Distributed WPA Cracking

Rodney Beede

University of Colorado  
Boulder, CO 80309-0430



Ryan Kroiss

University of Colorado  
Boulder, CO 80309-0430



Arpit Sud

University of Colorado  
Boulder, CO 80309-0430



**ABSTRACT**

In this paper, we describe a distributed system we developed for doing lookups in a rainbow table for passwords to WPA-PSK 1/2 wireless encrypted networks. Our motivation for developing such a system came from the abundance of distributed systems for generating rainbow tables but a lack of any to do the lookup.

The rainbow tables for even just a million passwords can be nearly 40GB in size. The problem we address is how a distributed system could be used to provide fast lookup of matching passwords in this table for some given captured wireless network data. We chose to focus on WPA versus other wireless encryption techniques (e.g. WEP) because it offers the best encryption that cannot be feasibly defeated via brute force methods.

Our approach involved writing new code to handle user submitted jobs of wireless data packets and modifying existing code from a project know as coWPAtty (todo reference) which handled the actual table lookup. We used a cluster of 9 nodes running on virtual machines to handle the work load and measured the performance of our system to the serial original coWPAtty implementation.

Our results show an order of magnitude of 8 times in speed increase for our distributed system versus the serial coWPAtty. While our data shows that serial coWPAtty could still return a single result in an order of several seconds versus our system returning results in under 1 second the scalability of our system would provide much more usability to offer this as a service.

We concluded that our results show that the importance of using a strong password for wireless networks is still very important. WPA is an improvement over previous encryption ciphers used in wireless networks, but it is still susceptible to weak passwords chosen by the end user. The most important effect from our research shows that using generated large rainbow tables of pre-computed passwords can provide an easy, fast, and scalable tool for finding weak passwords in encrypted wireless networks.

**Categories and Subject Descriptors**

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2011

Template - Copyright 2010 ACM 1-58113-000-0/00/0010

C.2.1 [**COMPUTER-COMMUNICATION NETWORKS**]: Network Architecture and Design– *Wireless communication;* K.6.5 [**MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS**]: Security and Protection– *Unauthorized access (e.g., hacking, phreaking);*

**General Terms**

Security

**Keywords**

WPA, rainbow table, dictionary attack

# INTRODUCTION

The use of wireless networks based on the 802.11 standards (also known as WiFi) has become much more common place in your typical household. Early advents of these networks provided security of their networks to limit access and protect sensitive data with a protocol standard known as WEP (todo citation). Research and analysis into the cipher algorithms used in this protocol lead to discovered weaknesses that could easily be exploited by a single computer with modest hardware using brute force. The deficiency of this protocol lead to the development of a new one knows as WPA (todo, reference). This newer protocol utilizes much more advanced and stronger encryption ciphers which have made the possibility of using brute force attempts not feasible (todo citation). With the advent of these new security measures ways of attacking them have evolved as well.

One technique for deciphering encrypted data or accessing protected resources is to simply try a large number of different passwords in the hope to get one correct. The nature of wireless networks enables one to capture encrypted packets as they pass through the air and store them to disk in their encrypted form. Part of the wireless encryption protocol involves handshaking at the initial connection in order to authenticate the client to a wireless access point. This handshake includes some initialization vectors (IV’s) that are used to allow the client to encrypt the password they will send for authentication. There are multiple types of WPA encryption. Some which use client certificates and server certificates for encryption of the data while others use simple passwords (WPA-PSK) that the user provides upon establishing the connection. The password has a minimum of eight characters and a maximum of 63 and is case sensitive. The encryption key is derived from this password and the case sensitive name of the wireless network (SSID) (todo citation). Attempting to brute force this key space would also prove to be computationally unfeasible. A common alternative is to try a large dictionary of common passwords. This dictionary can also include slight permutations on words such as appending numbers to the end in order to catch common passwords used by users.

The dictionaries used could contain millions of words which would require some significant amount of time to process for each wireless network someone wished to attack. A complimentary technique is to generate a large rainbow table of keys based on the dictionary and several common wireless network names. This would then enable an attacker to simply capture some wireless network handshake data (IV’s) and do a lookup in the pre-computed rainbow table for a match. Such a tool already exists and is known as coWPAtty.

coWPAtty is a serial program that can generate a rainbow table with hash values for matching keys based on wireless networks names (SSIDs) and WPA-PSK passwords. The generation is done in a serial manner and written to a file that can later be used in conjunction with captured wireless data. All operations in the original coWPAtty code are done in a serial manner on one machine.

For our project we decided to create a distributed system that could perform this rainbow table lookup among a cluster of nodes in order to increase performance. We chose to not generate the rainbow tables as others have already done so (todo citation). Instead we chose to use an existing rainbow table that contained pre-computed keys for 1,000 wireless network names (SSID’s) and divide it across multiple machines for doing lookup queries.

We developed a system with a single master node and multiple worker nodes that handled the job submission and work. The master node was written from scratch as a Java web application and is responsible for queuing jobs, sending them to the workers, and reading back the results. The worker nodes were created by modifying the original coWPAtty code to function in a distributed manner and are described laster.

Our testing methodology consists of capturing wireless data from our own personal networks and submitting them to the distributed system. We compared the times for our system to find the correct solution to that of the original coWPAtty serial code running on a single node. We document our findings in the results and conclusions portion of this paper.

All the project code and testing data can be found on the project’s website at http://code.google.com/p/distributed-wpa-cracking/

# ARCHITECTURE

## Overview



Figure . Architecture Overview

The system is comprised of a master node which is responsible for coordination of work among a cluster of worker nodes. The worker nodes each hold a portion of the rainbow table in memory and listen for requests over TCP from the master node.

A network file system is used as a shared location for the job input data and output results. In addition it serves as a central point for the binary code that will be run on both the master and the worker nodes.

The wireless network is not connected to the master or worker nodes nor must it even be anywhere in proximity to them. The end user captures wireless data from the wireless network using their computer and saves the result to disk. At a later time, after sufficient data has been captured, the user can then upload the captured data to the master node for processing.

Communication from the wireless network to the user’s computer is done via standard 802.11 wireless protocols. The data capture can be done with already existing tools such as airocrack (TODO reference). Communication from the user to the master is accomplished using the HTTPS protocol over a secure TCP socket. Communication internally occurs between the master node to a worker node. Worker nodes do not communicate with each other. The master node uses TCP connections with the SSH protocol for remotely starting the worker node binaries and a custom propriety protocol for sending running worker nodes jobs and checking their status.

## Master Node

### Network File System

In order to be able to share input data (the wireless capture file) with the worker nodes a networked file system was setup with common access between all worker nodes and the master node. This choice was made because it was a simpler configuration versus having the master transmit a copy over TCP to each worker node and have them store a temporary copy on their local disks. The master node was used to host the network file system, but a separate device could have been used as well

When the user uploads a job request to the master it stores the uploaded capture file to a newly created folder on the network file system. The folder is given a name that matches the newly generated job id. This folder is visible to the worker nodes and will be used by them to later read the wireless capture file and also for storing the output results.

Common binaries for all the workers such as the actual worker binary executable is stored here as well. This allows for updated code to be published in one place and be applied to all workers at once. A simple restart of the worker process on each worker node will then load the new updates to any code.

The large rainbow table data however is not stored on the network file system but instead a read-only copy exists on each node on local disks. Because gigabytes of data must be read from the rainbow table by each node at startup into memory having copies on local disk allow for shorter startup times since disk contention and network congestion are avoided. While this would add some burden to adding updates to the rainbow table this isn’t done frequently and just requires more time to copy any new data to each node’s local disk. In addition our project doesn’t generate new rainbow tables so no updates were really necessary.

## First Page Copyright Notice

Please leave 3.81 cm (1.5") of blank text box at the bottom of the left column of the first page for the copyright notice.

## Subsequent Pages

For pages other than the first page, start at the top of the page, and continue in double-column format. The two columns on the last page should be as close to equal length as possible.

Table . Table captions should be placed above the table

|  |  |  |  |
| --- | --- | --- | --- |
| **Graphics** | **Top** | **In-between** | **Bottom** |
| Tables | End | Last | First |
| Figures | Good | Similar | Very well |

## References and Citations

Footnotes should be Times New Roman 9-point, and justified to the full width of the column.

Use the “ACM Reference format” for references – that is, a numbered list at the end of the article, ordered alphabetically and formatted accordingly. See examples of some typical reference types, in the new “ACM Reference format”, at the end of this document. Within this template, use the style named *references* for the text. Acceptable abbreviations, for journal names, can be found here: <http://library.caltech.edu/reference/abbreviations/>. Word may try to automatically ‘underline’ hotlinks in your references, the correct style is NO underlining.

The references are also in 9 pt., but that section (see Section 7) is ragged right. References should be published materials accessible to the public. Internal technical reports may be cited only if they are easily accessible (i.e. you can give the address to obtain the report within your citation) and may be obtained by any reader. Proprietary information may not be cited. Private communications should be acknowledged, not referenced (e.g., “[Robertson, personal communication]”).

## Page Numbering, Headers and Footers

Do not include headers, footers or page numbers in your submission. These will be added when the publications are assembled.

# TEST ENVIRONMENT

We used a Cisco C210 M1 server with two Intel Xeon E5540 (2.5GHz) processors for a total of 8 logical CPUs (hyperthreading was turned off). The system had 72GB of RAM and sixteen 146GB SAS 6.0gbps drives in a single RAID5 configuration. VMware vSphere Hypervisor ESXi 4.1.0 348481 was the host operating system with 9 underlying virtual machines all running Ubuntu Server 10.10 64-bit Linux. The master node was allocated 2GB of memory while eight worker nodes were allocated 8GB of memory each. All nine virtual machines has 1 virtual CPU and 200GB of disk storage. Access to the cluster from the Internet was limited to ssh and https to the master node only. All worker nodes had an internal IPv4 network on a private vlan on the host machine only. The master communicated with the workers over TCP sockets and through ssh remote commands.

The master node has Oracle Java 1.6.0\_24 and Apache Tomcat 7.0.11 installed. It also hosted an NFS4 network share to the worker nodes for sharing common code binaries. The actual rainbow table was hosted on local disk for each node to provide better performance during loading of the node software.

# SECTIONS

The heading of a section should be in Times New Roman 12-point bold in all-capitals flush left with an additional 6-points of white space above the section head. Sections and subsequent sub- sections should be numbered and flush left. For a section head and a subsection head together (such as Section 3 and subsection 3.1), use no additional space above the subsection head.



Figure 1. Insert caption to place caption below figure.

.

## Subsections

The heading of subsections should be in Times New Roman 12-point bold with only the initial letters capitalized. (Note: For subsections and subsubsections, a word like *the* or *a* is not capitalized unless it is the first word of the header.)

### Subsubsections

The heading for subsubsections should be in Times New Roman 11-point italic with initial letters capitalized and 6-points of white space above the subsubsection head.

#### Subsubsections

The heading for subsubsections should be in Times New Roman 11-point italic with initial letters capitalized.

#### Subsubsections

The heading for subsubsections should be in Times New Roman 11-point italic with initial letters capitalized.

# ACKNOWLEDGMENTS

Our thanks to ACM SIGCHI for allowing us to modify templates they had developed.

Special thanks to Cisco for loading us the necessary hardware and Internet connectivity to run our cluster for testing.

Thanks to our professor Richard Han for guidance during our project for his class.

Acknowledgment of the great work done by Joshua Wright, the original developer of coWPAtty from which our work is based.

# REFERENCES

1. Bowman, M., Debray, S. K., and Peterson, L. L. 1993. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst.* 15, 5 (Nov. 1993), 795-825. DOI= <http://doi.acm.org/10.1145/161468.16147>.

Columns on Last Page Should Be Made As Close As Possible to Equal Length