Global Information Assurance Certification Paper

This is a GIAC Gold Template

Watermarks: An In-Depth Discussion

GIAC (GCIA) Gold Certification

Author: Allison Nixon, nixon.nixoff@gmail.com
Advisor: Manuel Humberto Santander Peláez

Accepted: XXXXXXXXXXX

Abstract

Watermarks are used as a way to retain control of information in a world where information is uncontrollable. They are a type of metadata that is inserted into data so the creator of the watermark can have a measure of control over how this data is used, viewed, and distributed. Watermarks can be human readable or machine readable, and they can be visible or invisible. Sometimes watermarks have flaws. They can be stripped out of the data or altered.

Introduction

In a world of general purpose computing, the person that possesses a piece of data has complete control over it. They can distribute it, copy it, or alter it. This can be a problem for content creators, which have an interest in retaining control over their data after it has left their private network and reached the wider Internet. Ever since data itself has been a product, creators have attempted to exert control over that product even after it has left their hands. While legislative measures are often attempted, technical measures are also used. Watermarks are one of the technical measures available to content creators.

Watermarks were originally used by paper makers to identify and assert ownership of their product. The first known watermark was produced in 1282 in Italy. Metal rollers or stamps were pressed into the paper while it was being produced so that the result had a visible mark that was very difficult to duplicate (Meggs, 1998). This allows the creator to assert their association with the physical product, even long after the original sale, and these same kinds of techniques are used nowadays for currency and financial documents.

In our modern age, "watermark" is a broad term used to describe a secondary channel for data to exist on top of the actual product. They can be used for tracking, such as when the watermark identifies who it was originally distributed to. They can be used to assert authenticity, as demonstrated by the numerous anti-counterfeiting measures in currency. They can be used to assert ownership, when photography studios watermark their product so people know where it came from no matter what transformation is applied to it. In these cases and others, watermarks are fundamentally used by the creator to retain control over a product even after it has left their hands.

Depending on the purpose the watermark fulfills, the design of the watermark must accommodate these needs. Robustness, Perceptibility, and Capacity are broad terms that could be used to describe many different types of watermarks. Every watermark has different amounts of each attribute depending on the needs it fulfills.

A watermark is said to be robust if it survives change. If data is duplicated in a lossy manner, a robust watermark will still be readable despite the alterations to the underlying data. On the other hand, a watermark can be deliberately designed to lack robustness for tamper detection purposes. An example of this is commonly seen in

physical financial documents. When they are copied, the copier does not create an exact duplicate, and the inevitable minor changes in the copy causes it to look vastly different from the original. The watermarks are altered in the process, typically covering the duplicate in “COPY” or “VOID”. In other cases, a watermark within a product might not be acting in the user’s best interests, so the user might want to remove the watermark. A robust design will ensure that the user will not be able to alter or remove the watermark from the product they possess.

Perceptibility is another aspect of watermarks. A watermark can be designed to be imperceptible if it does not contribute to the use of the underlying content or otherwise must be secret. Typically imperceptible watermarks are stored in metadata, or inserted into the content data in subtle ways- often using steganographic techniques. This is more often seen in digital media content where identifying information embedded in the media is almost always against the user’s interest. Highly perceptible watermarks, on the other hand, make their presence well known. Photography studios often produce watermarks like this to protect their photos and ensure their studio gets the advertising if their photo is popular.

Capacity describes how much data a watermark could carry. The quantity of data is limited by the efficiency of the encoding method and the size of the underlying content that carries the watermark. Any watermark inserted into a movie could potentially carry far more information than a watermark cleverly hidden within a single photograph.

Sometimes watermarks can have weaknesses. If a watermark is designed for copy protection, then it would be a weakness if the watermark could be removed. If a watermark is designed to track the recipient for copy-protection purposes, then it’s an even worse weakness if someone is able to alter the data so another recipient’s information is there instead.

Proper design of watermarks therefore needs to take into account the possible attacks used against it, but the design should not get in the way of normal use of the underlying content.

Real-world uses for watermarks

To assert ownership by the creator

Watermarks used to assert ownership typically need a high level of robustness and visibility. The creator wants every viewer to know exactly who created this content. A

well designed watermark for this purpose should be highly visible on the product, but should not detract from the function or aesthetics of the product.

1.1.1. Photography studios

Photography studios have remarkably little control over the content they release over the Internet. Copying and modifying images is done very easily and options for copy protection are limited. Typically, studios base their business model on trying to sell images, and the watermarks they use allow them to distribute samples without undermining their business model. The watermarks used by photography studios are typically very prominent, contain the name of the studio, and most important of all, are very difficult to remove without damaging the underlying content. Often images are mass watermarked with the use of automated scripts. In the below PHP code example, a watermark is generated from text and is applied to the image. The PHP code was written specifically for this example, and it uses the GD library which is not included with PHP5 by default.

Before and after:


```
<?php
$text = "George Washington";
$font = '/usr/share/fonts/truetype/freefont/FreeSerifBoldItalic.ttf';
$size = 25; //pt
$watermark = imagecreatetruecolor(276,32); //create empty image
imagesavealpha($watermark, true);  //allow transparency for this image
$background = imagecolorallocatealpha($watermark, 0, 0, 0, 0); //define color black
imagecolortransparent($watermark, $background); //set black as transparency color
imagefilledrectangle($watermark, 0, 0, 276, 32, $background); //fills empty image with black(transparent)
pixels
$color = imagecolorallocatealpha($watermark,255,255,255,0); //define text color(white)
imagettftext($watermark,$size,0,0,24,$color,$font,$text);//write white text into the watermark
$unWatermarkedImage = imagecreatefromjpeg("GWashington.jpg");//load the image to be watermarked
```

```
 imagecopymerge($unWatermarkedImage, $watermark, 260, 247, 0, 0, 276, 32, 50); //apply the watermark
 to the image
 //The remainder of the code outputs the image
 ob_start();
 imagejpeg($unWatermarkedImage);
 $WatermarkedImage = ob_get_contents();
 ob_end_clean();
 echo '<img src="'.base64_encode($WatermarkedImage).'" /><br><br>';
?>
```

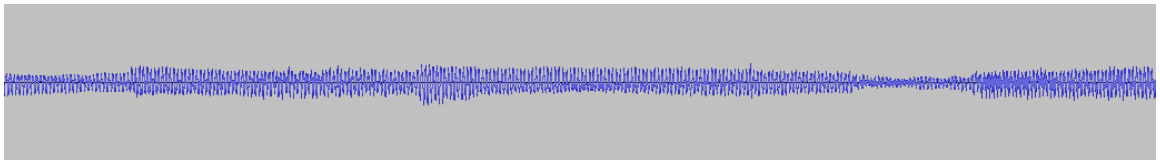
To Prevent Piracy

Content creators have an interest in maximizing their income and often they try to accomplish this by discouraging piracy. While they do employ legal measures to accomplish this, they also employ technical measures to assist as well. Identifying the parties responsible for facilitating piracy is one important step in efforts to combat piracy, as sometimes it can be difficult to identify the pirate. Watermarks are one of the measures used to either identify individuals or gather statistics on where piracy happens.

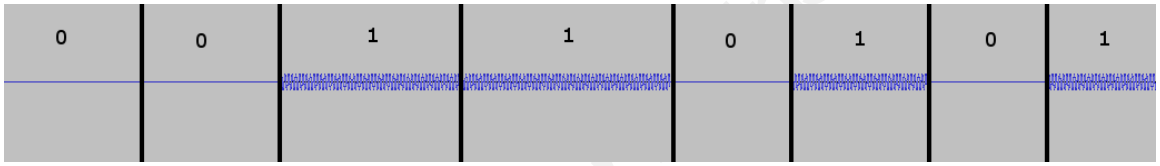
1.1.2. Watermarking in Audio

Watermarks embedded in digital music files can be individualized to the recipient to detect when someone is sharing music and identify who they are. The theory behind this identification scheme is that the rights holder can download the watermarked file from P2P networks, forensically analyze the file, and identify the person who purchased the file originally. Audio watermarking schemes are engineered to be robust, as the digital file may be recorded into an analog device over the air, or digitally recompressed, and information will be lost. A sufficiently robust watermark will remain mostly intact despite these alterations to the music file. There are watermarking schemes which encode information in the waveform of the music itself. Waveform encoding would be robust enough to survive playback over the air. Below is a simplified example of how modifying the waveform would work. The waveform is a screenshot of a couple of seconds taken from a song called “Blind Willie” (Towns, n.d.), using Audacity:

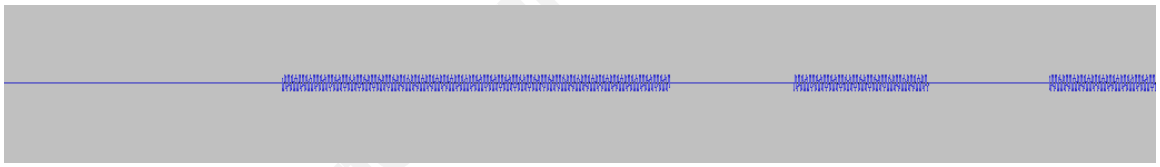
First, we begin with the music we want to watermark:



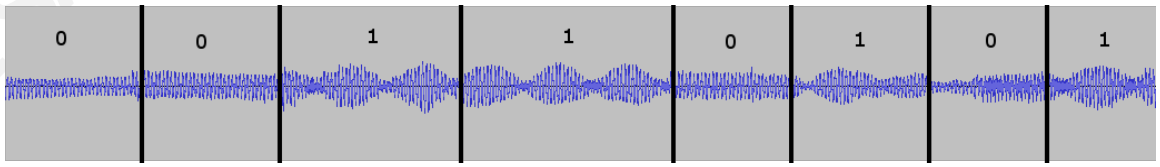
We want to embed the number “5” as a watermark in this music. Breaking it down into binary makes it easier to encode, as we would only need to convey an “on” and “off” state, and would not need to worry about anything in between. A constant tone of a known frequency and volume is prepared. The binary equivalent of 5, “00110101” is encoded into the tone, using a simple method where a “0” emits no sound, while a “1” plays at full volume. The encoding process was done manually for demonstration purposes, but real-world uses would encode this programmatically. The points in time where the watermark occurs is also a known fact which would be used in the encoding and decoding process. The below image is the watermark carrying the “00110101” data, clearly marked out for demonstration purposes:



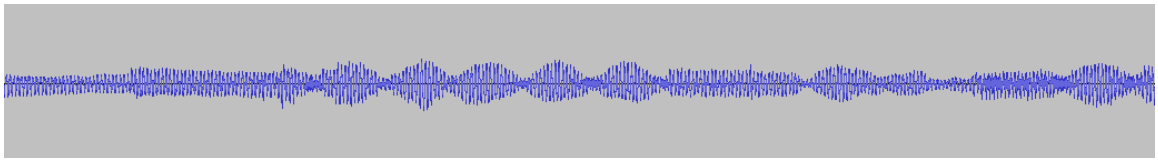
This is the watermark waveform without the markings:



From there, we will use audio processing software to add the watermark to the music clip. We will use a simple additive process (Using audacity’s Tracks > Mix and Render function to merge the watermark track and the music track) to ensure no information is actually lost. Note how the waveform is altered in regions carrying a “1” bit, and the waveform is not altered in regions carrying a “0” bit. Below is the waveform with each section explicitly marked with the bit of data that it carries:



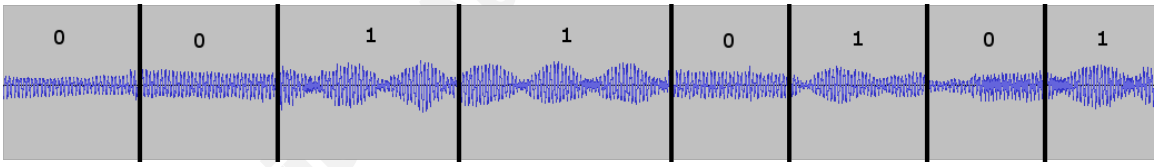
And here is the final watermarked waveform without the markup:



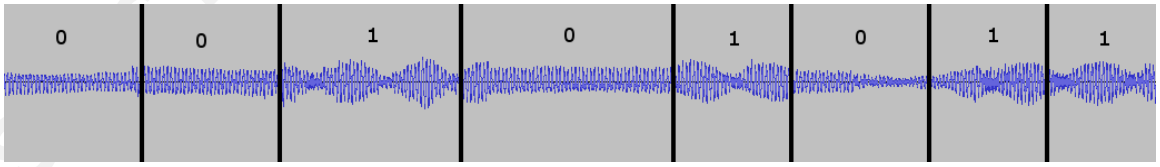
The resulting file contains both the music and the watermark. If the creator wants

to extract the watermark information, all they need to do is retrieve the original, non-watermarked audio file and subtract it from the marked copy. In Audacity, this is accomplished by inverting the waveform of one of the tracks(Effect > Invert) and adding them together using the "Tracks > Mix and Render" function. After this process(effectively subtracting one audio clip from the other), the only audio left should be the watermark encoding the data. Even if the audio was played back over the air and re-recorded, background noise could be easily filtered out since the data-carrying signal is of a known frequency and volume. This can be accomplished with Audacity's "Effect > Equalization" filter, where all audio can be filtered out except audio in the narrow frequency range used by the watermark.

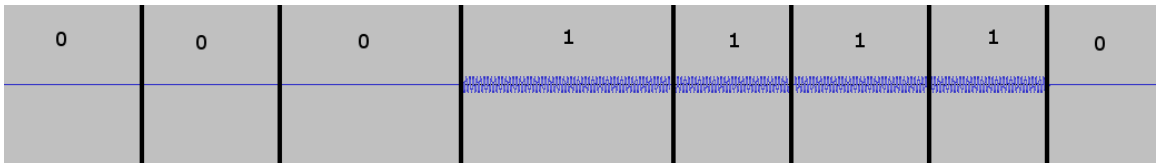
Real-world examples of watermarking by altering the waveform typically involve large amounts of obfuscation or encryption, but the basic principle remains the same. The attacks against it remain the same as well. Any sort of per-user watermarking scheme is vulnerable to a collusion attack, where two separate users with two separate watermarks compare notes. Consider the following. First is the waveform encoding "5" (00110101), produced in an identical manner to the waveform shown earlier.



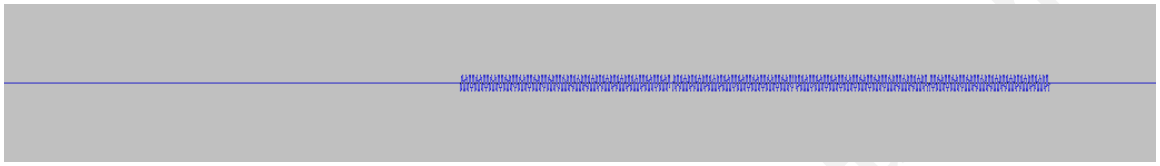
Compare it against the waveform encoding "+" (00101011), encoded in the same manner.



If we were to invert the waveform of one(using Audacity's "Effect > Invert" function), and then add them together (using Audacity's "Tracks > Mix and Render" function), effectively subtracting the differences between both waveforms, we would remove the original music as well as any watermark signal where the bits differ. The result of this transformation would be the following waveform encoding "00011110", again marked up for demonstration purposes:



Here is the waveform without the markup:



This effectively performs a logical XOR operation with the two different watermarks:

Where “5” = “00110101” and “+” = “00101011”

00110101 XOR 00101011 = 00011110

If more watermarks were included in this collusion attack, the same technique can be used to discover every point in the song where a “1” or “0” watermark bit exists. Once this information is known, the attacker can substitute every piece of audio representing a “1” with a “0”, effectively making the watermark read “00000000”, destroying any information carried by the watermark.

1.1.3. Watermarking in Metadata

Other per-recipient watermarks are simpler in design, such as those employed by Apple’s iTunes store. Every audio file download contains the real name and apple ID of the user in the metadata. Below is an example audio file that was legally downloaded using iTunes, and the results when simply using grep on the contents of the file. Grep was used because the information is available in plain text in the file. The name used on the account is “Allison Nixon” and the @yahoo.com e-mail is the appleID. The date of purchase is also present.

```
$ grep  --binary-files=text "Allison" 01\ We\ Three\ Kings\ \(feat.\ Mary\ J.\ Bli.m4a | hexdump -C
[removed]
00000270  22 6d 89 39 74 6f 6f 6c  50 34 35 32 6d 65 64 69  |"m.9toolP452medi|
00000280  00 00 00 01 6d 6f 64 65  00 00 00 00 00 00 00 00  |....mode........|
00000290  00 00 00 00 00 00 00 00  00 00 00 00 00 00 01 08  |................|
000002a0  6e 61 6d 65 41 6c 6c 69  73 6f 6e 20 4e 69 78 6f  |nameAllison Nixo|
000002b0  6e 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |n...............|
000002c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
[removed]


$ grep  --binary-files=text "yahoo" 01\ We\ Three\ Kings\ \(feat.\ Mary\ J.\ Bli.m4a | hexdump -C
[removed]
000004c0  24 64 61 74 61 00 00 00  01 00 00 00 00 32 30 31  |$data........201|
000004d0  32 2d 31 31 2d 32 37 54  30 38 3a 30 30 3a 30 30  |2-11-27T08:00:00|
000004e0  5a 00 00 00 2a 61 70 49  44 00 00 00 22 64 61 74  |Z...*apID..."dat|
000004f0  61 00 00 00 01 00 00 00  00 ** ** ** ** ** ** **  |a........*******|
00000500  ** 40 79 61 68 6f 6f 2e  63 6f 6d 00 00 00 5a 63  |*@yahoo.com...Zc|
00000510  70 72 74 00 00 00 52 64  61 74 61 00 00 00 01 00  |prt...Rdata.....|
00000520  00 00 00 e2 84 97 20 32  30 31 32 20 54 68 65 20  |...... 2012 The |
00000530  56 65 72 76 65 20 4d 75  73 69 63 20 47 72 6f 75  |Verve Music Grou|
00000540  70 2c 20 61 20 44 69 76  69 73 69 6f 6e 20 6f 66  |p, a Division of|
00000550  20 55 4d 47 20 52 65 63  6f 72 64 69 6e 67 73 2c  | UMG Recordings,|
00000560  20 49 6e 63 2e 00 00 00  1c 63 6e 49 44 00 00 00  | Inc.....cnID...|
00000570  14 64 61 74 61 00 00 00  15 00 00 00 00 22 6d 89  |.data........."m.|
[removed]
```

A purchaser could theoretically make unlimited copies of the watermarked file for their own use. If, however, the user uploaded the file to a filesharing network, then Apple would know who did it and could take legal action. This, of course, makes the assumption that the file was not stolen from the original purchaser somehow.

1.1.4. Watermarking In Movies

Watermarking in movies is similar to watermarking in audio, except there is a larger amount of data that the creator could hide a potential watermark in. Movie producers have sued file sharers while using per-recipient watermarks as evidence. In Flava Works, Inc v. Kywan Fisher, a pornography studio filed a copyright infringement suit against one of their customers when they found their own movies circulating on filesharing networks. The plaintiff asserted that their videos were watermarked with a unique code that could identify the purchaser, and by comparing the shared video with their private original, they could extract the identifying watermark. Because of this unique watermarking, they were able to verify every appearance of the video on file sharing networks and link it to the defendant. The code, “xvyynuxl”, was stated in the court documents as the watermark that was present in every video download and tied to the user’s account, however the court documents did not provide technical details about the encoding or extraction method. While the lawsuit did end in a default judgment, and the merits of this watermarking scheme were therefore not tested, it is important to note that these content creators consider watermarking as viable ammunition in their legal

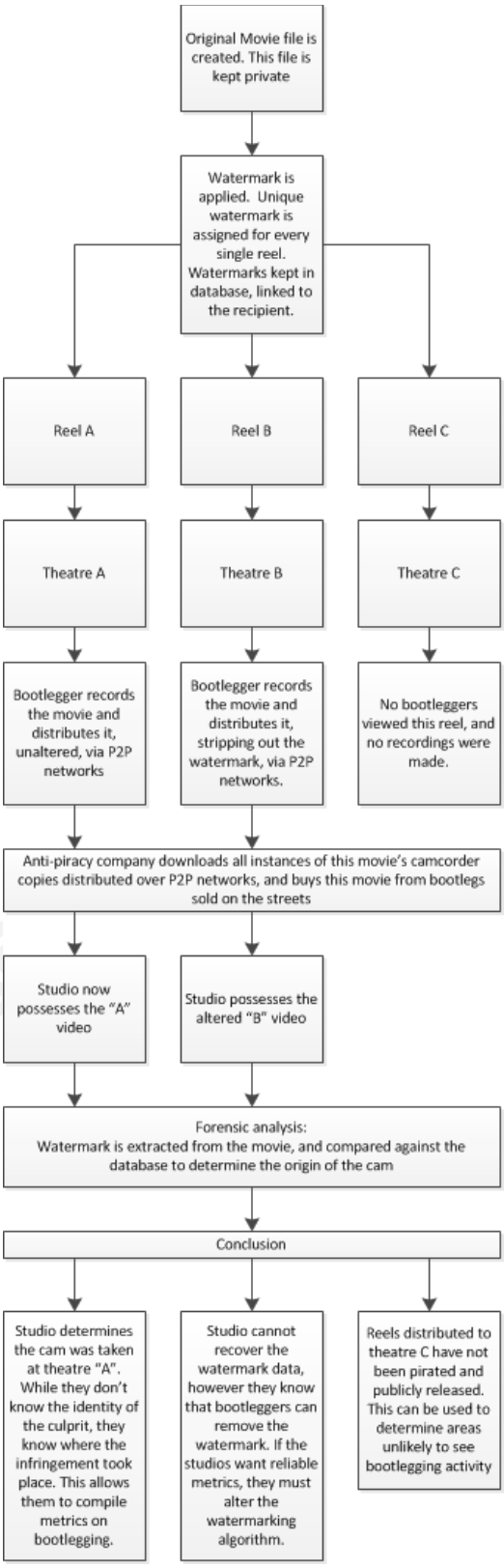
arsenal against file sharers (Flava Works, Inc v. Kywan Fisher, 2012). From a technical standpoint, the watermark involves hiding a very small amount of data in a relatively large amou(nt of content. Such watermarks have the potential to be imperceptible. Locating and removing 8 hidden bytes of data within several megabytes of video would be very difficult if the algorithm is not already known or a collusion attack is attempted.

Movies distributed to theatres use another method of watermarking. Since cams[1] are seen as a problem by movie studios, they can attempt to prevent it using various methods, such as policing the theatres, or watermarks such as the Cinefence watermarking product. The product embeds the watermark in both audio and video content in a format which is robust enough to survive any known compression algorithms ("Cinefence... forensic watermarking," 2007). They insert a unique watermark into the films they distribute to theatres ("Types of content," n.d.). Since the data is transferred over the air and saved in an extremely lossy camcorder format, the watermarks must be very robust so that even the poorest quality cam still carries the watermark information. Watermarks in digital files could reside in the metadata, but watermarks in this case must be carried by altering the audio and video of the movie. While the audio watermarking is some variant of the waveform watermarking method described above, the video watermarking relies on visible marks added to the video stream, as demonstrated by the below screenshot. Note the “T” formed by dots("How the mpaa," 2006):

[1] bootleg videos made from camcorder recordings distributed over p2p networks

Despite this visible appearance, it also needs to be imperceptible so it does not ruin the experience for paying viewers. Typically these dots only appear for one frame. The below diagram constructed with Visio explains how per-theatre watermarking can be used by movie studios to compile metrics on cam bootlegs, and locate “trouble” theatres where the activity occurs often:

Author Name, email@address



("Types of content," n.d.) (August, 2009)

Author Name, email@address

To Prevent Data Exfiltration

Watermarks can also be used to mark files of interest entering and leaving a network. Even with proper access controls in place, an employee could misuse the data and it could end up leaked or stolen by a competitor. Also, a trojan or hacker may steal confidential documents and upload them to a drop server. Unauthorized syncing and backup could also occur, resulting in confidential documents existing outside of the company's control. If hacking malware prevention efforts fail, detecting the presence of a watermark in an outbound document could be the last chance a company has at detecting malicious activity.

1.1.5. Detecting Data Exfiltration with Snort

Network intrusion detection systems on the market are already well suited for inspecting traffic for irregularities. Snort is an open source IDS[2] that will be used for this demonstration. Snort allows users to write custom signatures to add to any existing ruleset, so if an administrator notices a pattern of traffic they want to know about, they can write their own signature and deploy it on their Snort devices ("Snort official documentation," 2012). While custom signatures are typically written to deal with new malware threats, this feature could also be used to detect data exfiltration. In the below example, a watermarked document is exfiltrated and the IDS detects the activity. "THIS DOCUMENT IS CONFIDENTIAL" is the watermark for demonstration purposes; however in a real world situation the watermark would likely be obfuscated.

Example confidential document as displayed in the terminal:

```
$ cat completely\ confidential.txt
THIS DOCUMENT IS CONFIDENTIAL


Here is the secret recipe for the 11 herbs and spices we put in our top
secret soda formula.  This information is not to leave the corporate
network on penalty of firing.
```

The document contains the watermark string, so if it is sent over the network, a properly configured Snort device should pick up on it. If snort was in inline mode, it could even block the packet containing the watermark, potentially disrupting the entire transmission of the confidential document ("Snort official documentation," 2012).

[2] IDS: Intrusion Detection System

To detect transmission of this document, the following command is issued to start up Snort:

```
sudo snort -i eth0 -c rules.txt -A cmg -S HOME_NET=any -S EXTERNAL_NET=any -k none -K ascii
```

A breakdown of the definition of each keyword is as follows

"I eth0" defines the network interface that Snort is listening on. Snort will inspect all unfiltered packets sent and received via this interface.

"-c rules.txt" directs Snort to load this rules file, containing custom rules for watermark detection.

"-A cmg" directs Snort to display alerts in the terminal instead of logging them to a file. This is for demonstration purposes.

"-S HOME_NET=any -S EXTERNAL_NET=any" since Snort is only reading from the custom rules file, the home and external networks need to be defined for Snort to run. Proper configuration in a real environment should have HOME_NET set to the internal corporate network as well as any public IPs under the protection of the IDS, and EXTERNAL_NET should be set to everything else.

"-k none" This allows Snort to inspect outbound packets. The setup used for demonstration purposes involves requests being sent from the same machine that Snort is running on. Snort receives packets created by the machine before checksums are computed, so all checksums will fail and the packets will be ignored unless this setting exists (Kirk, 2010).

"-K ascii" This setting displays packet captures in ascii format for demonstration purposes.

The custom rules file contains this custom rule that was written to detect the presence of the watermark in the file:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Confidential document detected!"; content: "THIS DOCUMENT IS CONFIDENTIAL"; sid:1000002;)
```

A breakdown of the definition of each keyword is as follows:

"alert" If this rule detects packets that match, it will fire and record the alert for later analysis

"tcp $HOME_NET any -> $EXTERNAL_NET any" These keywords look for TCP traffic that is going outbound over any port. The rule must specify outbound traffic, because alerts based on internal-to-internal traffic are of no interest. Transfer of confidential documents between company-owned machines is normal and expected.

"msg:"Confidential document detected!";" This is the name of the alert that will be generated if the signature fires.

"content: "THIS DOCUMENT IS CONFIDENTIAL";" This matches on the contents of the watermark. Any packets containing this watermark string will cause Snort to alert, provided it is outbound.

"sid:1000002;" This is the unique signature identifier. Alert names (designated by the msg keyword) can be duplicated, but the sid must be unique for every signature loaded into Snort. According to the documentation, the convention for custom signatures is for the sids to exist between the 1 million and 2 million range, to prevent conflicts ("Snort official documentation," 2012).

Once Snort is running and the custom rules are loaded, the watermarked file was uploaded to a remote webserver in much the same way an attacker or malicious employee would do it. Snort detected this traffic and produced the following alert (slightly modified for readability purposes):

```
01/04-04:09:01.950839  [**] [1:1000002:0] Confidential document detected! [**] [Priority: 0] {TCP}
10.212.89.84:52234 -> 107.20.44.91:80
01/04-04:09:01.950839 12:31:3B:04:56:AA -> FE:FF:FF:FF:FF:FF type:0x800 len:0x3D5
10.212.89.84:52234 -> 107.20.44.91:80 TCP TTL:64 TOS:0x0 ID:36965 IpLen:20 DgmLen:967 DF
***AP*** Seq: 0x7922BC2  Ack: 0xCC623D4B  Win: 0x391  TcpLen: 32
TCP Options (3) => NOP NOP TS: 728218549 3263469672

POST /DropZone/upload.php HTTP/1.1
Host: 107.20.44.91
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:17.0) Gecko/20100101 Firefox/17.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Referer: http://107.20.44.91/DropZone/upload.php
Cache-Control: max-age=0
Content-Type: multipart/form-data; boundary=---------------------------114782935826962
Content-Length: 416
```

```
----------------------------114782935826962
Content-Disposition: form-data; name="uploaded"; filename="completely confidential.txt"
Content-Type: text/plain

THIS DOCUMENT IS CONFIDENTIAL

Here is the secret recipe for the 11 herbs and spices we put in our top secret soda formula. This information is not to leave the corporate network on penalty of firing.
----------------------------114782935826962--
```

A properly configured Snort device is capable of detecting and alerting on watermarks transmitted over the network in plain text. Detailed alerting can aid a potential investigation by showing where the file was sent, as well as a sample of the contents of the time. In the above example, the alert indicates that the confidential file was uploaded to a URI named "DropZone/upload.php" to an unknown IP address. The alert also shows enough of the file sample to determine that this is actually a confidential document and not a false positive. Additionally, the alert also records the source IP address where the traffic originated from. All of the facts uncovered by this Snort alert are vital to any subsequent investigation, and may in itself be enough information to discover the perpetrator and stop any additional damage. Without this Snort alert, the chances of discovering the leak are lower.

This watermark detection technique can be easily defeated by compressing or obfuscating the file before transferring it. Snort does not carve out files from traffic to unzip or deobfuscate them, and string searches are for the most part limited to using regular expressions to search the contents of traffic ("Snort official documentation," 2012).

Attacks against Watermarks

Watermarks often do not act in the user's best interests, and thus will need to be designed to resist attack. If a watermark is vulnerable to attack, then the validity of the entire watermarking operation could be called into question. SANS watermarks are vulnerable to all three types of attacks.

Detection

When a watermark is created to be imperceptible, detection is the first step to attacking a watermark. Often, collusion attacks are the only way to determine if a bit of data is part of the content, or part of a watermark.

If a watermarked file is compared to a clean file, the differences indicate the presence of watermark data. If the watermarking scheme involves a device specifically designed to read and detect watermarks, then it can be reverse engineered and the entire scheme will become known (Felten, 2006). If several different versions of a watermarked file are compared against each other in a collusion attack, most or all of the watermark locations will also be revealed.

1.1.6. iTunes Watermarks

In the below example, a collusion attack is performed against an Apple iTunes audio file to detect the presence and location of watermarks. The song "Silent Night (Lord of My Life)" by Lady Antebellum was used. Even though the audio file was free, it still contained the watermark. The audio file was downloaded by multiple different usernames and also downloaded by the same user multiple times. A comparison of the binary contents of the files could be used to gain a better understanding of the per-user watermarking that is present in the iTunes songs.

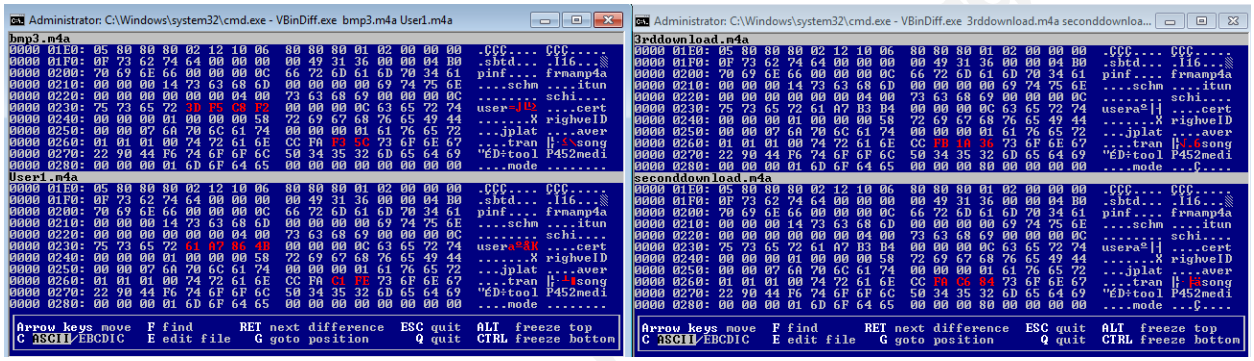
First, md5sums were taken of all the files to find out if there were any differences. The md5sum function is a hashing method that is often used to verify if files are identical or different. A single bit change in a large file will have a drastic change in the final hash value, so for this reason the md5sum function is a good way to detect even minor differences in files before dedicating the time to manually search for differences.

```
$ md5sum *
68a94b0a1e539007871ee1776c889c71  01 User1.m4a
1183c60ecf1bd2e5f1046b880fec684d  3rddownload.m4a
a53a0ebd27ce09fa319f31fa63c8f16a  bmp3.m4a
9163b5014f3f12cac6f43767480d7284  seconddownload.m4a
```

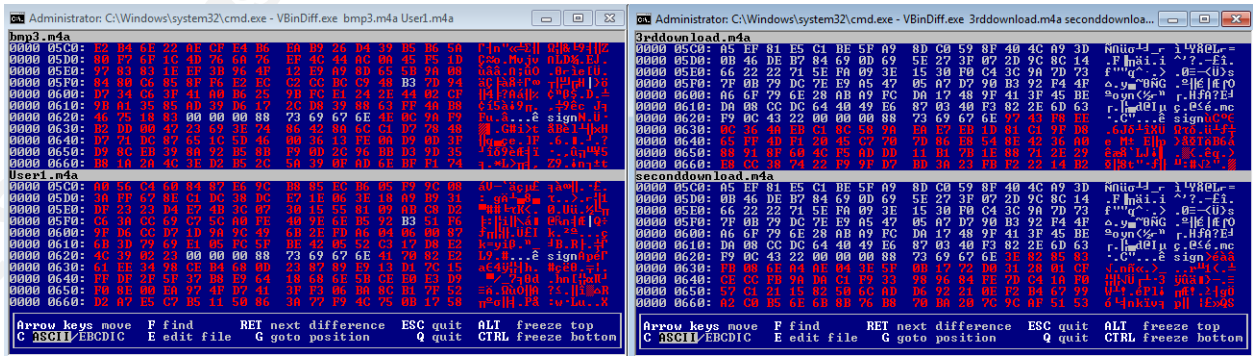
Since every file has a unique hash value, every file has some difference from the others. Standalone audio files don't have any functional need to be dynamic, and in the absence of corruption issues, the most likely explanation for this ever changing hash value is the presence of a watermarking function.

The program VBinDiff is used to compare the differences in the binary files(Madsen, 2008). This program provides a simple way to visually compare differences in binary files. Text that is identical across the top and bottom file is white,

and text that differs is red. In each of the below screenshots, the left side windows will contain data from two different users. In the right side windows, a third user will compare the same file downloaded twice. In every screenshot, the same region of the file is being displayed. This program is only capable of comparing two files at a time, so the highlighting is done between top and bottom files, and the screenshot depicts two instances of VBinDiff running alongside each other for demonstration purposes.



This first comparison shows the same region in the header of each file, and already we can see some differences. There is a difference in the areas marked red between the left and right sides. On the left, the user info is different, whereas on the right, the user info is the same but the download times are different. It is reasonable to surmize that the per-user information is stored in those areas which are white on the right but red on the left, because the user information is the same on the right and different on the left.



The above comparison shows an even more striking display of the differences. The top block of data(which has a size of 623 bytes) differs when the user is different, but is identical when the user is the same. The top and bottom block of data are separated by a “[NUL] [NUL] [NUL] ^sign” string which is identical across all 4 files. It is then followed by 128 bytes of data which differs across all 4 versions. It is reasonable to surmize that the top block of data identifies the user, and the bottom block of data

Author Name, email@address

identifies something unique about the individual download itself. In 751 bytes of encoded data, there is plenty of capacity to carry as much data as needed to identify the user and possibly how they originally purchased the file.

Analysis of all 4 files reveals something interesting about Apple's watermaking scheme. First of all, differences between files were not observed farther than 1700 bytes into the file, and the file was 6.7 megabytes large. This indicates that the watermarks are solely confined to the contents of the headers, and the audio content of the file is not affected. Additionally, the large block of data noted in the second screenshot is the only occurrence of a monolithic block of encoded data. The other differences in the file are either scattered bytes or the user's name and e-mail in plain text, an example of which is displayed from a screenshot of an iTunes music file in section 2.2.3.

Even though the exact contents of the watermark data are encoded, a collusion attack with multiple different users could reveal the presence and locations of a watermark within a file.

Removal

The most obvious attack against a watermark is its removal. When a watermark is no longer present, it can no longer perform its function.

1.1.7. Manual Removal of Visible Watermarks

Watermarks in images are sometimes quite obvious, and image editing software can sometimes be used to remove them with minimal damage to the underlying content, especially if the watermark is present over an area that lacks detailed features, such as any solid color background. In this example we will attempt to remove the watermark applied to the George Washington image in the first example. First, is the image with the watermark:

The Clone Brush in GIMP was used in an attempt to remove the words:

The image ends up mostly intact, however close analysis of the pixels would show that alterations have been made, and the image is not the same as the original image, which is shown below:

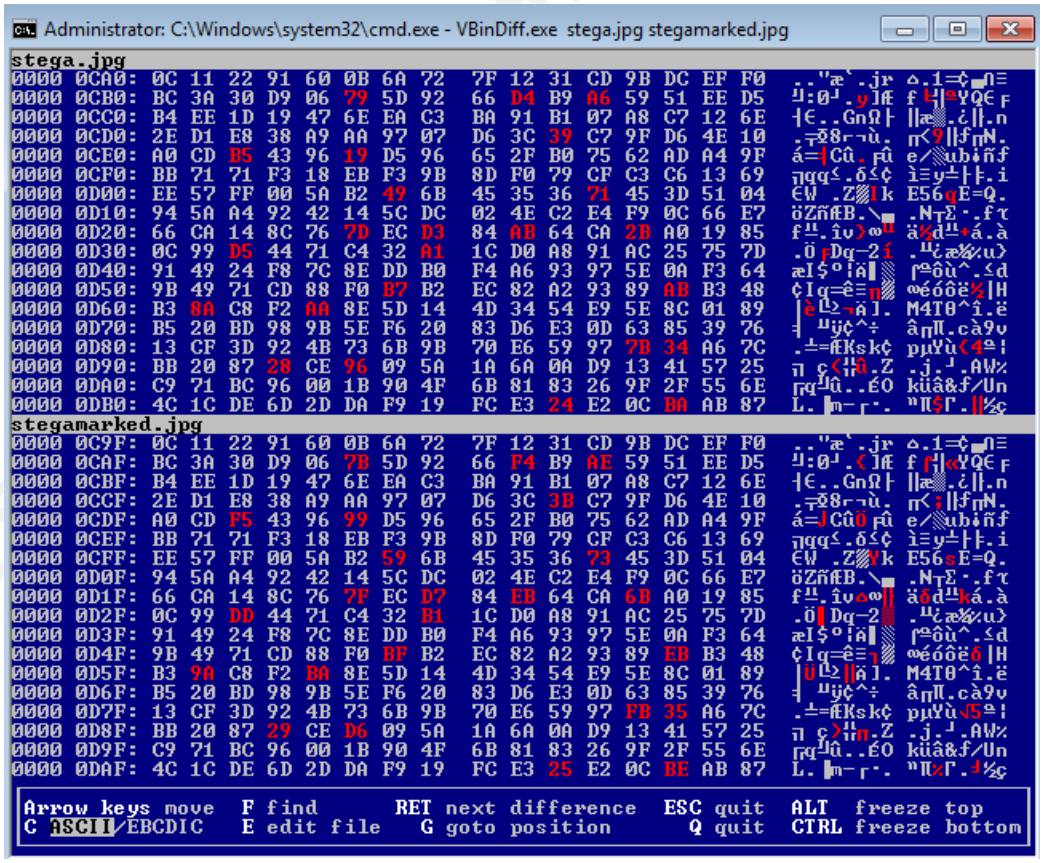
To defend a watermark against this attack, the creator would need to either make it imperceptible, or ensure it's entangled in so much of the underlying content that its removal would destroy the content. If watermarks can be easily removed, then tracking attempts using watermarks will result in incomplete data.

1.1.8. Image Compression

Watermark removal can also be accomplished through normal compression of data. The tool OpenPuff is used to steganographically hide information in images ("Openpuff 4.00," n.d.). The program VBinDiff was also used to demonstrate the presence of watermarks (Madsen, 2008).

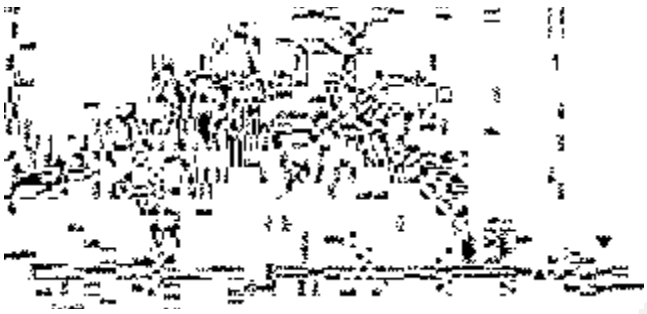
In the below image, a jpg is embedded with the secret message: "Secret Message!". OpenPuff embeds the information in the image without perceptibly altering the contents of the image. The top image is before, and the bottom image is after:

The below VBinDiff screenshot comparing the original to the watermarked version shows that the watermarking process peppers the file with slight alterations. In this case, the minor alterations were present throughout the entire file:



Analyzing the images in GIMP, however, indicates that subtle changes have been made to the actual appearance of the image. Both images were loaded into GIMP as

overlapping layers, and then the layer mode for one was set to “subtract”. Afterwards, the image is flattened. At this point it appears entirely black to the naked eye because all the colors from both images have been subtracted from each other. Using the Bucket Fill Tool with a threshold of 0 will bring the subtle differences to light, as it will paint every truly black #000000 pixel to white, and leave everything else alone. The below image represents every pixel that experienced any change during the watermarking process:

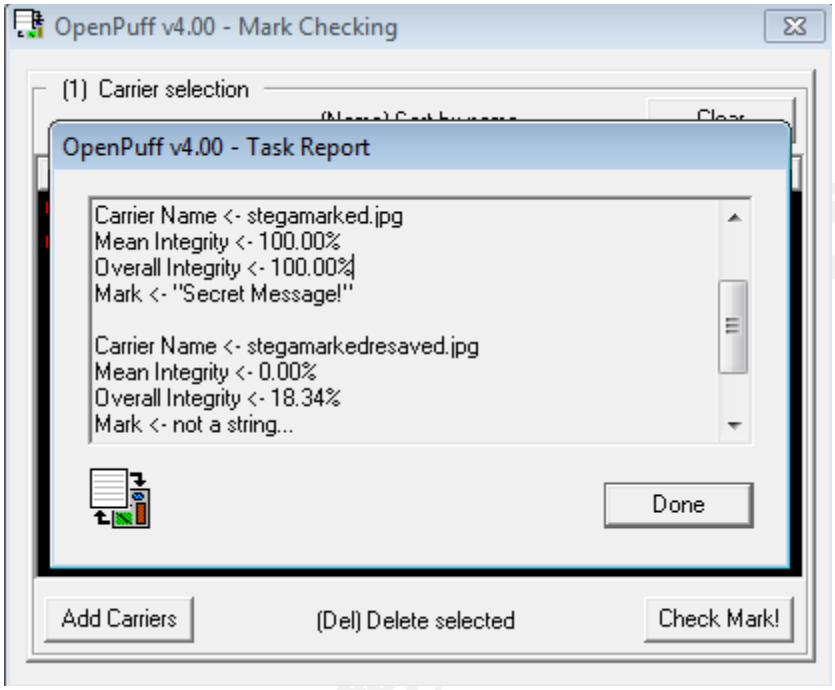


Additionally, for the pixels that were changed, only the lowest order bit for each color value was changed. So if the color is normally 0R 0G 0B(#000000), an altered pixel would have a color of 1R 1G 1B(#010101). Since that is the value of the pixel after the subtraction process, the watermarked image’s pixel would have a color value only the tiniest bit greater than the original value. To the naked eye, this is imperceptable.

Unfortunately for OpenPuff, this sort of watermarking scheme is also very fragile. Since the watermarking scheme relies on subtle changes in color, any lossy compression method will completely destroy the watermark. In the below image, the watermarked image was re-saved as a jpg, and then compared to the original using the same subtractive method. Unfortunately the Bucket Fill Tool can no longer turn the background white in one action, as the entire background is now filled with pixels of a mixture of colors and not many true black #000000 colored pixels. Even so, the differences are obviously much more drastic than the one-value-off changes made by the watermark, and can be seen with the naked eye if viewed at the right angle:

As a final confirmation that the watermark was removed, the OpenPuff tool was

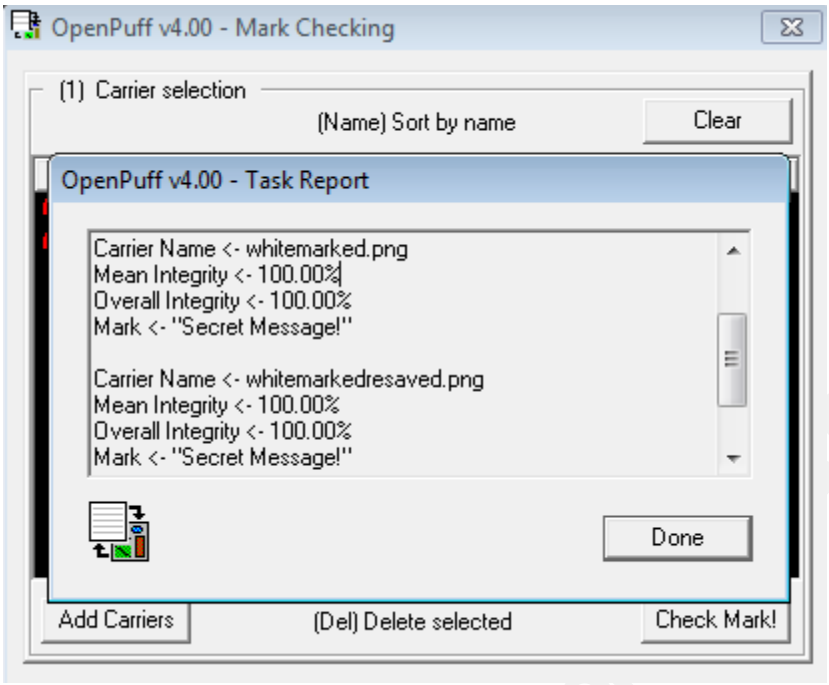
used to read back the watermark values from both images. The first image was the original watermarked image, which read back successfully. The second image is the same watermarked image, but re-saved as a jpg file. The compression process alone caused the watermark readback to fail:



While OpenPuff watermarks are not robust enough to survive jpeg compression, this is not true of all image watermarks. A watermark method can sacrifice data carrying capacity in exchange for the ability to survive large amounts of jpeg compression (Zain, 2011). Therefore, without knowledge of the algorithm used to watermark an image, it is not safe to assume that compression is sufficient to remove a watermark.

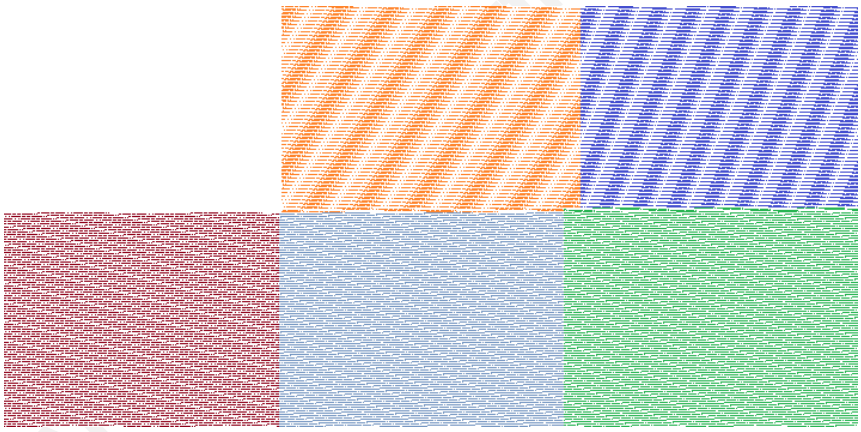
1.1.9. Image Alteration to Disrupt Invisible Watermarks

Not every file format is lossy like jpg, so a simple opening and resaving will not remove the watermark in those cases. PNG is a lossless format. In this example, a png image was created in Paint and watermarked using OpenPuff. When it was re-saved in this lossless format, the watermark remained intact:

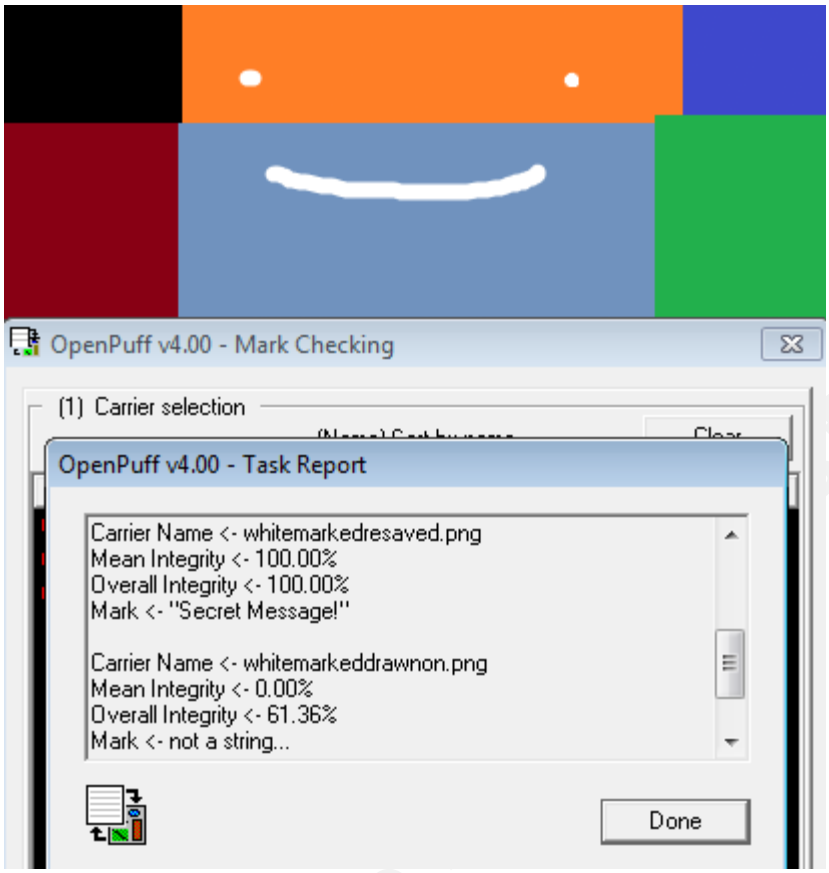


The watermark algorithm for png images is similar in concept to the algorithm for jpg images, where the watermark relies on subtle changes in the colors of pixels themselves. Below, the first image is the original, and the second image is watermarked. The changes are imperceptible:

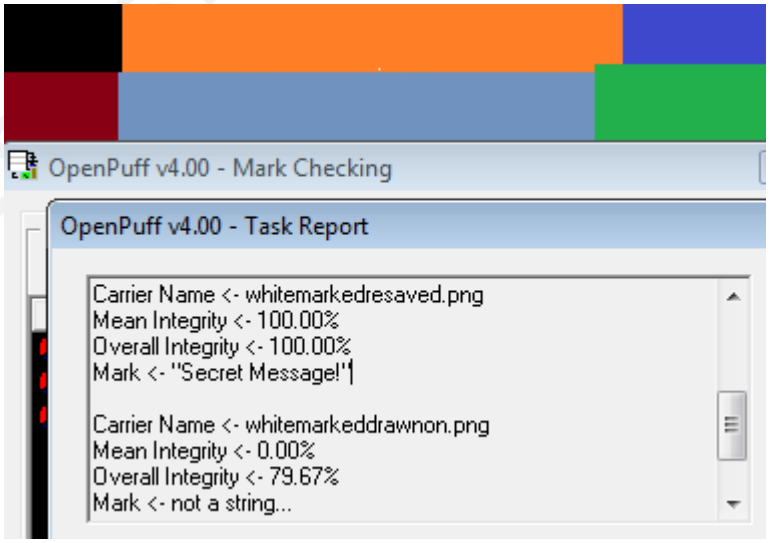
The Bucket Fill Tool was again used to expose the watermark to the naked eye. In this case, it was used once on every color, turning the identically colored pixels white and leaving behind the altered pixels. The algorithm for png watermarking that OpenPuff uses is clearly different from the jpg watermarking algorithm. It is interesting to note that the solid black color did not experience any changes during the watermarking process and therefore turned entirely white during this action:



A few tests were executed to see just what it would take to invalidate the watermark. The screenshots displayed below only show the modified portion of the image due to space constraints. First, a smiley face was drawn affecting a minority portion of the image and two colored areas. It caused the watermark read back to fail:

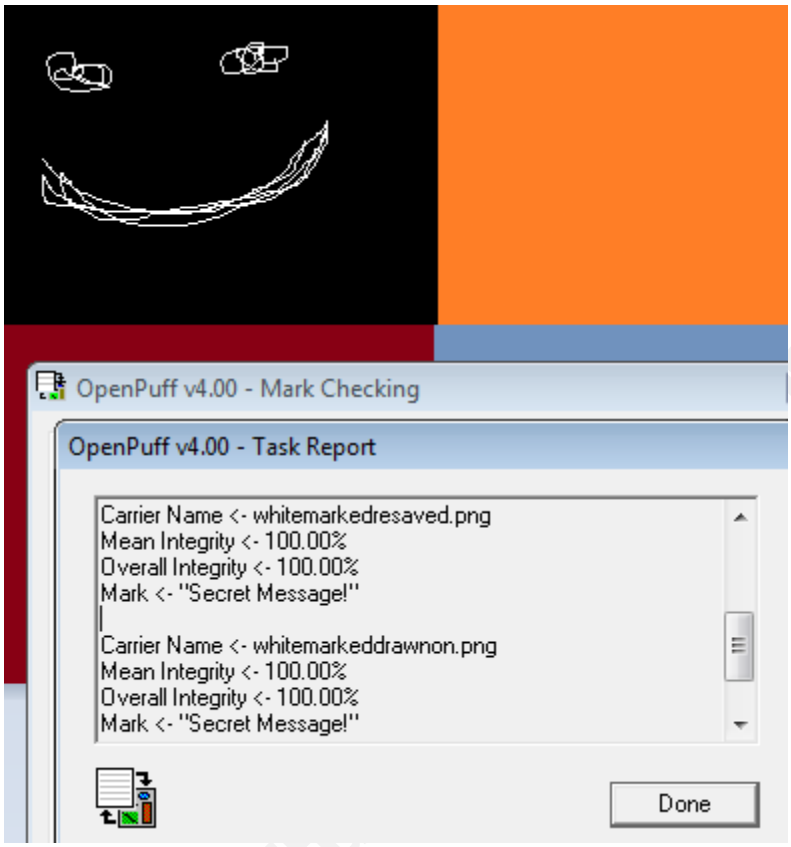


The change was then minimized to a single pixel in a colored area. Even though nothing else in the image changed, the watermark was simply not robust enough to carry information that OpenPuff's readback process could handle, despite reporting back a higher integrity value than the image containing the smiley face:



Alterations in the black region did not pose any problem for OpenPuff, even if

they were significant in size. Since this region was ignored during the watermarking process, it makes sense that it is also ignored during the readback process.



In these examples, the image alterations required to invalidate the watermark are heavily dependant on quirks in OpenPuff's algorithm. The robustness of a watermark is directly related to the robustness of the readback algorithm used by the content owner, and in the case of OpenPuff, it is simply not robust enough to survive any significant alteration.

Forgery

This is the worst case scenario for a watermarking scheme. First of all, this implies that the user can already detect and remove the mark if they wanted to, but instead are going to alter the data to state something false instead. This could be used to skew statistics or undermine anti-copyright-infringement watermarking schemes. To prevent this possibility, watermarks can be encrypted or obfuscated to prevent users from guessing valid watermark values or learning the embedding method. In the case where watermarks are used by devices to control copying and playback, a watermark could be inserted into an arbitrary file to trick the device into thinking it's protected or authentic when it really

isn't (Kutter, Voloshynovskiy & Herrigel, 2000).

1.1.10. SANS Watermarks

SANS watermarks are weak to forgery, due to the fact that they are easy to detect and valid watermark values are also easy to obtain. In this example, a text based watermark is easily forged. When SANS practice test questions are copy pasted, the watermark carries over in the text:

Original(watermark number is 1374364)	Altered(watermark is forged to show 1279694)
(Question 87) 1374364 1X24 (Answer) 27 26 23 24 245	(Question 87) 1279694 1X24 (Answer) 27 26 23 24 245

SANS "math tests" simulate the full test environment, including the watermarking scheme. The watermarks themselves are identical to the exam IDs viewable on a student's practice test dashboard (located at https://exams.giac.org/pages/practice). This watermark is so easy to forge because the contents exist in plain text, and the algorithm used to embed it is obvious.

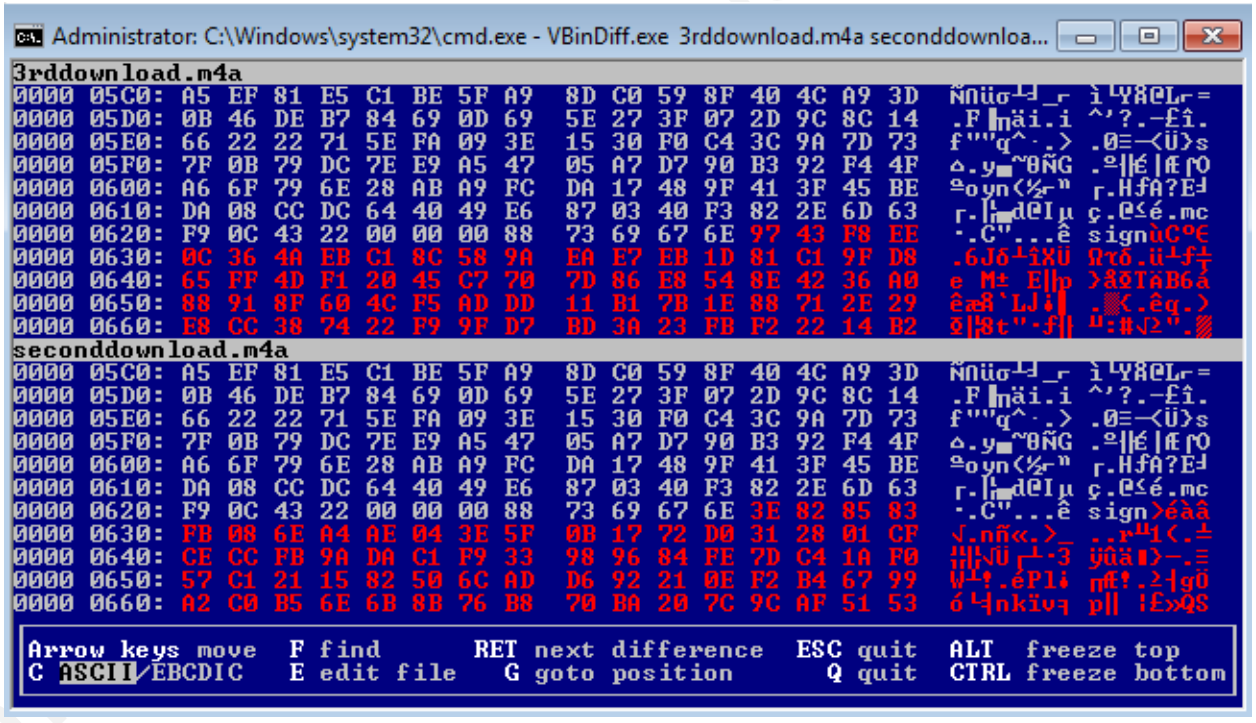
Exam ▲	Certification
Automated Math Test Exam - ID 1374349	--
Automated Math Test Exam - ID 1374354	--
Automated Math Test Exam - ID 1374359	--
Automated Math Test Exam - ID 1374364	--
Automated Math Test Exam - ID 1374369	--
Automated Math Test Exam - ID 1374379	--
GCIA Exam - ID 1279689	GIAC Certified Intrusion Analyst
GCIA Exam - ID 1279694	GIAC Certified Intrusion Analyst

When valid watermark values can be discovered and easily substituted, it allows malicious users to undermine the tracking schemes made possible by content watermarking.

1.1.11. iTunes Watermarks

The below screenshot is from the same iTunes song file mentioned in section

3.1.1. The watermark data contained in this iTunes files is very resistant to alteration as the data is stored in an encrypted format, and the algorithm used to embed that data is complex. Even if a malicious user were to alter the name and e-mail that exist in plain text, they are unlikely to accomplish much if the encrypted data contains information that contradicts their forgery. An attacker would need to reverse engineer the entire watermarking engine to correctly forge a watermark.

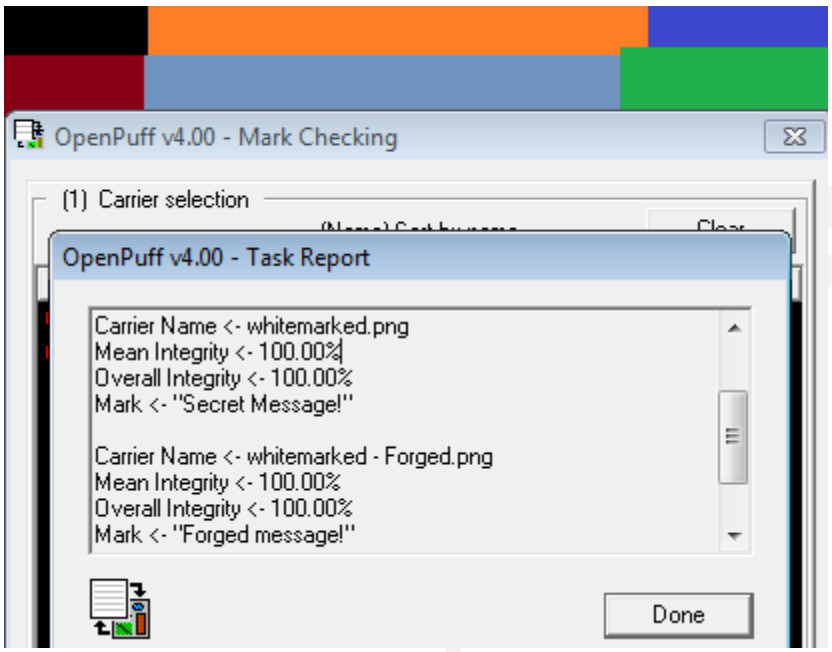


1.1.12. Known Algorithm

The contents of a watermark may not appear obvious when viewing the binary data of a watermarked file, however if the algorithm embedding the data can be guessed, then a watermark can be changed to contain arbitrary content. Relying on the algorithm alone to ensure the integrity of a watermark is like relying on security through obscurity.

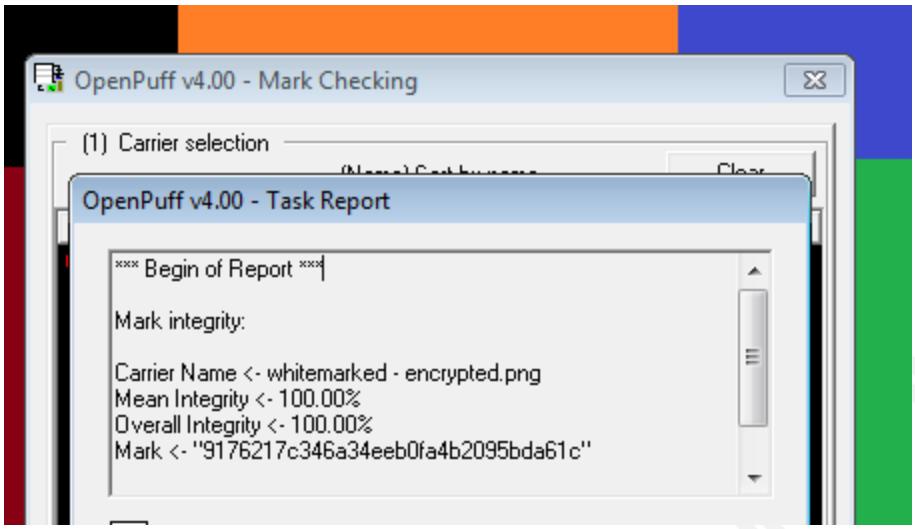
OpenPuff contains a function to remove a watermark from an image called "CleanUp". If it's applied to an image watermarked using OpenPuff, the image will be

reverted to its original unwatermarked state. From there, an attacker could use OpenPuff to watermark the image with their arbitrary messages. In the below screenshot, a watermark was removed using “CleanUp” and the image was successfully watermarked again:



Using steganographic techniques to watermark an image are unhelpful if the plain text can be retreived using the same tool used to encode. The solution is to use encryption on the input, so if an attacker knows the algorithm, they wouldn’t be able to do anything worse than removing the watermark.

In this example, an online AES encryption tool is used to create encrypted watermark contents before supplying it as input to OpenPuff("Aes calculator," n.d.). The watermark message, “this is a secret” is first converted to hex (74686973206973206120736563726574), and a key is created to encrypt the text with (123456789098765432 1abcdefabcdefa). Then the two values are encrypted together and only the output (9176217c346a34eeb0fa4b2095bda61c) is supplied to OpenPuff.



When an attacker attempts to read this watermark, they are presented only with unreadable hex data. If the content creator only accepts watermarks that can be decrypted with their secret AES key, then an attacker does not have enough information to create a convincing forgery, and the content creator can be confident in the integrity of their watermarks.

Conclusion

Watermarks are ultimately a covert channel used to transmit information to further the content creator's interests. If the creator of this information wants to retain control of this covert channel, they will need to design it carefully to withstand attacks. Advanced algorithms and efficient embedding methods can ensure the watermark will be robust enough to withstand detection and alteration. Encryption should be used when possible to ensure the data being carried by a watermark retains its integrity and confidentiality. Watermarks are a powerful tool to aid in tracking the movement of data, and when the watermarking scheme is well designed, it can be used to produce reliable metrics that inform strategic action.

References

Meggs, P. (1998). A history of graphic design.

Types of content theft. (n.d.). Retrieved from http://www.mpaa.org/contentprotection/types-of-content-theft

Author Name, email@address

Flava works, inc v. kywan fisher, 1:12-cv-01888, NDIL, (2012).

Felten, E. (2006, February 24). How watermarks fail. Retrieved from https://freedom-to-tinker.com/blog/felten/how-watermarks-fail/

Kutter, M., Voloshynovskiy, S., & Herrigel, A. (2000). The watermark copy attack. Manuscript submitted for publication, Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.40.9110

Towns, R. (n.d.). Blind willie. Retrieved from http://robtowns.com/music/

Cinefence.. forensic watermarking solutions for digital cinema. (2007). Retrieved from http://www.business-sites.philips.com/shared/assets/global/Downloadablefile/CineFence-13275.pdf

How the mpaa knows where movies are pirated. (2006, October 31). Retrieved from http://torrentfreak.com/how-the-mpaa-knows-where-movies-are-pirated/

August, J. (2009, March 18). [Web log message]. Retrieved from http://johnaugust.com/2009/cams-rips-and-release-dates

Madsen, C. (2008, July 26). Vbindiff — visual binary diff. Retrieved from http://www.cjmweb.net/vbindiff/

Openpuff 4.00. (n.d.). Retrieved from http://embeddedsw.net/OpenPuff_Steganography_Home.html

Zain, J. (2011). Strict authentication watermarking with jpeg compression (saw-jpeg) for medical images. European Journal of Scientific Research, 42(2), 232-241. Retrieved from http://arxiv.org/ftp/arxiv/papers/1101/1101.5188.pdf

Aes calculator. (n.d.). Retrieved from http://testprotect.com/appendix/AEScalc

Snort official documentation. (2012, November 05). Retrieved from http://www.snort.org/docs

Kirk, A. (2010, January 22). "flow:established" rules are never being fired. Retrieved from http://seclists.org/snort/2010/q1/155

Author Name, email@address