU instances and crack passwords [24].

(II) The rate at which it takes for the password to be found decreases linearly with additional EC2 instances. An attacker with a high value target could launch more instances and generate more PMKs per second [24].

3.2 Summary

Since this application built upon ZLS's source code, it is necessary to explain what they did and their results. However, the most benefit can be gained from reading their work in full. ZLS leveraged the AWS API to create a program that automates the creation of EC2 GPU instances to crack WPA/WPA2 passwords [24]. The master program relays the needed information to two (or more) GPU instances and those instances iterate through their range of passwords [24]. Their work made use of the open-source implementation of WPA/WPA2, which can be found in Pyrit [24, 14]. They were able to achieve a cracking efficiency of 25,000 PMKs/s/dollar by using spot instance pricing, which is an improvement over Roth, who achieved 22,500 PMKs/s/dollar [24, 17].

4 Motivation

The primary motivation for this work was to give security penetration testers (pentesters) a tool they could use when auditing large companies. Pentesters typically have varying degrees of experience with programming and often utilize software tools to aid in their audit. Pentesters want tools that are effective and easy to use. To do this, ZLS's code had to be modified.

4.1 Practicality

This work is important because the publically released source code of ZLS does not provide a practical implementation for cracking WPA/WPA2 as ZLS's application checks only phone numbers when attempting to crack a password. Although use of one's phone number as one's router password is a popular choice among consumers, it is not all-encompassing [24]. In personal experimentation and surveying of peers, it was found that phone numbers accounted for a user's password roughly 20% of the time. This means that 80% of the time, the password could not be found due to it not being a phone number. However for several years, crackers have been using large *cracking dictionaries*, files with a large list of words that are commonly used as passwords, in an attempt to check *only* the most common passwords. Dictionary-based attacks continue to be used to crack passwords because they are still effective

[16]. This work took the concept of the word list and simply put it online in a database. This means an attacker is no longer limited to phone numbers, but only by how comprehensive their word list is. However, an attacker is never truly limited by the number of words in their word list as more comprehensive dictionaries are released on a regular basis and better ones can always be generated [6].

4.2 Ease of Use

Although not the focus of this work, the pcap2hccap software was expanded upon [21]. The added functionality allows a user to specify a packet capture containing the four-way handshake and generate the .txt file, which is needed by this application to crack a given AP password. It was not immediately clear how ZLS originally generated their .txt file, but with the additions made to the pcap2hccap program, the process is now trivial. This removed another barrier to entry to cracking WPA/WPA2 passwords.

Another area in which was not the focus of this work, but which was made easier through automation was the creation of the database. A C++ program with a command-line interface creates the database and inserts words from a user specified file for the user. The user needs only a basic understanding of Database Management Systems (DBMSs) to generate the word list database.

Increasing ease of use and implementing a practical attack were main motivations for this project. If more users are able to crack router passwords easily, people will think twice about using a simple or default password and hopefully use a randomly generated one. Software such as LastPass can facilitate that[2].

5 System Architecture

As shown in Figure 1, the entire system includes one master node, multiple slave nodes, and one word list database. The master node runs the master program, the slave nodes run the cracking program, and the database node runs a MySQL database, which is opento the slave nodes for connection. The user specifies the location of the handshake file, which they have generated using the modified pcap2hccap application, the number of slave nodes, the port number at which those slave nodes are expecting a connection, the IP addresses of the slave nodes, the IP address of the database, and the range of passwords in the database that the user would like to check. The user puts all of the aforementioned information into a configuration file, which the master program reads and uses to distributes the work to the running slave nodes. Once the slave nodes recieve the necessary information from the master

node, they begin querying the database for their range of passwords and generating PMKs. Once the PMKs are generated, the slave nodes generate the MICs and compare them against the existing MIC. If there is a match, the slave will send the password back to the master. The master will then tell the other slave nodes to stop running. If there is not a match, the slaves will request more passwords from the database and the process will continue until either the password is found or there are no more passwords to query for in the range.

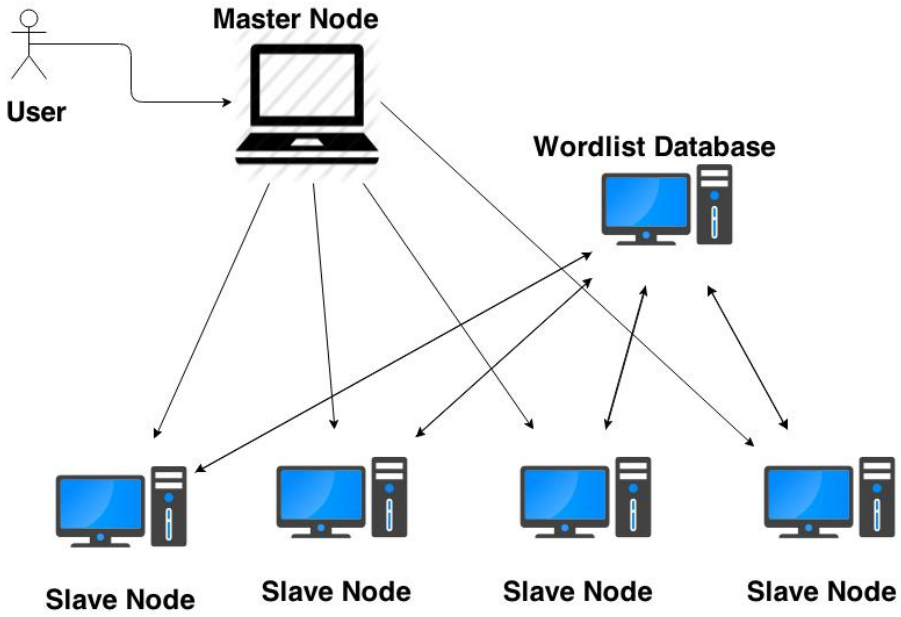

Figure 1: System Architecture

6 Implementation

The implementation of this project has 3 main parts, the master node, the slave node(s), and the database. The master node provides a way for the user to interact with the program. The cracking program has two distinct parts, the CPU code and the GPU code, both of which were modified. The final piece was the database code, which did not require a lot of programming, but moreso a lot of setup.

6.1 Master

The original implementation of the master node was fairly simplistic. It simply created the AWS instances using the API, sent all the instances the necessary information for cracking and then waited. During this waiting period, it did no computation. It simply checked if the slaves have sent back the password. It seemed wasteful to create an AWS instance to do this, so the code was modified to run on the local machine.

There was only one one main piece of functionality added to the master code. It was to remove the limitation that the slave nodes be only AWS instances. Instead, a configuration file where a user can specify the information required by the application, was added. This means that if a user has access to a large number of PCs either at home or at work, those could be utilized as well. No modifications were made to the original code that launched the EC2 GPU instances. That file was simply excluded. In all but the last testing phase, testing was done on either local or university machines. This not only saved money, but also was easier overall.

It is important to point out that in the original implementation it was not possible for the master program to launch an instance and have it run the cracking program. One needed to have already installed the CUDA SDK and necessary dependencies on the instance and then compile and run the slave code. Once a user had done that, they could create a snapshot of the instance and use the program to load that image onto the instance, so it would be ready to connect to the master program. This process is too technically long and complex for the a user and too time-consuming for any pentester. Most pentesters would either a) Already have several physical or virtual machines at their disposal to run the cracking code on or b) Would not bother with creating a snapshot of a configured instance, but simply create and configure the instances manually as needed. This is why there is an easy to use script incorporated into this project to help in the configuration process of an EC2 GPU instance.

6.2 Slave

There were many areas for improvement in the original implementation, but the focus was on making this tool practical. This is why the main modification to the cracking code was the incorporation of reading passwords from a MySQL database. Once it became clear the original implementation for dividing the password range between the CPU and GPU threads would still work, it was just a case of using those ranges when querying from the database. Due to the fact that the greatest number of open connections to the database from one particular instance would not

exceed 30 (highly unlikely a single PC would have more than 30 CPU threads and GPUs combined), a connection pool was not utilized. Each CPU thread has its own connection to the database and makes queries independent of the other threads, so the threads do not have to wait for each other. Each CPU thread queries for a set number of passwords, known as the CPU Password Size. The passwords are then individually passed to a function, which generates the PMKs and MICs. The GPU code is not much different except the number of passwords it queries for is far greater and is referred to as the GPU Password size. As one would expect, the GPU code does all the generation of the PMKs on the GPU and then transfers the result back to the CPU to generate and compare the MICs.

After profiling the CUDA portion of the cracking code, the limiting factor for the total number of threads present in each CUDA kernel was registers. An attempted solution to this was to break the kernel into two kernels and call them one after another. The first kernel computed the first 4096 iterations of SHA-1 and the second kernel computed the second 4096 iterations. Unfortunately, this did not provide any speedup. In fact, there was a slow down of about 10% in the number of PMKs generated per second, but there was not time to investigate further. It is likely that the overhead of calling the kernel multiple times removes any benefit of additional registers.

6.3 Database

In thinking about how to make ZLS's code more practical, it was clear a word list had to be added, but how best to implement it is the challenge After much thought, a MySQL database, hosted on an AWS instance, was used. This would allow a large number of slaves to connect to the database at one time with no lag. A MySQL database was chosen because MySQL is billed as a *production level* database, used by many commercial enterprises. To ease in the setup of the database, a C++ application was developed, which not only created the database, but also quickly inserts the potential passwords from a file up to 1GB in size. There was much internal debate about how best to load and store a file, but in the end an in-memory database had too many drawbacks. The main drawback with an in-memory database is not only the potential for corruption, but also that the size of the database could be grow to be greater in size than that of memory. So, a MySQL database was launched on an EC2 instance with the following specifications:

Cluster CPU Quadruple Extra Large Instance
23GB of 1333MHz DDR3 Registered ECC memory
2 x Intel Xeon X5570 (quad-core Nehalem)
2 x 845GB 7200RPM HDDs
64-bit platform
I/O Performance: Very High (10 Gigabit Ethernet)
API name: cc1.4xlarge

Table 2: EC2 Instance cc1.4xlarge Specifications

7 Results

Although this project is still a work-in-progress, some great things were achieved. The most obvious result is that users now have the ability to easily run this cracking software on physical hardware as well as virtual using words from their own word lists. There is the added barrier of creating a database, but unless the word list is split up and placed on each machine, there is no other way to easily overcome this obstacle.

7.1 Benchmarks

The benchmarks shown in Table 3 were achieved with the password located at index 5,555,555. However, due to the GPU Password Size more PMKs than necessary were always generated. As the reader can see, having 0 CPU threads increased the overall

CPU Password Size	GPU Password Size	Passwords Tested	Time (s)	PMKs/s
0	100,000	5,600,000	202	27,722
0	500,000	6,000,000	186	32,258
0	1,000,000	6,000,000	183	32,786
0	2,000,000	8,000,000	236	33,898
50,000 (8 threads)	1,000,000	6,000,000	222	27,027
100,000 (8 threads)	500,000	5,800,000	207	28,019
100,000 (16 threads)	500,000	6,100,000	248	24,596

Table 3: Benchmarks

speed. This is likely due the fact that each thread has its own database connection, which slowed things down. Additionally the number of PMKs/s increased as the GPU

Password Size increased, meaning there was a fair amount of overhead associated with transferring data on and off the GPU. The fastest recorded speed was 33,898 PMKs per second. Unfortunately, this is substantially less than the roughly 48,000 PMKs/s that Roth achieved [17]. However, with some more optimizations that speed could be obtained.

8 Mitigation Techniques

This work is proof that both WPA and WPA2 are not secure protocols and should be avoid whenever possible. Unfortunately, the highest grade encryption offered by some consumer-grade routers is WPA2. If that is the case and a more secure algorithm cannot be chosen, the best option is to use a passphrase longer than 12 characters that is randomly generated and contains both uppercase and lowercase letters as well as symbols. This ensures that the password does not exist in any cracking dictionary. The password to the network should also be changed every 6 months and whenever a guest has finished using the network. Another preventative measure is to limit the signal range of one's router to only areas a user controls, in order to prevent an attacker from obtaining a four-way handshake. Without the handshake, an attacker cannot penetrate the network [24]. A final precaution would be to use a Virtual Private Network (VPN) while at home, so that even if an attacker does gain access to the network, s/he cannot access any of the information being sent or receive as it is encrypted. However, this is not guaranteed [15].

9 Conclusions

With the introduction of the cloud and applications that easily distribute work, combined with the speedup achieved by leveraging CUDA, use of WPA/WPA2 for any secure transactions is no longer acceptable. Unless router manufactures switch to a more secure encryption algorithm, attackers will continue to target WPA/WPA2 due to its known weaknesses [24]. It does not matter how good one's security procedures and protections are in the corporate environment as attackers will continue to target the point of least resistance, which is the home network. Once an attacker has access to an employees home network, it is substantially easier to then gain access to the corporate network. This software aims to increase the effectiveness and success rate of penetration testers, which will help lead to a safer and more secure Internet.

10 Future Work

Looking toward the future, there are a number of things yet to be added to this application. The following is a prioritized list of what still needs to be done:

1. Limit all CPU and GPU threads to only one database connection each.
2. Query the database for the largest number of passwords possible to store in memory, minimizing the number of queries to the database.
3. Further optimize the CUDA code using with the Nvidia Visual Profiler
4. Add support for the addition of slave nodes during the cracking process.
5. Support reading from a local or in-memory database.
6. Incorportate this project into the Cryptohaze Multiforcer framework.

References

[1] "oclhashcat-plus." [Online]. Available: http://hashcat.net/wiki/doku.php?id=oclhashcat_plus

An explanation of oclHashcat-plus by the group that created the software.

[2] "The secure and trusted way to store passwords," Online. [Online]. Available: https://lastpass.com/how-it-works

[3] "oclhashcat v1.02 going for distributed cracking," January 2014. [Online]. Available: http://hashcat.net/forum/thread-3047.html

oclHashcat Administrator explains new support for distributed cracking in the latest release of their software.

[4] Amazon, "Amazon ec2," Online, 2014. [Online]. Available: http://aws.amazon.com/ec2

[5] Bitweasil, "Cryptohaze multiforcer," Online, 2012. [Online]. Available: http://www.cryptohaze.com/multiforcer.php

[6] J. Bonneau, "The science of guessing: analyzing an anonymized corpus of 70 million passwords," in *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012, pp. 538–552.

[7] A. Cassola, W. Robertson, E. Kirda, and G. Noubir, "A practical, targeted, and stealthy attack against wpa enterprise authentication," in *Proceedings of NDSS*, vol. 2013, 2013.

This paper describes novel additions to the 'Evil Twins' attack that prove to be very successful in duping users to transmit their passwords to an attacker

[8] D. Eastlake and P. Jones, "Us secure hash algorithm 1 (sha1)," 2001.

[9] D. Goodin, "Why passwords have never been weakerand crackers have never been stronger," 8 2012. [Online]. Available: http://arstechnica.com/security/2012/08/passwords-under-assault/

[10] J. Gosney, "Password cracking hpc," in *Passwords12 Conference, Oslo*, 2012.

[11] I. . W. Group *et al.*, "Ieee standard for information technology–telecommunications and information exchange between systems–local and metropolitan area networks–specific requirements–part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 6: Wireless access in vehicular environments," *IEEE Std*, vol. 802, pp. 1–51, 2010.

[12] G. Korlam and D. Ç. K. Koç, "Password cracking in the cloud."

Report on detailing the power and low-cost effectiveness of various cloud crackers.

[13] A. Kurtz, F. Freiling, and D. Metz, "Department informatik," 2013.

[14] L. Lueg, "The twilight of wi-fi protected access," 2008. [Online]. Available: http://pyrit.wordpress.com/the-twilight-of-wi-fi-protected-access

An explanation of Pyrit and it's uses by its creator.

[15] M. R. Moxie Marlinspike, David Hulton, "Defeating pptp vpns and wpa2 enterprise with ms-chapv2," in *Defcon*, July 2012.

[16] T. Ohigashi and M. Morii, "A practical message falsification attack on wpa," in *Procedings of Joint Workshop on Information Security, Cryptography and Information Security Conference System*, 2009.

This paper presents a practice approach to cracking WPA without GPU assistance

[17] T. Roth, "Breaking encryptions using gpu accelerated cloud instances." in *Black Hat Technical Security Conference*, 2011.

[18] V. Tendulkar, R. Snyder, J. Pletcher, K. Butler, A. Shashidharan, and W. Enck, "Abusing cloud-based browsers for fun and profit," in *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 2012, pp. 219–228.

[19] E. Tews and M. Beck, "Practical attacks against wep and wpa," in *Proceedings of the second ACM conference on Wireless network security*. ACM, 2009, pp. 79–86.

[20] E. Tews, R.-P. Weinmann, and A. Pyshkin, "Breaking 104 bit wep in less than 60 seconds," in *Information Security Applications*. Springer, 2007, pp. 188–202.

[21] tux mind, "cap2hccap," Online. [Online]. Available: http://sourceforge.net/projects/cap2hccap

[22] K. T. Viet, "Gpu - accelerated wpa psk cracking solutions," 2010.

This paper introduces GPUs into the realm of WPA cracking.

[23] L. Yong-lei and J. Zhi-gang, "Distributed method for cracking wpa/wpa2-psk on multi-core cpu and gpu architecture," *International Journal of Communication Systems*, pp. n/a–n/a, 2013. [Online]. Available: http://dx.doi.org/10.1002/dac.2699

[24] C. Zheng, C. Luo, and K. Sankaran, "Distributed wpa password cracking system exploiting amazon gpu clusters," April 2012.

Report on implementation of Distributed WPA cracking utilizing AWS GPU instances.