ON VARIANT OF FLAME MALWARE

By

Seharish Ajmal

A thesis submitted to the Faculty of Information Security Department, Military College of Signals, National University of Science and Technology, Pakistan in partial fulfillment of the requirements for the degree of Master of Science in Information Security (MS IS).

JULY 2016

SUPERVISOR CERTIFICATE

It is certified that the final copy of thesis has been evaluated by me, found as per specified format and error free.

____

Dr. Mehreen Afzal

ABSTRACT

The collision attack used by Flame Malware was found to be a variant of chosen prefix collision attack. The chosen prefix collision attack was first presented by Stevens, Lenstra and De Weger in 2007. Later on creation of two colliding certificates by Stevens et al, with different distinguished name fields using chosen prefix collision attack was a major breakthrough in the history of collision attacks against MD5.

As the collision attack that is reported to be used for Flame malware is a modification of Stevens et al's original chosen prefix collision attack, using which attackers tried to make two colliding certificates. Exact algorithm for construction of chosen prefix collision attack used by Flame is not known. Therefore this thesis is an attempt to explore and implement the chosen prefix collision attack that was used for Flame malware. Cost and complexity estimates for the subject attack are also discussed. Furthermore Flame's collision attack has been simulated to obtain replacement differential paths which are different from the differential paths of actual chosen prefix collision attack.

Four starting segments for forward replacement differential paths of Flame's collision attack are constructed along with ending segments. Four forward replacement differential paths are also constructed by utilizing the observed characteristics of Flame's differential path from literature.

Modern malware evasion techniques are also explored and counter and preventive measures are recommended as well, specifically a paradigm for detection of weak cryptographic primitives is proposed.

DEDICATION

All praises are due to Allah Almighty who is the Most Merciful and Most Compassionate, Lord of the worlds.

I am pleased to dedicate my work to my great Father for his support, encouragement and belief in me throughout my life and for making me who I am today. To my sweet Mother whose endless love, affection and encouragement made me able to complete my work. To my Teachers who have been a constant source of knowledge, encouragement and inspiration for me all through my research.

ACKNOWLEDGMENT

All praises are due to Allah Almighty who has blessed me the strength and ability to understand, learn and complete my work. May the peace and blessings of Allah be upon the Final Prophet Muhammad, his family and companions.

First and foremost I would like to acknowledge my supervisor Dr. Mehreen Afzal with my deepest gratitude. Regardless of her busy calender, she always made herself available for discussions and queries. Her incessant encouragement, enlightening ideas and valuable recommendations and suggestions kept me determined throughout the work. Without her consistent support, instructions and encouragement, my research work could not have reached its present form.

I am also deeply obliged to my guidance committee members Lt.Col Baber Aslam, Lecturer Waseem Iqbal and Lecturer Waleed Bin Shahid for their persistent support, interest and assistance.

I would also like to thank Lt.Col Muhammad Mubashir Quddoos who is the Head of the Department and his team for administrative help and support.

Lastly I want to give special thanks to my family for their constant encouragement and moral support during my master's studies.

TABLE OF CONTENTS

LIST OF FIGURES

LSIT OF TABLES

Chapter 1

INTRODUCTION

1.1 Overview

With increasing threats to information and communication, information security has gained worldwide importance. It is known that confidentiality, Integrity and availability are three main traits of Information security. However along with securing, authenticity of information is important as well.

So the cryptography also includes the study of methods for proving message authenticity. Message Authentication Codes (MAC) and Digital Signatures are the cryptographic primitives used for authenticating messages and Cryptographic Hash functions provide basis for building these. These Hash functions have certain characteristics such as pre-image resistance, second pre-image resistance, and collision resistance. And the collision resistance is typically the first characteristic to be breached for attacks on hash functions.

There are several collisions attacks performed on hash function, In 2007 the chosen-prefix collision attack was first presented by Stevens, Lenstra and de Weger on MD5[1]. It was published in EUROCRYPT 2007. Flame a very advanced malware was also found to use a variant of chosen-prefix collisions attack on hash function MD5 by Stevens et. al (to abuse the certificate) to deploy itself on systems using windows.

Flame mainly attacked and infected computers working with Microsoft windows operating system. It spread itself using a Windows Update, masked as a Microsoft security patch. Windows use signatures to validate the authenticity of Windows Update (to ensure that the updates are actually from Microsoft). The attackers used chosen-prefix collision cryptanalytic attack against MD5 to obtain a certificate by forging a Microsoft's digital signature that authorised them to be authenticated as a signed update from Microsoft. In chosen-prefix collision attack two message blocks M and M' are extended with post-fixes F and F' in such a way that both M||F and M'||F' produce the same MD5 hash.

1.2 Need for Research

A lot of detail is available about the Flame Virus and its modules and how they work, but the collision attack used by the attackers to abuse windows needs to be demonstrated and researched, so there is a need to implement the supporting collision attack specifically it's differential path construction to show how it is used to set up the malware. Although some research has been carried out to analyze the collision attack behind this malware. But there is a need to make it more clear to general public. Moreover there is need to theoretically discuss the techniques that could be exploited by modern malwares like Flame for their deployment. And is there any solution that despite the exploit of certificate (using modern hashes), we could prevent the damage caused by such malwares? And it is needed to discuss the solutions to thwart other malware evasion techniques as well.

1.3 Problem Statement

As malwares like Flame are a great threat to security because they are used for espionage. Although a lot of explanation is available on Flame malware as well but there is a need to demonstrate the collision attack and specifically the differential path construction mechanism used in this collision attack. Moreover a theoretical analysis is required to further explore that what different exploitation techniques modern malware could use and what potential preventive measures can be taken to thwart such viruses, before their deployment and even after they have implemented the collision attacks to exploit the certificates.

1.4 Objectives

1. To demonstrate in terms of complexity how collision attack is used to deploy Flame malware.

2. To simulate the chosen prefix collision attack in the way Flame malware used it.

3. To assess the use of the attack in terms of complexity for real time attack as as malware's entry point into the system. And to discuss any potential preventive measures to prevent malwares, first from infecting the systems and second from the damage in case if such malwares exploit the certificates and become successful in infection.

1.5 Research Methodology and Achieved Goals

This research has been carried out in three core phases. An in depth study and literature review in regard to Flame's collision attack has been performed as the first phase. By detailed study of literature a sound theoretical knowledge base has been developed about the functionality and complexity of the attack. The second phase consists of complexity analysis of the attack. This analysis provides insights into the strengths and weaknesses of the Flame's collision attack. In this phase malware evasion techniques are discussed that could be exploited by modern Flame like malwares. And possible countermeasures to thwart such techniques are also elaborated. In third phase the implementation of replacement differential paths for Flame's collision attack hash been carried out. Open source C++ code for chosen prefix collision attack is used with some modifications for implementation. The starting segments for partial forward replacement differential paths are created along with particular ending segments.

The ultimate achieved goal is that the partial forward replacement differential differential paths of the Flame's collision attack have been constructed. As the actual algorithm for construction of Flame Attack's differential paths is unknown so the differential path construction algorithm for the chosen prefix collision is used. However the attributes such as the message block differentials δm and δIHV eliminations of Flame collision attack as observed in literature are used for these replacement differential paths' construction.

1.6 Thesis Organization

The Thesis document has been structured in to seven chapters. Chapter 2 presents the concise literature review of all related work regarding this research, found in literature. Chapter 3 contains all the attacks carried out on MD5 hash algorithm including chosen prefix collision attack and their details. The description of Flame's collision attack and it's complexity analysis is presented in chapter 4. Chapter 5 provides the simulation details for constructing the replacement differential paths of the Flame attack. Chapter 6 elaborates the evasion techniques used by modern Flame like advance malwares and their counter/preventive measures. The thesis is concluded in chapter 7 along with suggestions for future work.

1.7 Conclusion

This chapter gives an overivew of Flame malware and it's collision attack. It also discusses the problem statement and objectives of this research work. Research methodology which is used to carry out simulation and goals that are accomplished are also elaborated in this chapter. Lastly it deliberates the thesis report organization.

Chapter 2

LITERATURE REVIEW

2.1 Introduction

In this chapter various collision attacks on MD5 are described that are basically derived from it's Merkle Damgard construction. How collision attacks on hash function MD5 are discovered and improved with time has been reviewed. The chapter has been sectioned into four parts. Section 2.2 gives a description of MD5, in section 2.3 generic collision attacks on MD5 are described, sections 2.4 elaborates identical-prefix collision attacks and section 2.5 presents chosen-prefix collision attacks carried out on MD5 hash.

2.2 Message Digest Algorithm 5

In 1992, Ronald Rivest proposed the hashing algorithm MD5[2]. It is established on Merkle Damgard structure [3] and consists of a compression function. There are two standards to represent 32-bit words *big endian* and *little endian*, the former is comparatively straightforward. MD5 uses *little endian* while working with bytes.

2.2.1 Overview of MD5

MD5 input consists of a 512-bits message and a 128 bit Intermediate hash value (IHV) as a result it produces output hash of 128-bits. Its message processing is done as follows:-

1. Initially a '1' followed by a number of zeroes to make message size equals to 448 modulo 512 bits is appended to the message. Afterwards original size of message is appended as 64 bit little-endian integer thus making its length equal to 512.n, for an integer n.

2. Next padded message M is divided into *'n'* 512 bit blocks i.e. $M_0, M_1, M_2, ..., M_{n-1}$. The 128 bit IHV consists of four 32 bit words *IHV=(a,b,c,d)*, the initial value (IV) is a fixed public value i.e.

$$IV = IHV_0 = (67452301_{16}, efcdab89_{16}, 98badcfe_{16}, 10325476_{16})$$

3. A n block message hashing requires MD5 to process through $n+1$ states IHV. The Compression function of MD5 is used to compute IHV_j (where $j = 1...n$) as follows:

$$IHV_j = MD5Compress(IHV_{j-1}, M_{j-1})$$

4. The last IHV_n, presented as concatenation of the four words a_n, b_n, c_n, d_n, is the resulting MD5 hash.

2.2.2 Compression Function of MD5

The inputs to the compression function of MD5 are a message block B of 512 bits and an initial value IHV of 128 bit, taken as $MD5Compress(IHV, B)$. It is computed in 64 steps enumerated from 0 to 63, these steps are divided in to 4 rounds. The modular addition, left rotation, a bitwise boolean function f_s, an addition constant and a rotation constant are used at every single step. The addition constants are defined by $AC_s = 2^{32}|sin(s+1)|$, The rotation constants are

$$(RC_s, RC_{s+1}, RC_{s+2}, RC_{s+3}) = \begin{cases} (7,12,17,22) \text{ for } s = 0,4,8,12, \\ (5,9,14,20) \text{ for } s = 16,20,24,28, \\ (4,11,16,23) \text{ for } s = 32,36,40,44, \\ (6,10,15,21) \text{ for } s = 48,52,56,60. \end{cases}$$

and the boolean functions for each round are given by

$$f_s(X,Y,Z) = \begin{cases} F(X,Y,Z) = (X \wedge Y) \oplus \overline{X} \wedge Z) \text{ for } 0 \leq s < 16, \\ G(X,Y,Z) = (Z \wedge X) \oplus (\overline{Z} \wedge Y) \text{ for } 16 \leq s < 32, \\ H(X,Y,Z) = X \oplus Y \oplus Z \text{ for } 32 \leq s < 48, \\ I(X,Y,Z) = Y \oplus (X \vee \overline{Z}) \text{ for } 48 \leq s < 64. \end{cases}$$

Sixteen 32 bit words of messages blocks can be derived by dividing a 512-bit message and are expanded into 64 words using following:

$$W_s = \begin{cases} m_s \text{ for } 0 \leq s < 16, \\ m_{(1+5s) \mod 16} \text{ for } 16 \leq s < 32, \\ m_{(5+3s) \mod 16} \text{ for } 32 \leq s < 48, \\ m_{(7s) \mod 16} \text{ for } 48 \leq s < 64. \end{cases}$$

For step s=0..., 63 the compression function $MD5Compress$ maintains four working states of 32 bit words $Q_s, Q_{s-1}, Q_{s-2}, Q_{s-3}$. These states are set up as $Q_0 = b, Q_{-1} = c, Q_{-2} = d, Q_{-3} = a$ for all 64 steps, and Q_{s+1} can be calculated as follows:

$$F_s = (Q_s, Q_{s-1}, Q_{s-2}), T_s = F_s + Q_{s-3} + A_s + W_s, R_s = RL(T_s, RC_s)$$

$$Q_{s+1} = Q_s + R_s$$

After all steps the output hash is calculated as follows

$$IHV_{out} = MD5Compress(IHV_{in}, B) = (Q_{61} + a, Q_{64} + b, Q_{63} + c, Q_{62} + d)$$

2.3 Generic Collision Attacks

In this section comparatively less sophisticated attacks on MD5 are discussed before discussing the complex ones in next sections as they will give a starting point for comprehending the sophistication of the later ones.

2.3.1 Birthday Search

In a birthday search attack if n is the output length of a function/hash then it requires $O(n/2)$ assessments of that function/hash, to find a collision. So modern hash functions are designed keeping this in mind that complexity of the hash function should be good enough to make this attack impractical.

MD5 produces a message digest of length 128 so for a birthday search brute force attack the complexity is 2^{64} which is quite practical.

MD5CRK a distributed computing project was started in March 2004 to perform a birthday search brute force attack on MD5, nevertheless the project ended in August 2004 without declaring it's progress at that time.

2.3.2 Pseudo Collision

MD5 was already shown weak by Boer and Bosselaers in 1992 [4] by presenting a pseudo collision attack on MD5.

This attack used only one message block B and using an IHV along with this message block found another IHV' such that *MD5(IHV, B) = MD5(IHV', B)*. The difference between two IHV's was of following form: *IHV = (a,b,c,d)*, *IHV' = (a*+2^{31} *mod* 2^{32}*, b*+2^{31} *mod* 2^{32}*, c*+2^{31} *mod* 2^{32}*, d*+ 2^{31} *mod* 2^{32}*)*

2.3.3 Semi Free-Start Collision

A semi free start collision attack was announced in 1996 by Dobbertin published in [5] and [6]. It used two distinct messages and a particularly selected initial value to produce collision. However this collision attack was not on full MD5 but it shows that the higher order bit differences diffusion in working states of algorithm is a bit slow. And with this

Dobbertin suggested to relinquish the further use of MD5 based digital signature schemes.

2.4 Identical-prefix Collision Attacks

In 2004 a group of Chinese cryptographers Wang et al. presented identical-prefix collision attack in annual CRYPTO rump session [7], they carried out this identical-prefix collision attack on MD4, MD5, HAVAL-128 and RIPEMD. It was a two block collision. The detailed description of the attack is given in section 3.2. The attack has complexity of 2^{39} and on an IBM P690 it takes 15 minutes to calculate the collision. The details and method used to find the collision was presented in 2005 [8] by Xiaong Wang and Hangbo Yu. However before the publication of their explanation Hawkes et al.[9] also tried to reconstruct Wang's collision attack and described the necessary bit-conditions to satisfy the differential paths constructed by Wang et al.

After the publication of Wang's identical-prefix attack several attempts were made to apply the attack in real world scenario. The first one was presented by Dan Kaminsky [10] in which he created two colliding archives using the colliding messages presented by Wang et al. and extracted two different files with same MD5 hash, all this was done using a special tool he called Stripwire. This attack was targeted to file-system auditing tool Tripwire, that uses hashes of the files to detect unauthorized changes to the files.

The unreliability of digital certificate setting was demonstrated by Lenstra, Wang and de Weger in [11] by creating two X.509 digital certificates consisting of distinct public keys but same signatures. One of the complex applications of the attack was creating two programs with same hash but different behavior, Daum and Lucks [12] presented such a demonstrative application in which they constructed two postscripts files with same MD5 hash but their content was totally different. Looking into any of these postscript files in text or hex editor the postscript code can be seen and it gives an idea about the mysteriousness of the files.

Many improvements over the original attack by Wang et al. have been published. An improvement by Jun Yajima and Takeshi Shimoyama [13] found collision in many hours on a PC. Vlastimil Klima in [14] presented an alteration of the attack in which he was able to find collision in eight hours which was many times faster than the original attack, with complexity of 2^{33} MD5 compression function calls.

Later Kalima presented another attack [15] through which collisions could be found in a minute on notebook PC. This new technique was called Tunneling. Tunneling made collision finding faster and showed a new way of hash function cryptanalysis. It is also used in identical-prefix and chosen-prefix collision later on.

Stevens found new techniques in [16] and made the attack significantly faster by introducing particular control over bit rotations. As it was a two block collision, for the first block a novel algorithm was used based on Kalima's approach and used Kalima's algorithm for second block.

In [17] Stevens at el. presented an improvement of the attack with complexity of 2^{16} MD5 compression function calls.

2.5 Chosen-prefix Collision Attacks

In 2007 Stevens, Lenstra and De Weger presented the first *chosen-prefix* collision attack [1] on MD5. It has complexity of 2^{50} MD5 compression function calls. They discussed several possibilities of abusing their *chosen-prefix* collision attack, and they also showed that colliding X.509 digital certificates with distinct public keys and varied distinguished names can be constructed using their Attack. The collision causing blocks for these certificates were hidden in public key fields. Moreover colliding executables and documents can be created using *chosen-prefix collisions* if in different document layouts collision causing blocks are appended after the completion of the program or in-between pictures.

In 2009 Stevens at el. presented an improvement of the attack in [18] with complexity of 2^{39}. Using this attack they settled a certification authority. They created a cosmetic CA certificate colliding with a conventional website certificate obtained from a trusted CA. The certification authority that they settled was non-hazardous as the certificate's expiration date was intentionally scheduled in the past.

In 2012 a spy-ware with the name of Flame [19] was exposed, researchers found that this spy-ware spread itself using windows update. For this attackers had to obtain a certificate from Microsoft terminal services licensing server with code signing rights. In [20] Marc Stevens done a preliminary analysis of ,Flame and claimed that to deploy itself this malware used a variant of already known *chosen-prefix* collision attack. As using this *chosen-prefix* attack, attackers created two colliding certificates one the innocuous legitimate one singed by Microsoft and the other crafted by the attackers for causing collision.

In 2013 Fillinger Maximilian Johannes and Marc Stevens tried to reconstruct the details of the Flame attack [21] and they give estimates of the complexity of the different steps (i.e. the complexity of the birthday search part and near collision extension construction) of the attack. They claimed that the expected complexity of the Flame's collision attack was $2^{46.6}$ MD5 compression function calls. Their estimates and findings are discussed later in chapter 4 in analysis of Flame attack's reconstruction attempts.

2.6 Conclusion

Flame’s cryptanalytic attack is a modified version of chosen prefix collision attack on message digest algorithm 5. This chapter presents a brief history of generic collision attacks and identical prefix collision attacks carried out on MD5 hashing algorithm moreover phases of improvements of chosen prefix collision attack are also discussed.

Chapter 3

ATTACKS ON HASH FUNCTION MD5

3.1 Introduction

A collision finding algorithm of a cryptographic hash algorithm gives the same hash values for two distinct messages. However some of such attacks provide more control to attacker over the resulting collisions.

In this chapter we describe two of important collision attacks. In section 3.2 the technical details of identical prefix collision attack are discussed. Section 3.3 elaborates the technicalities of chosen prefix collision attack.

3.2 Identical Prefix Collision Attack

3.2.1 Differential Crytanalysis

The differential cryptanalysis [22] is a set of equations involving the intermediate hash values (IHVs) and other intermediate variables (e.g. Q_s, F_s, T_s and $R_s)$ used in MD5 calculation. It shows how the differences in message blocks (B, B') and IHVs propagate. These differences are usually presented using XOR differentials ΔQ, arithmetic differentials $\delta Q = Q' - Q$ arithmetic modulo 2^{32} and *binary signed digit representation* BSDR.

BSDR is more effective as it is useful for tracking bit-wise differentials $(\Delta Q[k])_{k=0,\ldots,31}$ whereas $\Delta Q[k] = Q'[k] - Q[k]\epsilon[1, 0, -1]$. In addition, BSDR encodes both XOR differentials and arithmetic differentials. Arithmetic difference can be easily obtained from a BSDR: $\delta Q = \sum_k \Delta Q[k].2^k \bmod 2^{32}$. A *non-adjacent form* (NAF) is a useful form and has minimal weight among the BSDRs of a word where no two non-zero bits are adjacent. Due to working with modulo 2^{32} NAF of a word is not unique but uniqueness is enforced by using $Q[31]\epsilon[0, +1]$. *NAF weight* is defined as $w(\delta Q)$ i.e. all the non-zero bits in δQ.

Attacks on MD5 based on differential crypanalysis are carried out as follows. Let M and M' be two messages of length $512.n$ and M_i and M'_i be the ith blocks of these two messages where $IHV_0 = IHV'_0$. For $i = 0..., k$ the IHV differential is as follows

$$IHV_1 = MD5(IHV_0, M0), ..., IHV_k = MD5(IHV_{k-1}, Mk - 1)$$

$$IHV'_1 = MD5(IHV'_0, M'0), ..., IHV'_k = MD5(IHV'_{k-1}, M'k - 1)$$

$$\delta IHV_k = (\delta a, \delta b, \delta c, \delta d) = IHV'_k - IHV_k = (a' - a, b' - b, c' - c, d' - d)$$

$512.n$ bits message blocks M_k and M'_k are constructed to obtain a specific δIHV_{k+1} and calculating such intermediate value is part of collision attack. The message blocks are found using differential paths. Wherever for working states and other intermediate hash values, a group of differential equations during the computation of MD5 compression function $MD5Compress$ is defined as a differential path. Message blocks satisfies a differential path if the differences in the computation of $MD5Compress$ with these message blocks as inputs e.g $MD5Compress(IHV_i, Mi), MD5Compress(IHV'_i, M'i)$ are same as the differentials in the differential path.

In identical-prefix attack by Wang et al. [8] initially $M = M'$ because of identical-prefix constraint and $\delta IHV_k = 0$. The first differential path was designed by Wang in such a way that it gives $\delta IHV_{k+1} = (2^{31}, 2^{31} + 2^{25}, 2^{31} + 2^{25}, 2^{31} + 2^{25})$ from $\delta IHV_k = 0$. And the second differential path was designed such that it gives again $\delta IHV_{k+2} = 0)$ from $IHV_{k+1} = (2^{31}, 2^{31} + 2^{25}, 2^{31} + 2^{25}, 2^{31} + 2^{25})$. This identical-prefix collision attack uses their first differential path to find out message blocks M_k and M'_k such that IHV_{k+1} and IHV'_{k+1} cause the differential δIHV_{k+1}.

A difference is introduced between two message blocks by appending M_k and M'_k to

M and M'. As this attack is based on two message blocks so in order to complete the attack a second differential path is created and used to find message blocks M_{k+1} and M'_{k+1} such that δIHV_{k+2} is again zero, when these message blocks are added to M and M' we get the same resultant hash value, that is an md5 collision has been achieved.

The differential path solving algorithm works as follows: given both intermediate hash values i.e. IHV_k and IHV'_k and a differential path, the values of working states $Q_1, Q_2, Q_3..., Q_{16}$ and $Q'_1, Q'_2, Q'_3, ..., Q'_{16}$ are selected corresponding to the differential path. And message blocks M_k and M'_k are selected using these states, then $MD5Compress(IHV_k, M_k)$ and $MD5Compress(IHV'_k, M'k)$ is calculated if these inputs satisfy the differential path a message block is found if not then different values for the pair of working states are tried.

3.2.2 Attack Overview

The identical-prefix collision attack was a two message block collision using differential cryptanalysis presented by Wang et al.[7]. The attack takes M and M' two messages of size $512.n$ and $M = M'$, thus after the computation of MD5 they result in identical IHV. 512 bit message blocks B_0 and B_1 are appended to M and different blocks B'_0 and B'_1 are appended to M' by identical-prefix collision algorithm such that $MD5(M \parallel B_0 \parallel B_1) = MD5(M' \parallel B'_0 \parallel B'_1)$.

This attack constructs two hand crafted differential paths for two succeeding message blocks and uses both modular and XOR differentials for these differential path construction. The first differential path starts from $IHV_k = IHV'_k$ such that $\delta IHV = IHV'_k - IHV_k = (0, 0, 0, 0)$ and ends with

$$\delta IHV_{k+1} = (2^{31}, 2^{31} + 2^{25}, 2^{31} + 2^{25}, 2^{31} + 2^{25})$$

The first differential path is established with following message block differentials in δB_0

$$\delta m_4 = 2^{31}, \delta m_{11} = 2^{15}, \delta m_{14} = 2^{31} \text{ and } \delta m_i = 0 \text{ for all other } m_i s$$

The second differential path starts with above mentioned δIHV_{k+1} and is produced with message block differentials having opposite signs

$$\delta m_4 = 2^{31}, \delta m_{11} = -2^{15}, \delta m_{14} = 2^{31} \text{ and } \delta m_i = 0 \text{ for all other } m_i s$$

So second differential path cancels out the differences produced in first differential path thus making $\delta IHV_{k+2} = (0, 0, 0, 0)$.

Further these differential paths are solved with the help of an algorithm to find the input that satisfy them and produce blocks B_0 and B'_0, B_1 and B'_1 that cause the collision. For speeding up the algorithm two techniques are used. The first technique is message modification in which having a differential path solution upto a particular step a number of solutions are generated. The other one is Tunnels which is more efficient and if a solution does not satisfy a condition on differential path it causes early abort and tries a different solution.

3.2.3 Differential Paths

A differential equations' group established with working states and associated midpoint variables is called a differential path. It shows how differences propagate in two associated MD5 calculations. If two inputs are given to MD5 compression function i.e. $(IHV, m0\|, ... \|m15)$ and $(IHV', m'0\|, ... \|m'15)$ the terminologies for working states and midpoint variables for the input number one will be Q_s, F_s, W_s, T_s and for input number two they are Q_s, F'_s, W'_s, T'_s. The arithmetic differential for these intermediate variables is computed as $\delta Q_s = Q'_s - Q_s$ and the BSDR differential is represented as $\Delta Q_s[i] = Q'_s[i] - Q_s[i]$.

Mainly a differential path for step $s = s_0, ..., s_n$ consists of following information:-

- ΔQ_s for $s_0 - 2, ..., s_n, \delta Q_{s0-3}, ..., \delta Q_{sn+1}$

- ΔF_s for $s_0, ..., s_n$

- $\delta m_0, ..., \delta m_{15}$ from these $\delta W_0, ..., \delta W_{63}$ are derived for s=0,...,63.

- $\delta T_s, \delta R_s$ for $s_0 - 2, ..., s_n,$

From step $s = s'_0, ..., s'_n$ a differential path is said to be solved by a set of inputs to the compression function of MD5Compress (IHV, C) and (IHV', C'). If the difference $C' - C$ = $\delta m_0, ..., \delta m_{15}$, and the working states and midpoint variables that results from the computation of $MD5Compress(IHV, C)$ and $MD5Compress(IHV', C')$ for steps $s = s'_0, ..., s'_n$ solves the differentials given by path, and in addition the working state differentials.

For steps $s = s_0, ..., s_n$ if a pair of input solves a partial differential path, then it is considered as valid. While if s_0 is zero and $s_n = 63$ then it is considered a full valid differential path.

3.2.4 Bit-conditions

Sufficient conditions also called bit-conditions were introduced by Wang et al. He defined them on given differential path bits to find the messages that solves the path. An entire differential path can be described using bit-conditions. There are two types of bit-conditions, *differential bit-conditions*, *boolean function bit-conditions*, the former is used to find out values of ΔQ_s and the latter is used to determine ΔF_s. For steps $s = 0, 1, 2, ..., 63$ differential bit-conditions are presented in Table 3.1 and boolean function bit-conditions are presented in Table 3.2 below.

Symbol	**Condition**	$\Delta Q_s[k]$
.	$Q_s[k] = Q'_s[k]$	0
+	$Q_s[k] = 0 \wedge Q'_s[k] = 1$	+1
-	$Q_s[k] = 1 \wedge Q'_s[k] = 0$	-1

Table 3.1: Differential Bit-conditions.

Symbol	**Condition**	**Type**	**Direction**
.	$Q_s[k] = Q'_s[k]$	direct	
0	$Q_s[k] = Q'_s[k] = 0$	direct	
1	$Q_s[k] = Q'_s[k] = 1$	direct	
ˆ	$Q_s[k] = Q'_s[k] = Q_s[k-1]$	indirect	backward
$\vee$	$Q_s[k] = Q'_s[k] = Q_s[k+1]$	indirect	forward
!	$Q_s[k] = Q'_s[k] = \overline{Q_s[k-1]}$	indirect	backward
y	$Q_s[k] = Q'_s[k] = \overline{Q_s[k+1]}$	indirect	forward
m	$Q_s[k] = Q'_s[k] = Q_s[k-2]$	indirect	backward
w	$Q_s[k] = Q'_s[k] = Q_s[k+2]$	indirect	forward
$\#$	$Q_s[k] = Q'_s[k] = \overline{Q_s[k-2]}$	indirect	backward
h	$Q_s[k] = Q'_s[k] = \overline{Q_s[k+2]}$	indirect	forward
?	$Q_s[k] = Q'_s[k] \wedge (Q_s[k] = 1 \vee Q_s[k-2] = 0)$	indirect	backward
q	$Q_s[k] = Q'_s[k] \wedge (Q_s[k] = 1 \vee Q_s[k+2] = 0)$	indirect	forward

Table 3.2: Boolean Function Bit-conditions.

3.3 Chosen Prefix Collision Attack

3.3.1 Attack Overview

Chosen prefix collision attack work with two arbitrary message blocks A and A' and suffixes F and F' such that MD5($A \parallel F$)= MD5($A' \parallel F'$). Suffixes F and F' are divided into two main parts birthday bits F_b and F'_b and near collision extensions F_c and F'_c.

The first step is birthday bits (F_b and F'_b) search, in which these birthday bits are appended to prefixes $A \parallel F_b$, $A' \parallel F'_b$ and an MD5 calculation is performed on them i.e. $MD5(A \parallel F_b)$ and $MD5(A \parallel F'_b)$. That results in intermediate hash values (IHVs), IHV_n and IHV'_n, causing differentials of specific form $\delta IHV_n = IHV'_n - IHV_n = (\delta a, \delta b, \delta c, \delta d)$, these differentials are then eliminated in next stage with the help of near collision extensions.

The next stage is the search for a sequence of near collision extensions (F_c and F'_c). When these extensions are appended with the birthday bits, they reduce the NAF weight of the IHV differences $\delta IHV_n = (\delta a, \delta b, \delta c, \delta d)$ step by step, until the difference between them becomes zero. A differential path is constructed per near collision extension such that NAF weight of each difference δIHV_{n+1} is less than the NAF weight of the previous difference δIHV_n till after a certain number k of near collision extensions this NAF weight reduces to zero $\delta IHV_{n+k} = (0, 0, 0, 0)$. Near collision extensions are constructed as follows:-

- First two partial differential paths (an upper path and a lower path) are constructed and then they are connected to construct a full differential path. An upper differential path starts with a particular intermediate hash value, whereas lower differential path cause a specific form of intermediate value differentials. Again this differential path construction is done in three different steps using three different algorithms. In first step upper path is extended forward, in second step lower path is extended backward and in third step both these partial differential paths are connected thus making a full differential path.

- Next step is finding message blocks that solves the differential paths and through which target IHV value is acquired.

3.3.2 Elimination Process

For this phase the aim is to find birthday bits F_b and F_b' of length 64+k such that when appended to prefix A and A' and calculating the MD5, i.e. $MD5Compress(A \parallel F_b)$ and MD5Compress $(A' \parallel F_b')$ the resulting intermediate hash values (IHV) difference $\delta IHV_n = IHV_n' - IHV_n = (\delta a, \delta b, \delta c, \delta d)$ is of a particular structure. In which δa is zero and there is some δb and $\delta c = \delta d.$ These differences are gradually reduced to zero using a number of near collision extensions. Near collision extensions are constructed based on the differential paths' end segments. The message blocks differences for these differential paths are $\delta m_{11} = 2^r$ where $0 \leqslant r \leqslant 31$ and $\delta m_i = 0$ for i $\neq 11$. Table below shows the differences between different intermediate differential variables.

s	δQ_s	δF_s	δW_s	δT_s	RC_s	δR_s
61	0	0	$\pm 2^{k-10 \mod 32}$	$\pm 2^{k-10 \mod 32}$	10	$\pm 2^k$
62	$\pm 2^k$	0	0	0	15	0
63	$\pm 2^k$	$\sum_{\lambda=0}^{w'} s_\lambda.2^{k+\lambda}$	0	δF_i	21	$\sum_{\lambda=0}^{w'} s_\lambda.2^{k+21\lambda \mod 32}$
64	$\pm 2^k + \delta R_{63}$					

Table 3.3: Intermediate Variable Differences.

The difference values are calculated in following manner: With δQ_s, 61 is the last trivial step in the order of trivial steps. δF_{61} is 0 because three of the previous $Q_t's$ are zero. And $\delta T_{61} = \delta W_{61} = \delta m_{11} = 2^{k-10 \mod 32}$. The rotation constant for step 61 (RC_{61}) is 10. In the case of right rotation $\delta R_{61} = \delta Q_{62}$ will be $\pm 2^k$.

If both $Q_{60}[k]$ and $Q'_{60}[k]$ are equally zero then δF_{62} = 0. Subsequently $\delta Q_{63} = \delta Q_{62}$. Then there is a possibility of getting non zero values for $\Delta Q_{63}[k]$... $\Delta Q_{63}[k + w']$ and then it may allow ΔF_{63} = $[k + w'] = s\lambda$. And using rotation $RC_{63} = 21$ the possible value for δQ_{64} is as given in table above.

To be more specific, after birthday search the resultant intermediate hash value difference is $\delta IHV = (0, \delta b, \delta c, \delta c)$. Take a parameter $w < 32$. Let denote $(a_i)_{i=0}^{31}$ as NAF weight of $\delta b - \delta c$ and $(b_i)_{i=0}^{31}$ as NAF weight of $\delta c = \delta d$. The first target for elimination is to reduce δc to zero. After reducing δc to zero the intermediate hash value difference will be written as $\delta IHV = (0, \delta b, 0, 0)$. Next phase is reducing the NAF weight of δb to zero. Let $(c_i)_{i=0}^{31}$ denote the NAF weight of δb. Other values of IHV will remain zero as differences added in first and second block will eliminate each other. The collision search is done when the difference δb becomes zero.

3.3.3 Tunnels

Tunneling [15] is a message modification approach used to accelerate collision attacks significantly. Stevens use tunneling technique to accelerate his chosen prefix collision cryptanalytic attack on MD5. Using a Tunnel certain change in a particular working state bit $Q_s[k]$ and it's related message block bits can be introduced. Without upsetting other working states.

3.3.4 Birthday Bits Search

For efficient and storage saving birthday search a specific framework [23] is adopted. In this framework a function f, having search space S is taken. And this function f is applied on different values belonging to this search space S to find a collision $f(a) = f(b)$ where $a \neq b$. Another assumption about f is that it is pseudo-random, so are the compression functions of the cryptographic hash functions.

Hence pseudo-random walks are computed and only the starting values, ending values and lengths of these walks are saved instead of saving the whole trails. The end points of the walks are called distinguished points. Collision generating walks have the same distinguished points. A collision occurs when two walks intersect each other. Then using their stored information the colliding trails are recomputed. Multiple processors can be used to calculate these pseudo-random walks. After a processor finishes computing a trail, it's starting values, ending values and lengths are saved in a list for future reference. And another random a_i is chosen to calculate another random walk.

In the case of chosen prefix collision attack's birthday bits search, in order to produce a specific intermediate hash value differences the search space is taken as $S = Z_{32} \times Z_{32} \times Z_{32}$. If for P and P' we take C and C' as the ending $512 - 64 - l$ bits where $k = 0, ..., 32$. Then the function f would be defined as

$$f(u, v, w) = (a, (c - d), (b - c)\ mod\ 2^k)$$

where as

$$(a, b, c, d) = \begin{cases} MD5Compress(IHV, C \parallel u \parallel v \parallel w), & \text{if } u \equiv 0 \bmod 2 \\ MD5Compress(IHV', C' \parallel u' \parallel v' \parallel w'), & \text{if } u \equiv 1 \bmod 2 \end{cases} \tag{3.1}$$

Nevertheless, all collisions might not be beneficial. Two series of birthday bits are required here one for appending after P and other for P'. Also a particular difference in δb and δc is required that can be removed with a certain number of near collision blocks.

3.3.5 Differential Path Extension

Differential path construction initiates from message block difference, $\delta m_1, ..., \delta m_{15}$, (from these, word differences $\delta W_0, ..., \delta W_{63}$ are fixed) and IHVs. From intermediate hash values we initialize the working states, Q_{-3},..., Q_0 and Q'_{-3},..., Q'_0. Hence the differences ΔQ_{-3},..., ΔQ_0 and related bit-conditions q_{-3},..., q_0 already exist.

3.3.6 Forward Extension of Differential Paths

Forward extension of a differential path follows following method. Let for a step s the information we have is δQ_s and δQ_{s-3} and some bit-conditions q_{s-2} and q_{s-1}.

- First we have to determine bit-condition on Q_s according to δQ_s. It is better to construct differential paths with a small number of bit-conditions. However randomizing the process is also a requirement, a NAF or any other low weight BSDR of δQ_s is used and according to this BSDR, bit-conditions are selected.

- Next the differential ΔF_s is selected and for $j = 0, 1, 2, 3...31$ it is supposed that $(a, b, c) = (q_s[j], q_s[j-1], q_s[j-2])$. we further assume that among these bit-conditions if $q_s[j-2]$ is indirect, then it involves $Q_s[j-1]$. If $V_{s,abc}$ is one then the differential ΔF_s is completely determined by (a, b, c). Else some random g_j is selected from $V_{s,abc}$ and the bit-conditions (a, b, c) are replaced by other bit-conditions (a', b', c') to resolve the ambiguity.

- Next δT is computed using $\delta Q_{s-3} + \delta F_s + \delta W_s$ and δQ_{s+1} is computed using high probability rotation of δT.

3.3.7 Backward Extension of Differential Paths

Lets assume that the information given for a partial differential differential path is q_s, q_{s-1}, δQ_{s+1} and δQ_{s-2}. In order to extended this path backward for a step s and obtaining modular difference δQ_{s-3} we have to give bit-conditions q_{s-2} and change the [.] bit-conditions in q_s and q_{s-1} with suitable boolean function bit-conditions.

The processing of algorithm with backward extension is similar to forward extension algorithm. A BSDR having low weight usually a non adjacent form (NAF) of δQ_{s-2} is chosen, thus on this BSDR bit-conditions $q_s - 2$ are given. ΔF_s differential is acquired by letting for $j = 0, 1, 2, 3...31$ bit-conditions (a, b, c) = $(q_s[j], q_s[j-1], q_s[j-2])$.

In these bit-conditions it is assumed that only $q - s[j]$ is indirect and involves $\delta Q_{s-1}[j]$. If $V_{s,abc}$ is one then there is no ambiguity but if it's greater than one then (a, b, c) is replaced by BC(s ,$V_{s,abc}$, g_j).

And to rotate $\delta R_s = \delta Q_{s+1} - \delta Q_s$ on $32 - RC_s$ bit positions, a high probability $\delta T \,\epsilon$ dRL $(\delta R_s, 32 - RC_s)$ is chosen. Using this we can calculate $\delta Q_s - 3 = \delta T + \delta F_s + \delta W_s$ and one step of backward extension finishes with this calculation.

3.3.8 Connecting Two Partial Differential Paths

Connection of two upper and lower paths is carried out as follows. For δQ_{-3}, δQ_{s+1} and bit-conditions $q_{-2}, q_{-1}, ..., q_s$ the upper path is computed upto some step s. And for δQ_{s+2}, δQ_{64} and bit-conditions $q_{s+3}, ..., q_{62}, q_{63}$ a lower path is constructed backwards till step $s+5$. For constructing a valid differential path the next target is to find the bit-conditions that are compatible with initial sufficient conditions and which determine δQ_{s+1}, δQ_{s+2}, δF_{s+1}, δF_{s+2}, δF_{s+3}, δF_{s+4}.

As all lower and upper paths are not useful so a lot many differential paths are created using different variations of ΔQ_s and ΔF_s while extending these differential paths forwards and backwards. For connection algorithm certain calculations are needed to be performed. For steps j equals to $s + 1$, $s + 2$, $s + 3$ and $s + 4$ first δR_j = δQ_{s+1} + δQ_j is calculated. After that for $32 - RC_j$ bit positions, a high probability rotation differential δT is computed using $dRL(\delta R_j, 32 - RC_s)$.

3.4 Conclusion

This chapter explains the technical details of two collision attacks i.e. identical prefix collision attack and chosen prefix collision attack. It is important to understand the details of these attacks to better comprehend the collision attack used by Flame malware.

Chapter 4

FLAME'S COLLISION ATTACK AND IT'S COMPLEXITY ANALYSIS

4.1 Introduction

This chapter elaborates the details and analysis about the collision attack used by Flame to deploy itself on the target systems. Flame malware exploited a modified version of original chosen-prefix collision cryptanalytic attack on hashing algorithm MD5 [20] in order to deploy itself on the target systems.

Section 4.2 discusses the technical details of the Flame's collision attack. Sections 4.3 provides an analysis of the reconstruction attempts performed for Flame's collision attack. In section 4.4 the cost and complexity estimates of the attack are elaborated. A theoretical analysis of collision attack used by Flame is presented in section 4.5.

4.2 Flame and Collision Attack

In May 2012, Flame a super advanced malware was found by MAHER an Iranian computer emergency response team (CERT) [24]. This malware used collision attack to exploit Windows security update, although the collision attack was not it's part.

4.2.1 Flame Malware

In a technical report [25] by CrySyS Lab and in a blogpost [26] by Kaspersky Labs, the details about the Flame malware were uncovered. It was discovered by Kaspersky that this extremely complex malware was present on the attacked systems since 2010. They stated that this malware consists of multiple modules, these modules could be downloaded later, after the initial deployment. Flame used all key likelihoods for gathering information containing keyboard data, screen captures, microphone and camera content, data from storage devices, Bluetooth input and network traffic. After full deployment its size is above 20 MB. The reading of reports both by CrySys Lab [25] and Kaspersky Lab [26] is recommended in order to better comprehend the functionality, objective and source of the malware.

This malware reportedly attacked computers with Microsoft Windows. Microsoft used to issue certificates through its terminal services licensing server to users for remote logging with code signing rights. These certificates were issued using the weak MD5.

Flame exploited this use of weak MD5 by Microsoft to launch itself. Being linked to Microsoft core certification authority these certificates could also be used to digitally sign any software/program on the behalf of Microsoft. Flame disguised itself as a windows security update in order to locally deploy itself. A certificate linked to the certificate of the core Microsoft certification authority is usually used to acquire a legitimate digital signature on Microsoft Windows update. This digital signature ensures that they are valid updates originated from Microsoft.

To evade this security barrier and to obtain a digital signature on their fake Microsoft Windows security patch. A chosen prefix collision attack was used by Flame developers, on hash function MD5 to obtain two colliding certificates. For this, they crafted the to-be-signed portions of the two digital certificates, such that both to-be-signed parts had identical MD5 hash value.

The real certificate, obtained using above-mentioned terminal services licensing server from Microsoft came with code signing rights. As both of these certificates (real and forged) had identical hash values so digital signature of the former was acceptable for the latter. Since the real certificate came with code signing rights, therefore Flame authors were able to sign the fake windows security patch using this certificate.

4.2.2 Flame's Differential Paths

While analyzing the Flame malware researchers discovered a rogue certificate. Marc Stevens used his newly found technique called counter-cryptanalysis [20] to verify that this certificate was constructed using a variant of his chosen prefix collision attack. He reverse-engineered through this method and extracted four near collision blocks and tried reconstructing of differential paths for all of these near collision blocks.

It was observed that the extracted differential paths were not like any of the collision attack from literature. However for the near collision blocks, the differences of the message blocks were similar to those used in Wang's [8] differential paths. 1st and 3rd block of Flame used the difference of the first differential path of Wang's attack, while the 2nd and 4th block used the difference of second differential path (negated differences) i.e.

$$\begin{aligned} \delta m_4 = \delta m_{14} = +2^{31} \text{ and } \delta m_{11} = +2^{15} \\ \delta m_4 = \delta m_{14} = +2^{31} \text{ and } \delta m_{11} = -2^{15} \end{aligned} \tag{4.1}$$

Furthermore in differential paths used by Flame the initial eight steps carry larger number of bit-conditions than succeeding differential steps.

4.2.3 Birthday Search Part

For this part it was observed birthday search produced an δIHV with a lot of bit differences and four near collision blocks are used to decrease these differences down to zero. According to Fillinger et al's [27] analysis after birthday search fixed differences are required in δa and δd of resultant intermediate hash value. Birthday search part of Flame's collision attack aims for the following function to collide.

$$f(y) = (a, b'_{10}, b'_{11}, ...b'_{13}, b'_{21}, b'_{22}, ...b'_{26}, c_0, c_1, ...c_7, c_{15}, c_{16}, ...c_{19}, c_{31}, d) \tag{4.2}$$

$$\text{whereas } (a,b,c,d) = \begin{cases} MD5Compress(IHV, C||u) + (-2^5, 0, -2^5, 2^9 - 2^5) \text{ u is 0} \\ \\ MD5Compress(IHV', C'||u) \text{ u is 1} \end{cases} \tag{4.3}$$

whereas *b'=b-c*

4.2.4 Tunnels

Tunnels are important to get the accurate complexity estimates of the attack. It was observed that the tunnels T_4, T_5 and T_8 were important for this attack. The use of tunnel T_8 is confirmed however how this tunnel used, is unclear. The observed strengths for tunnels T_4 and T_5 are less than average still it can not be concluded that they are not used at all. It is therefore assumed that average tunnel strength was used for all of the three tunnels.

4.2.5 Near Collision Blocks and Elimination Scheme

As mentioned before only four near collision blocks are used in attack. This could be justified by observing the space where only four differential paths could accommodate. According to observation prior to the search of first near collision extension the IHV differentials are

$$\begin{aligned} \delta IHV &= (\delta a, \delta b, \delta c, \delta d) \\ &= (-2^5, +2^{30} - 2^{21} - 2^{19} - 2^{17} + 2^{12} - 2^2, -2^{27} - 2^{20} + 2^{14} + 2^{12} - 2^5, +2^9 - 2^5) \end{aligned} \tag{4.4}$$

The first near collision block causes following IHV differential.

$$\delta IHV = (\delta a, \delta b, \delta c, \delta d)$$
$$= (2^{31} - 2^{5}, -2^{30} + 2^{25} - 2^{22} + 2^{20} + 2^{17} + 2^{9} - 2^{2}, +2^{31} - 2^{27} + 2^{25} - 2^{20} + 2^{9} - 2^{5},$$
$$2^{31} + 2^{25} + 2^{9} - 2^{5}) \quad (4.5)$$

the second causes

$$\delta IHV = (\delta a, \delta b, \delta c, \delta d) = (0, -2^{30} - 2^{24} - 2^{20} + 2^{17} - 2^{14} + 2^{5}, +2^{0}) \quad (4.6)$$

The third causes

$$\delta IHV = (\delta a, \delta b, \delta c, \delta d) = (2^{31}, 2^{25} + 2^{14} + 2^{9} + 2^{5} - 2^{3} + 2^{0}, 2^{31} + 2^{25}, 2^{31}) \quad (4.7)$$

The forth near collision blocks makes all the differences zero. i.e. δIHV = $(\delta a, \delta b, \delta c, \delta d) = 0$. After the first two near collision blocks the δa and δd becomes zero. The difference δc is left with only a constant term and has NAF weight one. While δb's NAf weight has increased from six to seven. Thus one can infer that first two blocks targeted δc hence reducing it's NAF weight to one and adding random differences to δb that causes the increase in it's NAF weight. While the second two near collision blocks reduce δb to zero along-with eliminating the constant term from δc.

4.3 Analysis of Findings and Reconstruction Attempts

4.3.1 Assumptions

- Flame's collision attack used four differential paths. This assumption is justified from a previous research in which limited space was observed where only few differential paths could be placed.
- The forward and backward differential paths meet at steps 5,6,7 and 8.

- It is assumed that an average tunnel strength is used for all three tunnels T_4, T_5 and T_8.
- The δa and δd differences in the birthday search part were the target differences i.e. the attackers wanted to obtain these from birthday search part of the attack.

4.3.2 Analysis of Differential Paths

Differential path construction for MD5 can be done using two methods. One method is by Stevens et al. [1] meet in the middle approach and the second by Mendel et al.[28]. According to Stevens [20] first few steps of observed differential paths carry large amount of differences and conditions which indicates that a meet in the middle procedure is used. In a meet in the middle approach two partial differential paths are constructed and later they are connected into one full differential path. Furthermore all non-zero bit differences of $\Delta Q6$ in differential paths shows an unknown method was used in full differential path construction instead of Steven's Method.

Flame probably used steps 5, 6, 7 and 8 for connecting two partial differential paths. After completion of step 5 exhaustive search is used for steps 6, 7 and 8. If this is the approach used for connecting two partial differential paths by Flame, exhaustive search can be done using Stevens's method or using brute force. Stevens tried to reconstruct differential paths using their own open source hashing tool [29] with 2^{29} complexity approx.

Later Max Filliner and Marc Stevens in their reconstruction attempt of chosen prefix collision attack specifically differential path construction, also gave an analysis of differential paths. They analyzed that Random IHV differences can occur in first two differential block pair, which can be eliminated in last pair in Flames attack. Because this attack doesn't use static differential paths. Each block's differential path contains a main carry chain starting either from position 5 or 25 in ΔQ_{62} or ΔQ_{63}. These chains are used by Fillinger for their reconstruction attempt and they parametrized the allowed carries. Hence according to them using other carry chains was not useful and their usage could make attack methodology more complex. He also stated that differential path construction technique is sub-optimal.

Finally it is analyzed by looking at both previous observations that It seems a somewhat combination or an unknown approach is used for constructing Flame's differential paths.

Stevens's open source toolkit hashclah can be used to develop differential paths, with more efficiency and less complexity. As as shown by Stevens [20] himself. Because his technique is more efficient than Flame's differential path construction method.

4.3.3 Real Time Attack Complexity

As actual method or algorithm for construction of chosen prefix collision attack used by Flame is not known therefore reconstruction of the attack can be done based on observed findings. However not with exactly same complexity and parameters as used by Flame authors. The original cost of the attack might be higher than the reconstruction attempts. The reason for probable higher cost of original attack is that the observed differential paths carry parameters have minimum values than the parameters used by Fillinger et al. for complexity lower bound C_{min} : $2^{46.6}$. Hence using Flame's differential paths parameters (w_i; v_i; u_i) leads to more birthday factors thus increasing the birthday search cost. And in this way, the overall complexity would be more than C_{min}. Nonetheless the attack can be reconstructed completely and efficiently if simulated on mutually parallel architecture (GPUs) and utilizing the above mentioned assumptions. Although improved assumptions/techniques for tunneling could be used in order to improve overall speed of the attack.

Table 4.1: Analysis of Flame Attack's Differential Path's Construction.

Marc Stevens	• First few steps of observed differential paths carry large amount of differences and conditions which indicates that a meet in the middle procedure is used. • All non-zero bit differences of ΔQ_6 in differential paths shows an unknown method was used in full differential path construction instead of Steven's Method[1]. • Flame probably used steps 5, 6, 7 and 8 for connecting two partial differential paths. After completion of step 5 exhaustive search is used for steps 6, 7 and 8. • Using above approach exhaustive search can be done using Stevens's method or using brute force. • Stevens tried to reconstruct differential paths using their own open source hashing tool [29] with 2^{29} complexity approx.
Marc Stevens and Max Fillinger	• Random IHV differences can occur in first two block pair, which can be eliminated in last pair in Flames attack. • Because it does not used static differential paths. • Differential path construction technique is sub-optimal.
Our Observations and Suggestions	• It seems that somewhat a combination or an unknown approach is used for constructing Flame's differential paths. • Stevens's open source toolkit hashclah [29] can be used with few modifications in the code to develop all four differential paths, with more efficiency less complexity and less bit-conditions. Because his technique is more efficient than Flames differential path construction method.

Note: Differential path construction for MD5 can be done using two methods. One method is by Stevens et al. [1] meet in the middle approach and the second by Mendel et al.[28].

Table 4.2: Analysis of overall Flame's Chosen-prefix Collision Attack.

Marc Stevens	• Gave good lower bounds of 2^{36}, 2^{44}, $2^{30.8}$ and $2^{30.5}$ for 1st, 2nd, 3rd and 4th block construction receptively. • Gave weak lower bounds of 2^{42} for birthday search cost with possibility of increase in complexity due to less systematic differential path system. • Gave a weak lower bound of $2^{44.3}$ for overall attack. [20]
Max Fillinger and Marc Stevens	• Gave estimated complexity assessments with varying values of parameters w_1, v_1, w_2, v_2, w_3 and w_4, u_4* for near collision block construction [21] • Gave Birthday search complexity of $2^{44.8}$ with maximum values of parameters (for minimizing differential path cost i.e. Flame's approach). • C_{msg} : $2^{55.8}$, C_{flame} : $2^{49.3}$, C_{search} : $2^{48.4}$, C_{min} (overall minimum cost): $2^{46.6}$
Our Observations and Suggestions	• The cost of 4 near collision blocks could be reduced using some more proficient techniques from literature. • Running birthday search on massively parallel architecture (GPUs) and looking for birthday bits side by side (parallel) would be faster and better. Rather than using maximum parameters for birthday search, above average parameters could be tried. • In this way both above ideas could possibly reduce the total attack cost.

*Note: w_i, v_i and u_i are differential path parameters. w_i denotes the carry chain bits length and indicate number of bits on which differences can be controlled in differential paths. u_i is another carry chain used in block 4. v_i denotes bits where random changes are allowed. C_{msg} : is the cost when decrease in the cost of near collision search is tried. C_{flame} : Decrease in cost of near collision search is tried and consistency with observed differential parameters is maintained. C_{search} : likely cost, when decrease in birthday search cost is tried.

4.4 Cost and Complexity Analysis

Cost and complexity estimates given by Stevens [20] include good lower bounds of 2^{36}, 2^{44}, $2^{30.8}$ and $2^{30.5}$ for 1st, 2nd, 3rd and 4th near collision extensions construction respectively.

He gave weak lower bounds of 2^{42} for birthday search cost with possibility of increase in complexity due to less systematic differential path system. And also Gave a weak lower bound of $2^{44.3}$ for overall attack.

While Filliner et al. gave estimated complexity assessments with varying values of parameters w_1; v_1, w_2; v_2, w_3 and w_4; u_4 for near collision block construction. He estimated birthday search complexity of $2^{44.8}$ with maximum values of parameters (for minimizing differential path cost i.e. Flame's approach).

For overall cost he gave C_{msg} : $2^{55.8}$ i.e. the cost when decrease in the cost of near collision search is tried. C_{flame} : $2_{49.3}$ i.e. the cost when decrease in cost of near collision search is tried and consistency with observed differential parameters is maintained. C_{search} : $2^{48.4}$ i.e. likely cost, when decrease in birthday search cost is tried. C_{min} : $2^{46.6}$ is the overall minimum cost.

The second and fourth near collision block has the highest complexity in all four blocks reconstructed by Fillinger. Therefore in order to reduce the cost of message block search, the cost of the carry chain parameters of block 2 and 4 should be reduced [21].

The cost of 4 near collision blocks could be reduced using some more proficient techniques from literature. Running birthday search on massively parallel architecture (GPUs) and looking for birthday bits side by side (parallel) would be faster and better. Rather than using maximum parameters for birthday search, above average parameters could be tried. In this way both above ideas could possibly reduce the total attack cost.

4.5 A Theoretical Analysis of Flame's Collision Attack

According to our analysis of the chosen-prefix attack used by Flame malware, this attack is not very efficient as more advanced techniques could have been used from the literature. The attack developers were not focused on optimizing the attack with advanced techniques. Rather they aimed at a successful attack having time efficiency. E.g. the tunnels strengths in observed differential paths are lower than their maximal attainable strengths.

On the whole the complexity, differential paths construction technique and near collision search are sub-optimal. And could be improved using more sophisticated approaches. More advanced and proficient techniques from literature could be used for achieving lower theoretical complexity of four near collision blocks constructions and the birthday search part. Better parameter choices as well as utilization of the full capabilities of already used techniques could lead to better results (e.g. in Tunneling (technique for speeding up near collision) maximum attainable tunnel strengths could be used).

Although a lower bound of the overall cost of Flame's collision Attack is given by Fillinger et al. i.e. $2^{46.6}$, however the closest complexity could be $2^{49.3}$. As the observed differential paths' parameters have minimum values. This could be achieved using massively parallel architectures (GPUs) easily by putting more load on birthday search part while reducing the complexity of near collision part. Birthday search part would give feasible results if done massively parallel architecture. While Near collision search could also be done on a normal CPU as done by Stevens.

4.6 Conclusion

This chapter first describes the technical details of collision attack used by Flame malware. It also provides an analysis of research done on this attack and analysis of reconstruction attempts. It gives an analysis of cost and complexity estimates. Lastly a theoretical analysis of attack has been presented.

Chapter 5

SIMULATION OF FLAME'S DIFFERENTIAL PATHS

5.1 Introduction

In this chapter the replacement differential paths' construction of of Flame's collision attack is described. This construction is performed by making crucial changes in the open source code by Stevens [29]. The substantial algorithm for construction of Flame Attack's differential paths is unknown therefore the differential path construction algorithm for the chosen prefix collision is utilized. However the attributes such as the message block differentials δm and δIHV eliminations of Flame collision attack as observed in literature are used for these replacement differential paths construction.

In this chapter section 5.2 presents the details, figures and tables demonstrating the construction of replacements differential paths.

5.2 Construction of Replacement Differential Paths

In this section the implementation and reconstruction details of differential Flame attack's differential paths are elaborated.

5.2.1 The Intermediate Hash Values (IHVs) Differences

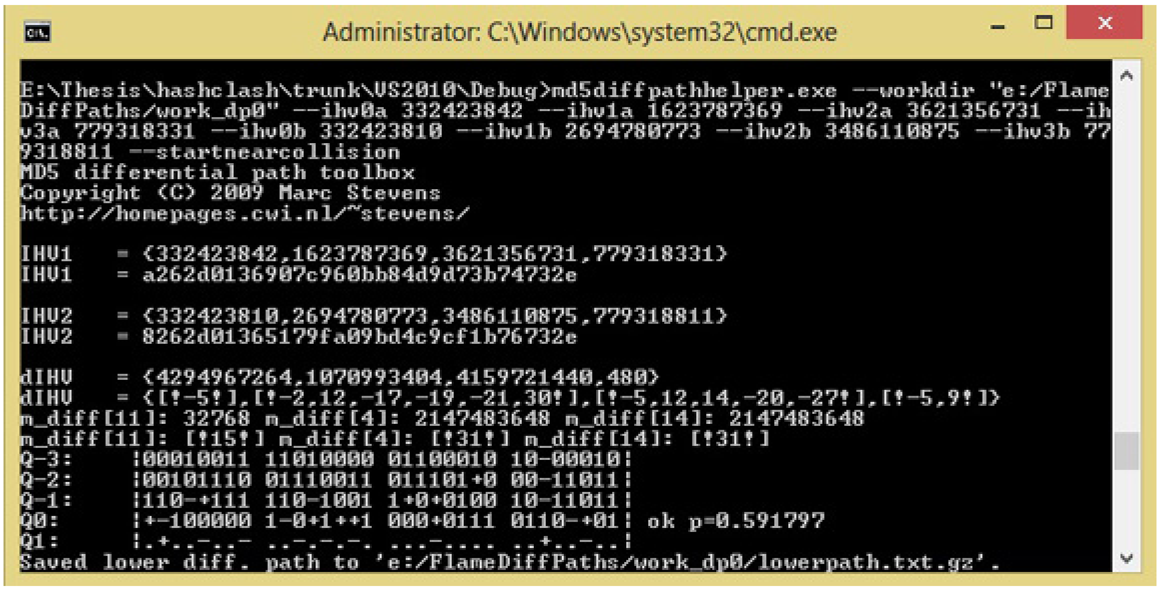
The intermediate hash value differences used for first, second, third and fourth differential paths are shown in table 5.1.

5.2.2 C++ Compiler

The code of HashClash toolkit by Marc Stevens [29] is in C++. As this code and mainly its libhashutil5 c++ library is used in the construction of the Flame's replacement

IHVs For The First Differential Path					
IHV1	Hex	a262d013	6907c960	bb84d9d7	3b74732e
	Dec	332423842	1623787369	3621356731	779318331
IHV2	Hex	8262d013	65179fa0	9bd4c9cf	1b76732e
	Dec	332423810	2694780773	3486110875	779318811
IHVs For The Second Differential Path					
IHV1	Hex	63fc3d45	3bdacbc8	826faa39	cc7df2cc
	Dec	1161690211	3368802875	967470978	3438443980
IHV2	Hex	43fc3dc5	395c9d8a	62719ab3	ac7ff24e
	Dec	3309173827	2325568569	3013243234	1324515244
IHVs For The Third Differential Path					
IHV1	Hex	7aeea241	ddd49e30	b9ce4dab	4b8e0ff4
	Dec	1101196922	815715549	2874003129	4094660171
IHV2	Hex	8262d013	65179fa0	9bd4c9cf	1b76732e
	Dec	332423810	2694780773	3486110875	779318811
IHVs For The Fourth Differential Path					
IHV1	Hex	ac3aa31b	d79e7f3a	9b34ec0a	850e3940
	Dec	463682220	981442263	183252123	1077481093
IHV2	Hex	ac3aa39b	ee607f3c	9bf6eb8c	851039c2
	Dec	162740908	1014980846	2364274331	3258519685

Table 5.1: Intermedite Hash Value Differences (IHVs) for Differential Paths



```
Administrator: C:\Windows\system32\cmd.exe

E:\Thesis\hashclash\trunk\VS2010\Debug>md5diffpathhelper.exe --workdir "e:/Flame
DiffPaths/work_dp0" --ihv0a 332423842 --ihv1a 1623787369 --ihv2a 3621356731 --ih
v3a 779318331 --ihv0b 332423810 --ihv1b 2694780773 --ihv2b 3486110875 --ihv3b 77
9318811 --startnearcollision
MD5 differential path toolbox
Copyright (C) 2009 Marc Stevens
http://homepages.cwi.nl/~stevens/

IHV1    = {332423842,1623787369,3621356731,779318331}
IHV1    = a262d0136907c960bb84d9d73b74732e

IHV2    = {332423810,2694780773,3486110875,779318811}
IHV2    = 8262d01365179fa09bd4c9cf1b76732e

dIHV    = {4294967264,1070993404,4159721440,480}
dIHV    = {[!-5!],[!-2,12,-17,-19,-21,30!],[!-5,12,14,-20,-27!],[!-5,9!]}
m_diff[11]: 32768 m_diff[4]: 2147483648 m_diff[14]: 2147483648
m_diff[11]: [!15!] m_diff[4]: [!31!] m_diff[14]: [!31!]
Q-3:     |00010011 11010000 01100010 10-00010|
Q-2:     |00101110 01110011 011101+0 00-11011|
Q-1:     |110-+111 110-1001 1+0+0100 10-11011|
Q0:      |+-100000 1-0+1++1 000+0111 0110-+01| ok p=0.591797
Q1:      |.+..-..- ..-.-.-. ...-.... ..+..-..|
Saved lower diff. path to 'e:/FlameDiffPaths/work_dp0/lowerpath.txt.gz'.
```

Figure 5.1: Starting Point for Forward Differential Extension of First Differential Path.

differential paths. Hence Simulation is done using Microsoft 2010 Professional.

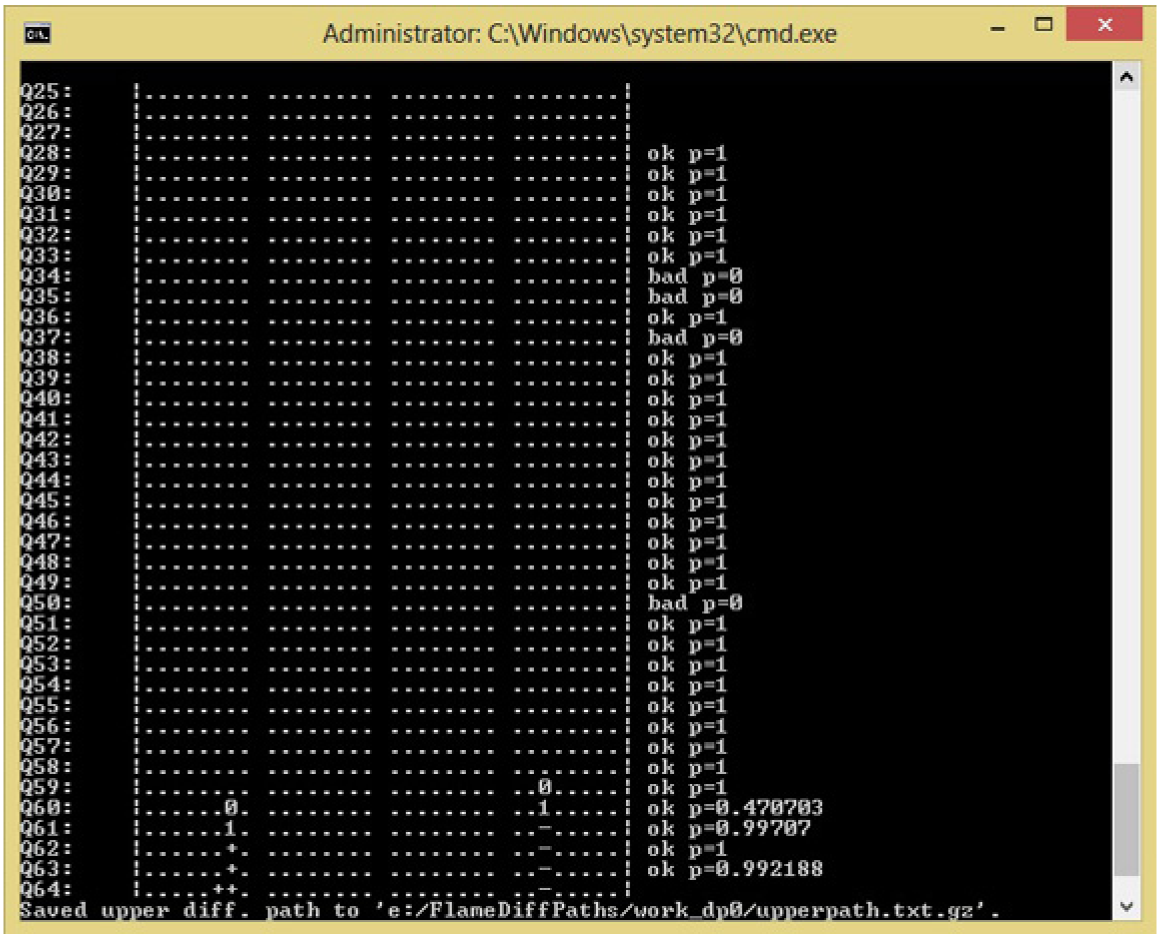
5.2.3 Construction of Starting and Ending Segment of Replacement Differential Paths

Construction of lower differential path starts with creating the starting segments of the forward differential step. Which is achieved by first determining the bit-conditions q_{-3}, q_{-2}, q_{-1} and q_0 against the corresponding IHV and IHV', and extending it one step further. Thus constructing a starting segments of partial lower path. And the construction of upper path starts with constructing the desired end segments of a partial upper differential path keeping it compatible with given message differences δm.

Figure 5.1 shows the construction of starting segments of first differential path. Here first bit-conditions against the IHVs are determined and differential step is calculated for $t = 1$. The message block differentials used for first differential path are $\delta m_4 = 15$, $\delta m_{11} = \delta m_{14} = 31$ as can be seen in figure.

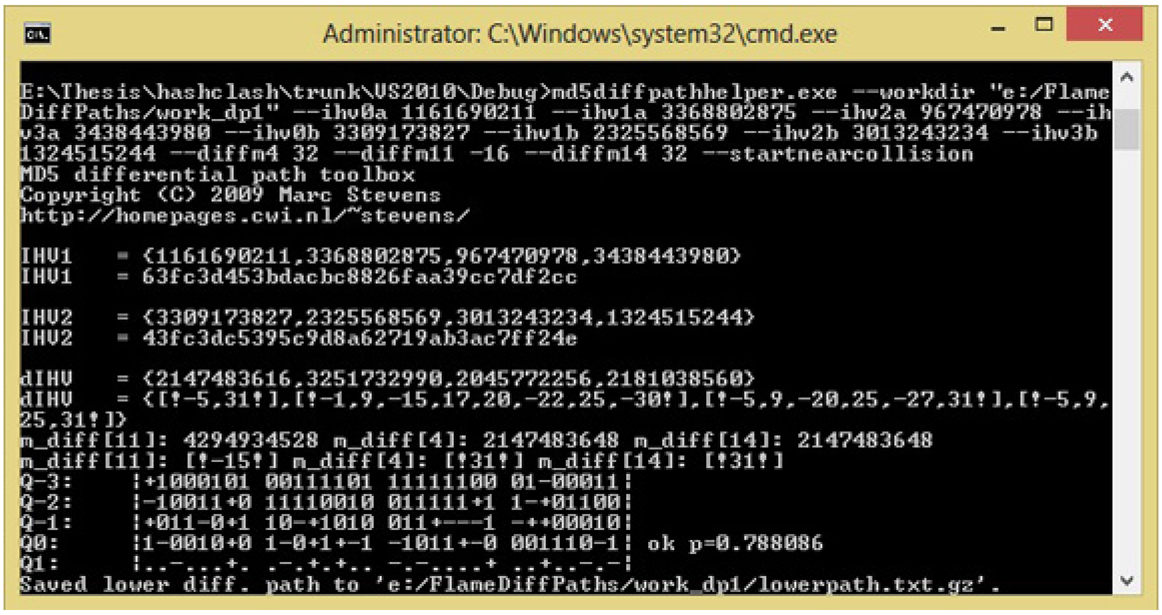
A specified end of the first differential path is shown in figure 5.2.

Similarly Starting Segments of the Second Replacement Differential Path is shown in figure


Administrator: C:\Windows\system32\cmd.exe

```
Q25:     |........ ........ ........ ........|
Q26:     |........ ........ ........ ........|
Q27:     |........ ........ ........ ........|
Q28:     |........ ........ ........ ........| ok p=1
Q29:     |........ ........ ........ ........| ok p=1
Q30:     |........ ........ ........ ........| ok p=1
Q31:     |........ ........ ........ ........| ok p=1
Q32:     |........ ........ ........ ........| ok p=1
Q33:     |........ ........ ........ ........| ok p=1
Q34:     |........ ........ ........ ........| bad p=0
Q35:     |........ ........ ........ ........| bad p=0
Q36:     |........ ........ ........ ........| ok p=1
Q37:     |........ ........ ........ ........| bad p=0
Q38:     |........ ........ ........ ........| ok p=1
Q39:     |........ ........ ........ ........| ok p=1
Q40:     |........ ........ ........ ........| ok p=1
Q41:     |........ ........ ........ ........| ok p=1
Q42:     |........ ........ ........ ........| ok p=1
Q43:     |........ ........ ........ ........| ok p=1
Q44:     |........ ........ ........ ........| ok p=1
Q45:     |........ ........ ........ ........| ok p=1
Q46:     |........ ........ ........ ........| ok p=1
Q47:     |........ ........ ........ ........| ok p=1
Q48:     |........ ........ ........ ........| ok p=1
Q49:     |........ ........ ........ ........| ok p=1
Q50:     |........ ........ ........ ........| bad p=0
Q51:     |........ ........ ........ ........| ok p=1
Q52:     |........ ........ ........ ........| ok p=1
Q53:     |........ ........ ........ ........| ok p=1
Q54:     |........ ........ ........ ........| ok p=1
Q55:     |........ ........ ........ ........| ok p=1
Q56:     |........ ........ ........ ........| ok p=1
Q57:     |........ ........ ........ ........| ok p=1
Q58:     |........ ........ ........ ........| ok p=1
Q59:     |........ ........ ........ ..0.....| ok p=1
Q60:     |......0. ........ ........ ..1.....| ok p=0.470703
Q61:     |......1. ........ ........ ..-.....| ok p=0.99707
Q62:     |......+. ........ ........ ..-.....| ok p=1
Q63:     |......+. ........ ........ ..-.....| ok p=0.992188
Q64:     |.....++. ........ ........ ..-.....|
Saved upper diff. path to 'e:/FlameDiffPaths/work_dp0/upperpath.txt.gz'.
```

Figure 5.2: End Point for Backward Differential Extension of First Differential Path.



```
E:\Thesis\hashclash\trunk\VS2010\Debug>md5diffpathhelper.exe --workdir "e:/Flame
DiffPaths/work_dp1" --ihv0a 1161690211 --ihv1a 3368802875 --ihv2a 967470978 --ih
v3a 3438443980 --ihv0b 3309173827 --ihv1b 2325568569 --ihv2b 3013243234 --ihv3b
1324515244 --diffm4 32 --diffm11 -16 --diffm14 32 --startnearcollision
MD5 differential path toolbox
Copyright (C) 2009 Marc Stevens
http://homepages.cwi.nl/~stevens/

IHV1    = {1161690211,3368802875,967470978,3438443980}
IHV1    = 63fc3d453bdacbc8826faa39cc7df2cc

IHV2    = {3309173827,2325568569,3013243234,1324515244}
IHV2    = 43fc3dc5395c9d8a62719ab3ac7ff24e

dIHV    = {2147483616,3251732990,2045772256,2181038560}
dIHV    = {[!-5,31!],[!-1,9,-15,17,20,-22,25,-30!],[!-5,9,-20,25,-27,31!],[!-5,9,
25,31!]}
m_diff[11]: 4294934528 m_diff[4]: 2147483648 m_diff[14]: 2147483648
m_diff[11]: [!-15!] m_diff[4]: [!31!] m_diff[14]: [!31!]
Q-3:     |+1000101 00111101 11111100 01-00011|
Q-2:     |-10011+0 11110010 011111+1 1-+01100|
Q-1:     |+011-0+1 10-+1010 011+---1 -++00010|
Q0:      |1-0010+0 1-0+1+-1 -1011+-0 001110-1| ok p=0.788086
Q1:      |..-...+. .-.+.+.. -.-....+ ..+..-.-|
Saved lower diff. path to 'e:/FlameDiffPaths/work_dp1/lowerpath.txt.gz'.
```

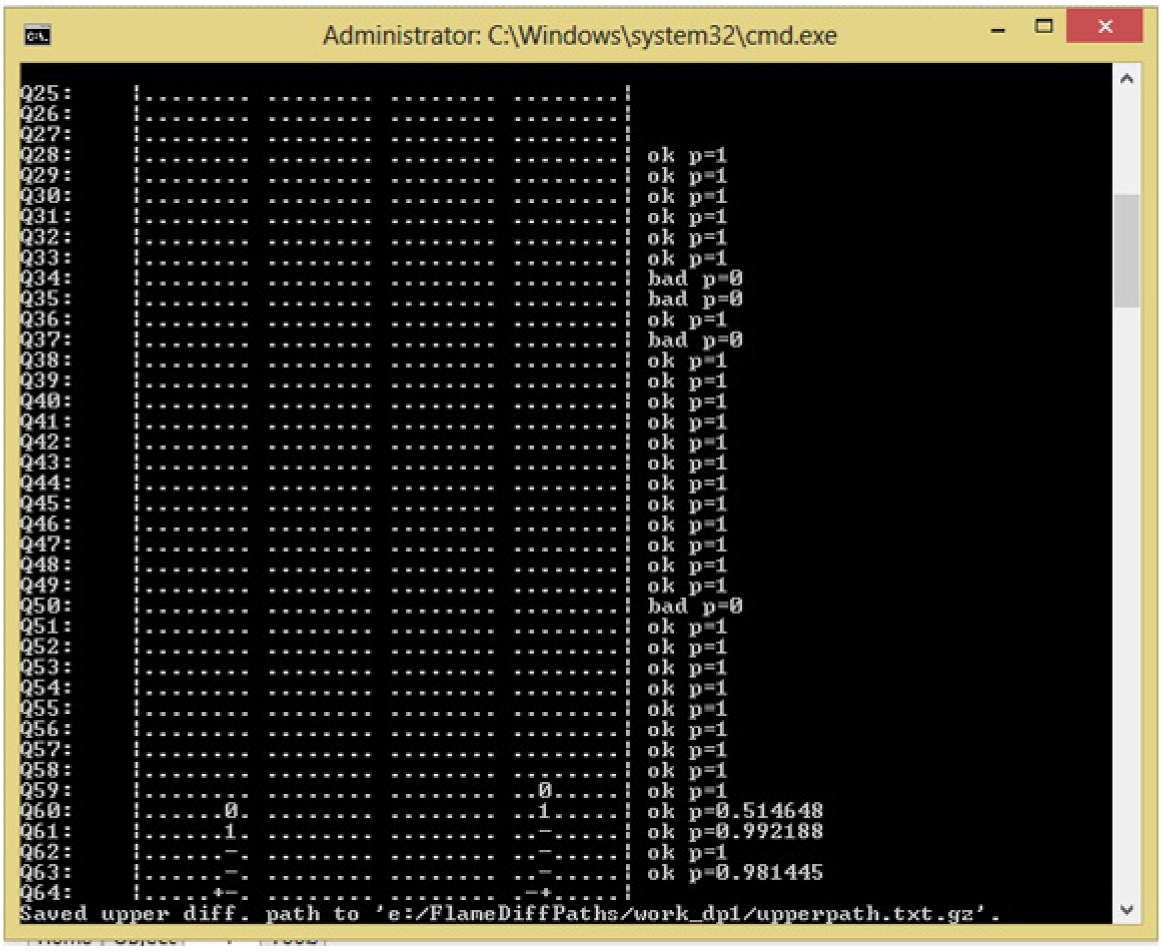
Figure 5.3: Starting Points for Forward Differential Extension Second Differential Path.

5.3. The message differences used for this differential path are $\delta m_4 = -15$, $\delta m_{11} = \delta m_{14} = 31$

Figure 5.4 displays the end points of second replacement differential path. In addition the third and fourth differential paths' starting and ending points are displayed by figures 5.5, 5.6, 5.7 and 5.8 respectively. The message block differentials used in the construction of third differential path are same as used for constructing the first differential path. Whereas differentials used for the construction of fourth differential path are same as used in second differential path construction.

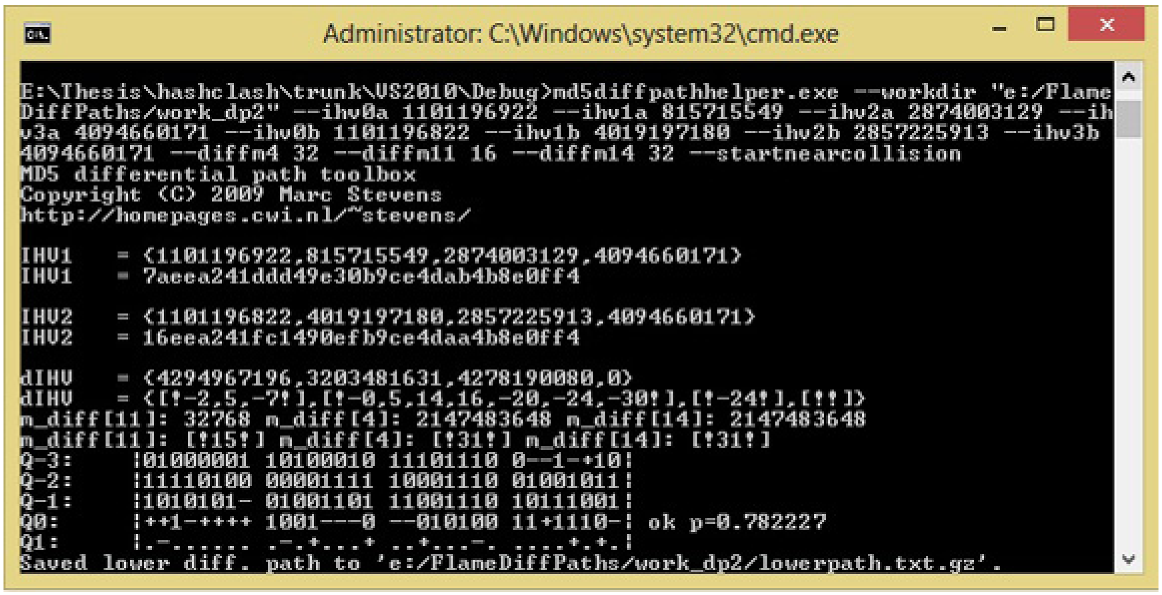
5.2.4 Forward Differential Path Extension

For constructing partial lower replacement differential path, first message block differences δm_0, δm_1, δm_2,..., δm_{14}, δm_{15} are determined. In the case of Flame for the first differential path construction message block differences $m_4 = m_{14} = 31$ and $m_{11} = 15$ are used. And the IHV differentials used are given in table 5.1. The main function for forward differential path construction takes name of working directory *workdir*, name of the input



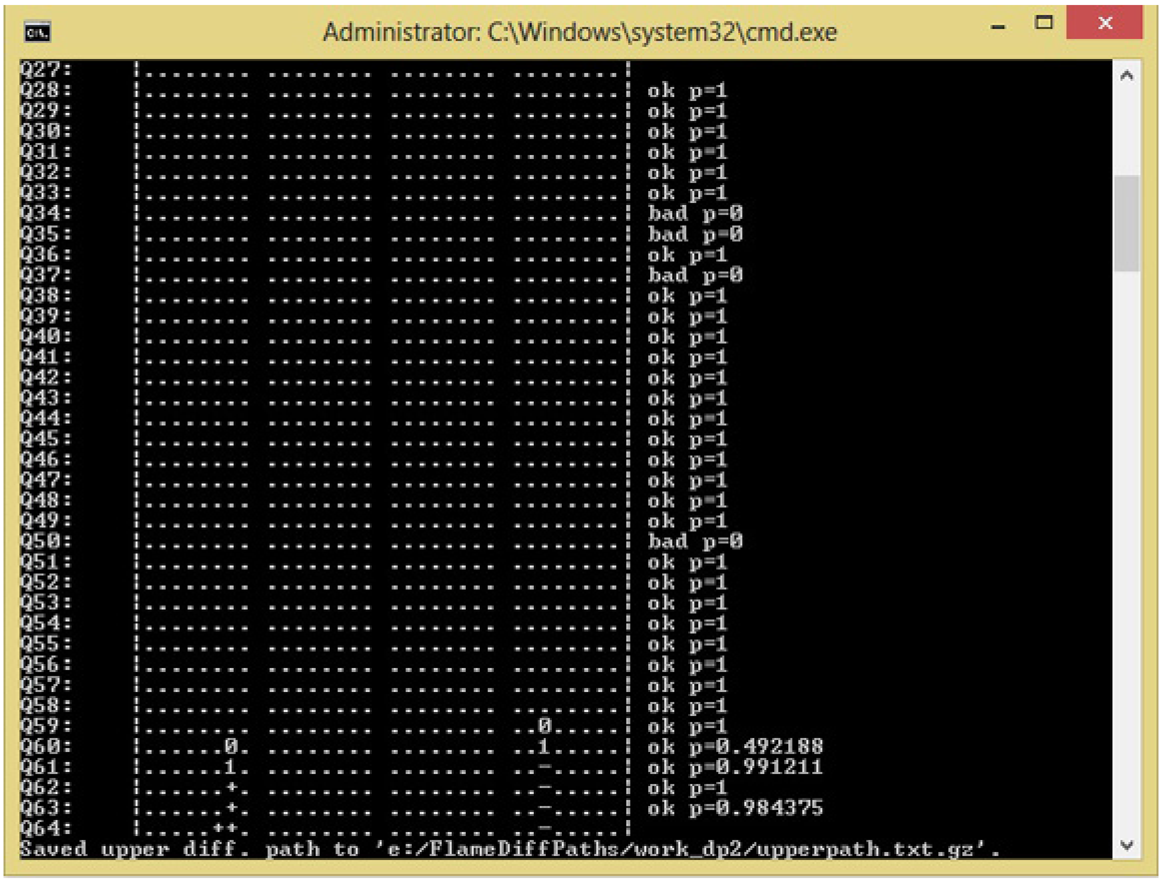
```
Q25:     |........ ........ ........ ........|
Q26:     |........ ........ ........ ........|
Q27:     |........ ........ ........ ........|
Q28:     |........ ........ ........ ........| ok p=1
Q29:     |........ ........ ........ ........| ok p=1
Q30:     |........ ........ ........ ........| ok p=1
Q31:     |........ ........ ........ ........| ok p=1
Q32:     |........ ........ ........ ........| ok p=1
Q33:     |........ ........ ........ ........| ok p=1
Q34:     |........ ........ ........ ........| bad p=0
Q35:     |........ ........ ........ ........| bad p=0
Q36:     |........ ........ ........ ........| ok p=1
Q37:     |........ ........ ........ ........| bad p=0
Q38:     |........ ........ ........ ........| ok p=1
Q39:     |........ ........ ........ ........| ok p=1
Q40:     |........ ........ ........ ........| ok p=1
Q41:     |........ ........ ........ ........| ok p=1
Q42:     |........ ........ ........ ........| ok p=1
Q43:     |........ ........ ........ ........| ok p=1
Q44:     |........ ........ ........ ........| ok p=1
Q45:     |........ ........ ........ ........| ok p=1
Q46:     |........ ........ ........ ........| ok p=1
Q47:     |........ ........ ........ ........| ok p=1
Q48:     |........ ........ ........ ........| ok p=1
Q49:     |........ ........ ........ ........| ok p=1
Q50:     |........ ........ ........ ........| bad p=0
Q51:     |........ ........ ........ ........| ok p=1
Q52:     |........ ........ ........ ........| ok p=1
Q53:     |........ ........ ........ ........| ok p=1
Q54:     |........ ........ ........ ........| ok p=1
Q55:     |........ ........ ........ ........| ok p=1
Q56:     |........ ........ ........ ........| ok p=1
Q57:     |........ ........ ........ ........| ok p=1
Q58:     |........ ........ ........ ........| ok p=1
Q59:     |........ ........ ........ ..0.....| ok p=1
Q60:     |......0. ........ ........ ..1.....| ok p=0.514648
Q61:     |......1. ........ ........ ..-.....| ok p=0.992188
Q62:     |......-. ........ ........ ..-.....| ok p=1
Q63:     |......-. ........ ........ ..-.....| ok p=0.981445
Q64:     |.....+-. ........ ........ ..-+....|
Saved upper diff. path to 'e:/FlameDiffPaths/work_dp1/upperpath.txt.gz'.
```

Figure 5.4: End Points for Backward Differential Extension of Second Differential Path.



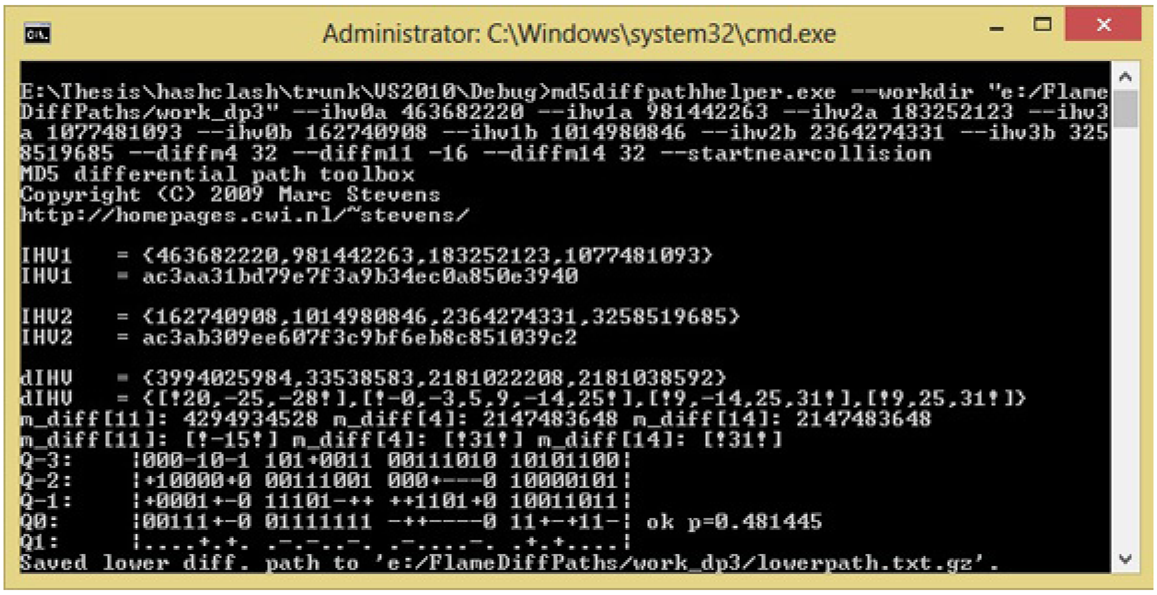
```
E:\Thesis\hashclash\trunk\VS2010\Debug>md5diffpathhelper.exe --workdir "e:/Flame
DiffPaths/work_dp2" --ihv0a 1101196922 --ihv1a 815715549 --ihv2a 2874003129 --ih
v3a 4094660171 --ihv0b 1101196822 --ihv1b 4019197180 --ihv2b 2857225913 --ihv3b
4094660171 --diffm4 32 --diffm11 16 --diffm14 32 --startnearcollision
MD5 differential path toolbox
Copyright (C) 2009 Marc Stevens
http://homepages.cwi.nl/~stevens/

IHV1    = {1101196922,815715549,2874003129,4094660171}
IHV1    = 7aeea241ddd49e30b9ce4dab4b8e0ff4

IHV2    = {1101196822,4019197180,2857225913,4094660171}
IHV2    = 16eea241fc1490efb9ce4daa4b8e0ff4

dIHV    = {4294967196,3203481631,4278190080,0}
dIHV    = {[!-2,5,-7!],[!-0,5,14,16,-20,-24,-30!],[!-24!],[!!]}
m_diff[11]: 32768 m_diff[4]: 2147483648 m_diff[14]: 2147483648
m_diff[11]: [!15!] m_diff[4]: [!31!] m_diff[14]: [!31!]
Q-3:     |01000001 10100010 11101110 0--1-+10|
Q-2:     |11110100 00001111 10001110 01001011|
Q-1:     |1010101- 01001101 11001110 10111001|
Q0:      |++1-++++ 1001---0 --010100 11+1110-| ok p=0.782227
Q1:      |.-...... .-.+...+ ..+...-. ....+.+.|
Saved lower diff. path to 'e:/FlameDiffPaths/work_dp2/lowerpath.txt.gz'.
```

Figure 5.5: Starting Point for Forward Differential Extension Third Differential Path.



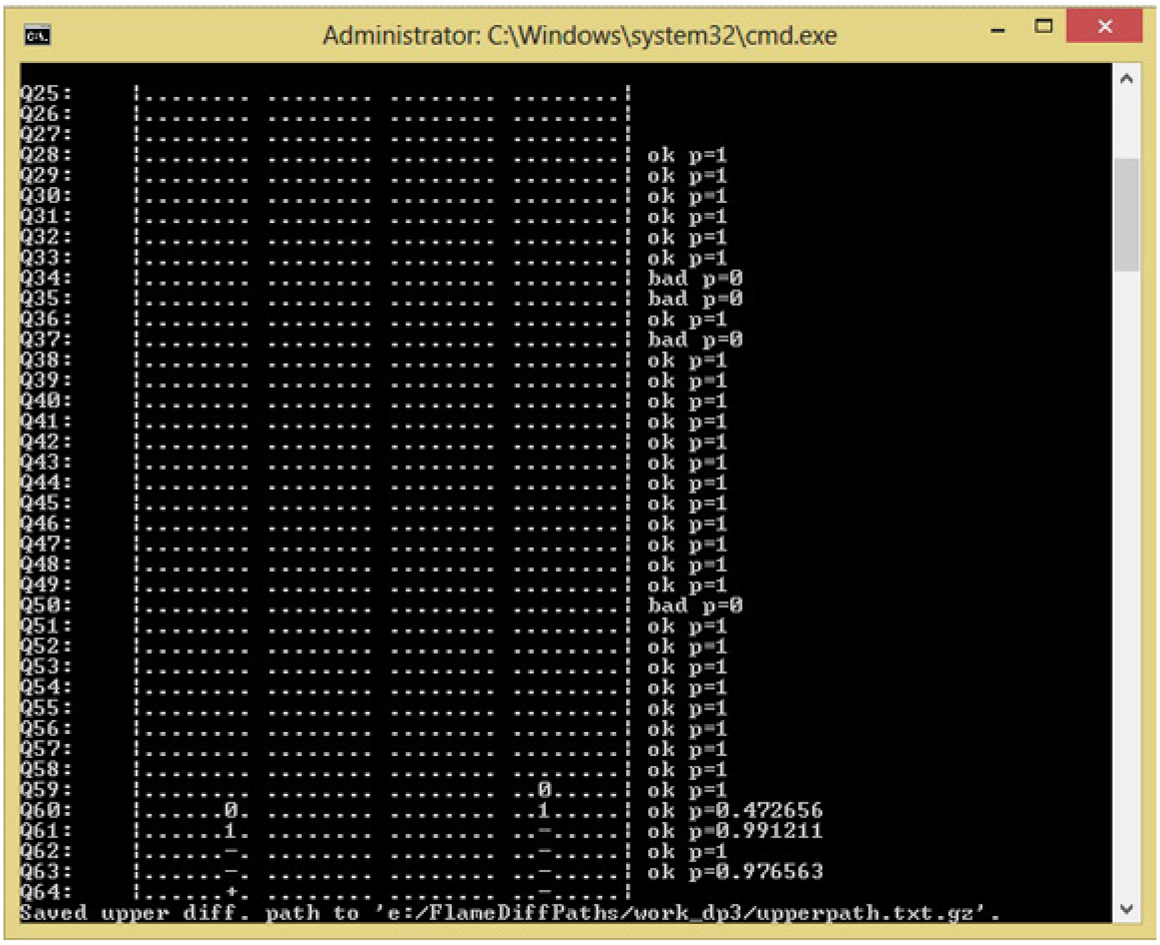
```
Q27:      |........ ........ ........ ........|
Q28:      |........ ........ ........ ........| ok p=1
Q29:      |........ ........ ........ ........| ok p=1
Q30:      |........ ........ ........ ........| ok p=1
Q31:      |........ ........ ........ ........| ok p=1
Q32:      |........ ........ ........ ........| ok p=1
Q33:      |........ ........ ........ ........| ok p=1
Q34:      |........ ........ ........ ........| bad p=0
Q35:      |........ ........ ........ ........| bad p=0
Q36:      |........ ........ ........ ........| ok p=1
Q37:      |........ ........ ........ ........| bad p=0
Q38:      |........ ........ ........ ........| ok p=1
Q39:      |........ ........ ........ ........| ok p=1
Q40:      |........ ........ ........ ........| ok p=1
Q41:      |........ ........ ........ ........| ok p=1
Q42:      |........ ........ ........ ........| ok p=1
Q43:      |........ ........ ........ ........| ok p=1
Q44:      |........ ........ ........ ........| ok p=1
Q45:      |........ ........ ........ ........| ok p=1
Q46:      |........ ........ ........ ........| ok p=1
Q47:      |........ ........ ........ ........| ok p=1
Q48:      |........ ........ ........ ........| ok p=1
Q49:      |........ ........ ........ ........| ok p=1
Q50:      |........ ........ ........ ........| bad p=0
Q51:      |........ ........ ........ ........| ok p=1
Q52:      |........ ........ ........ ........| ok p=1
Q53:      |........ ........ ........ ........| ok p=1
Q54:      |........ ........ ........ ........| ok p=1
Q55:      |........ ........ ........ ........| ok p=1
Q56:      |........ ........ ........ ........| ok p=1
Q57:      |........ ........ ........ ........| ok p=1
Q58:      |........ ........ ........ ........| ok p=1
Q59:      |........ ........ ........ ..0.....| ok p=1
Q60:      |......0. ........ ........ ..1.....| ok p=0.492188
Q61:      |......1. ........ ........ ..-.....| ok p=0.991211
Q62:      |......+. ........ ........ ..-.....| ok p=1
Q63:      |......+. ........ ........ ..-.....| ok p=0.984375
Q64:      |.....++. ........ ........ ..-.....|
Saved upper diff. path to 'e:/FlameDiffPaths/work_dp2/upperpath.txt.gz'.
```

Figure 5.6: End Point for Backward Differential Extension of Third Differential Path.



```
E:\Thesis\hashclash\trunk\VS2010\Debug>md5diffpathhelper.exe --workdir "e:/Flame
DiffPaths/work_dp3" --ihv0a 463682220 --ihv1a 981442263 --ihv2a 183252123 --ihv3
a 1077481093 --ihv0b 162740908 --ihv1b 1014980846 --ihv2b 2364274331 --ihv3b 325
8519685 --diffm4 32 --diffm11 -16 --diffm14 32 --startnearcollision
MD5 differential path toolbox
Copyright (C) 2009 Marc Stevens
http://homepages.cwi.nl/~stevens/

IHV1    = {463682220,981442263,183252123,1077481093}
IHV1    = ac3aa31bd79e7f3a9b34ec0a850e3940

IHV2    = {162740908,1014980846,2364274331,3258519685}
IHV2    = ac3ab309ee607f3c9bf6eb8c851039c2

dIHV    = {3994025984,33538583,2181022208,2181038592}
dIHV    = {[!20,-25,-28!],[!-0,-3,5,9,-14,25!],[!9,-14,25,31!],[!9,25,31!]}
m_diff[11]: 4294934528 m_diff[4]: 2147483648 m_diff[14]: 2147483648
m_diff[11]: [!-15!] m_diff[4]: [!31!] m_diff[14]: [!31!]
Q-3:      |000-10-1 101+0011 00111010 10101100|
Q-2:      |+10000+0 00111001 000+---0 10000101|
Q-1:      |+0001+-0 11101-++ ++1101+0 10011011|
Q0:       |00111+-0 01111111 -++----0 11+-+11-| ok p=0.481445
Q1:       |....+.+. .-.-..-. .-....-. .+.+....|
Saved lower diff. path to 'e:/FlameDiffPaths/work_dp3/lowerpath.txt.gz'.
```

Figure 5.7: Starting Point for Forward Differential Extension Third Differential Path.



```
Q25:     |........ ........ ........ ........|
Q26:     |........ ........ ........ ........|
Q27:     |........ ........ ........ ........|
Q28:     |........ ........ ........ ........| ok p=1
Q29:     |........ ........ ........ ........| ok p=1
Q30:     |........ ........ ........ ........| ok p=1
Q31:     |........ ........ ........ ........| ok p=1
Q32:     |........ ........ ........ ........| ok p=1
Q33:     |........ ........ ........ ........| ok p=1
Q34:     |........ ........ ........ ........| bad p=0
Q35:     |........ ........ ........ ........| bad p=0
Q36:     |........ ........ ........ ........| ok p=1
Q37:     |........ ........ ........ ........| bad p=0
Q38:     |........ ........ ........ ........| ok p=1
Q39:     |........ ........ ........ ........| ok p=1
Q40:     |........ ........ ........ ........| ok p=1
Q41:     |........ ........ ........ ........| ok p=1
Q42:     |........ ........ ........ ........| ok p=1
Q43:     |........ ........ ........ ........| ok p=1
Q44:     |........ ........ ........ ........| ok p=1
Q45:     |........ ........ ........ ........| ok p=1
Q46:     |........ ........ ........ ........| ok p=1
Q47:     |........ ........ ........ ........| ok p=1
Q48:     |........ ........ ........ ........| ok p=1
Q49:     |........ ........ ........ ........| ok p=1
Q50:     |........ ........ ........ ........| bad p=0
Q51:     |........ ........ ........ ........| ok p=1
Q52:     |........ ........ ........ ........| ok p=1
Q53:     |........ ........ ........ ........| ok p=1
Q54:     |........ ........ ........ ........| ok p=1
Q55:     |........ ........ ........ ........| ok p=1
Q56:     |........ ........ ........ ........| ok p=1
Q57:     |........ ........ ........ ........| ok p=1
Q58:     |........ ........ ........ ........| ok p=1
Q59:     |........ ........ ........ ..0.....| ok p=1
Q60:     |......0. ........ ........ ..1.....| ok p=0.472656
Q61:     |......1. ........ ........ ..-.....| ok p=0.991211
Q62:     |......-. ........ ........ ..-.....| ok p=1
Q63:     |......-. ........ ........ ..-.....| ok p=0.976563
Q64:     |......+. ........ ........ ..-.....|
Saved upper diff. path to 'e:/FlameDiffPaths/work_dp3/upperpath.txt.gz'.
```

Figure 5.8: End Point for Backward Differential Extension of Third Differential Path.

file *inputfile* (i.e. *lowerpath.txt.gz* which is generated as a result of running the previous code of constructing the start and end segments of differential paths), step *t/tstep* from where to start constructing the forward differential path, upto *trange* of additional steps input, using *maxconditions* number of bit-conditions, starting step *condtbegin* from where *maxconditions* will apply, number *autobalance* of forward differential paths to be created.

When required parameters are given for all four replacement differential path the results are four partial forward replacement differential path shown in table 5.1, 5.2, 5.3 and 5.4 respectively. 1000000 lower differential paths are generated as not all paths are useful so it is always beneficial to create this many paths.

Step t	Bit-conditions	Probabilities
-3	00010011 11010000 01100010 10-00010	
-2	00101110 01110011 011101+0 00-11011	
-1	110-+111 110-1001 1+0+0100 10-11011	
0	+-100000 1-0+1++1 000+0111 0110-+01	0.487305
1	1+.1-..- .-.+.++. 11.-.1.. 0V+.1-..	0.698242
2	-1.++..0 .0.-.+.. -..-.+.. +0+.++..	0.189453
3	–.0-..1 .-.-.+0. 0....+.. 0+..-0..	0.749023
4	.+.10... .-...+.. 1..1.-.. –1.-1..	0.744141

Table 5.2: Replacement Differential Path 1

Step t	Bit-conditions	Probabilities
-3	+1000101 00111101 11111100 01-00011	
-2	-10011+0 11110010 011111+1 1-+01100	
-1	+011-0+1 10-+1010 011+—1 -++00010	
0	1-0010+0 1-0+1+-1 -1011+-0 001110-1	1
1	11-.01+. .-1+.+10 -.-111.+ 11+.-.++	1
2	.01..-0. .....+1- ..-.-11- 1.-.-.00	0.636719
3	.-1..00. .1.0...- 1...+..- -.-.0.0+	0.492188
4	.+...1-. .....1.. ..1.1... -...-..-	0.548828
5	..+.+..- .+.-.+.. .....-.+ ..+.+...	

Table 5.3: Replacement Differential Path 2

Step t	Bit-conditions	Probabilities
-3	01000001 10100010 11101110 0–1-+10	
-2	11110100 00001111 10001110 01001011	
-1	1010101- 01001101 11001110 10111001	
0	++1-++++ 1001—0 –010100 11+1110-	1
1	++.11111 .-.+101+ 01+...-. ..0.+.+0	0.548828
2	...+1-10 .0.-001+ 1+-...+. V.1.0.-0	1
3	01.+.-.. .1.-...- -+-...+. ....+.0.	0.724609
4	...+.+.. .+.....- -.-...-. +...-.1.	0.694336
5	...+.+.+ ..-...-. +.-..+.. -.+...-.	

Table 5.4: Replacement Differential Path 3

Step t	Bit-conditions	Probabilities
-3	000-10-1 101+0011 00111010 10101100	
-2	+10000+0 00111001 000+—0 10000101	
-1	+0001+-0 11101-++ ++1101+0 10011011	
0	00111+-0 01111111 -++—-0 11+-+11-	0.441406
1	0V..+.+. .-.-.1-1 1-1110-. +.–1..0	0.914063
2	..V.+1-. .+.+..-. 1+0111-. +.+-0..-	0.376953
3	.-..+.-. .+.0..-. –....+. 1.0-...-	0.904297
4	.0-..... .-.1.... 0-....-. -.-....-	0.473633
5	..-.-.-. -.+....- ...-.-.. +.+.+..-	

Table 5.5: Replacement Differential Path 4

5.3 Conclusion

This chapter describes the details of the construction of forward replacement differential paths. In the initial phase starting segments for forward replacement differential paths are constructed based on observed characteristics of Flame's differential paths along with possible ending segments. Then four forward replacement differential paths are presented in the form of bit-conditions.

Chapter 6

ADVANCE MALWARE EVASION TECHNIQUES AND COUNTERMEASURES

6.1 Introduction

Like Stuxnet, Duqu and Gauss, Flame is also a member of the weaponized group of malwares. They are also called Advance Persistent Threats (APTs) because of their highly sophisticated, covert and targeted behavior. Each of these malwares use a new technique or exploit a vulnerability to deploy themselves.

In this chapter few possible ways that could be used by advanced malwares to launch themselves are discussed. Counter-measures are also deliberated that could be used both for detection prior to infection and for prevention from damage after infection by malwares like Flame.

In section 6.2 possible malware evasion techniques are explored that are used by malwares to deploy themselves. Section 6.3 provides countermeasure techniques for malware detection before infection. While countermeasures for detecting malwares after deployment are elaborated in section 6.4.

6.2 Possible Malware Evasion Techniques

Malwares usually exploit previously unknown or known yet overlooked vulnerabilities to infect their targets. Attackers employ a system of specially programmed bots that automatically scrutinize the internet and report about susceptible/weak systems, servers, certification authorities (CAs) and websites.

6.2.1 Exploiting Weak Cryptographic Primitives

Weak cryptographic primitives that are already proven to be broken but still in use are a great threat to the security of the systems they protect. Whereas due to various reasons such as compatibility issues, cost/risk concerns, sometimes implausible weaknesses/abuse scenarios and also remissness, weak primitives are continued to be used even after their expiry dates and warnings against their usage.

Constructing digital signature schemes is a main application of hash function cryptographic primitives. Weak digital signature systems suffer from the problem of accepting old digital signatures even if these signatures are based on weak primitives. As the signer of the digital certificates has no control over them because of their number so replacement of all the old and weak signatures with new secure ones is impractical. As a result, it is evident that the signature verifiers will keep on accepting the old/weak and perhaps rogue digital signatures.

The cryptographic attack behind Flame is an example of exploiting weak digital signatures, since it is already mentioned before how Flame authors fooled Microsoft and obtained a certificate with code signing rights using MD5 based digital signature scheme. Although Microsoft was doing serious effort since 2008 to shift to a secure hash algorithm for new signatures but they keep on accepting old/weak MD5 based digital signatures. Furthermore, they totally overlooked the use of digital signatures based on MD5 for licensing users by their terminal services licensing server. However despite the security update by Microsoft, there might be old certificates that are not expired yet e.g. they will still accept binary files that were signed using insecure MD5 based certificates before 2009.

Nevertheless, in spite of warnings and real world attacks any entity that still signs certificates using digital signature schemes based on MD5 could be attacked using chosen prefix collision attack used by Flame malware. Also anyone can be targeted using the consequential digital signature abuse, because MD5 based signatures are supported by everyone nearly ubiquitously. [20]

6.2.2 Exploiting Cryptographic Backdoors

Cryptographic or encryption backdoors are inserted in cryptographic algorithms usually by government intelligence agencies [30][31]. Reportedly using these backdoor an intruder may get authorization for accessing encrypted information in the absence of valid credentials. An intruder can either deduce the access key using the message context or will present a master/golden key that always allows the access. Even if the attacker does not succeed in the total break of the cipher, some vulnerability leading to any cryptanalytic attack might be introduced using the backdoor.

Knowledge of any such backdoor becomes an open invitation for an attacker to launch a sophisticated attack.

6.2.3 Exploiting Software Vulnerabilities

A software vulnerability is defined as a weakness in security and/or design of a software or an operating system that makes them prone to serious security issues. Apparently a harmless looking glitch can become a serious security concern when attackers discover it and exploit it to launch their malwares or execute their malicious plots. Sometimes even implementation pitfalls, i.e. wrong implementation of cryptographic primitives makes them vulnerable.

6.2.3.1 zero-day vulnerabilities

In this category of attacks, attackers aim at exploiting unpatched software glitches that are either unknown yet or overlooked by software vendors. Zero-day vulnerabilities remained a favorite target for the authors of many advance malwares.

Stuxnet an advanced malware discovered in 2010 [32] also used four zero-day vulnerabilities in Microsoft Windows to spread itself. This was the first time a malware exploited so many unknown and unfixed glitches.

Likewise, DuQu a successor of Stuxnet used a DOC file that exploited a zero-day vulnerability in Microsoft Windows kernel when opened. [33]

6.3 Countermeasures for Detection Before Infection

6.3.1 Collision Detection using Counter-Cryptanalysis

In [17] Marc Stevens presented the concept of counter-cryptanalysis. This new criterion could be used for supporting weak cryptographic primitives against cryptanalytic attempts. Suspicious attempts can be detected through this method even before malwares could be deployed. This paradigm facilitates detection of cryptanalytic attacks by exploiting slight unavoidable anomalies. These anomalies are introduced by active cryptanalytic attacks that give specially crafted inputs to weak cryptographic primitives.

Strengthening the design of weak cryptographic primitive by replacing it, in order to make it resistant against cryptanalytic attempts, introduce the backward compatibility problems. While counter-cryptanalysis maintains backward compatibility and is applied on cryptographic primitive. Thus changing maneuver of this cryptographic primitive, with detecting and blocking cryptanalytic attacks. The techniques introduced in this paradigm can solve the problems of signature verifiers, as they can now determine whether a signature is a part of cryptanalytic collision attack or not. And thus, it supports continued safe use of vulnerable cryptographic primitive with complete backward compatibility. Suspicious attempts can be detected through this method even before malwares installation.

6.3.2 Weak Cryptographic Primitive Detection

Another approach for countering cryptographic attacks could be weak cryptographic primitive detection. This paradigm could have the capability of detecting and indicating that an entity (e.g. a website, a certification authority (CA)) is using weak a cryptographic primitive. And proceeding to the use of such entity could lead to serious security risks. Consequently, a user who is unaware of the technicalities of digital certificates and digital signatures could prevent him/herself from malwares, who target weak primitives for deploying themselves.

6.4 Countermeasures After Malware Deployment

6.4.1 File-based Sandboxing

File-based sand boxing assesses the maliciousness of an unknown file by analyzing its execution behavior. This technique provides a better chance of detection of unknown malwares by basing the decision on file's action (behavior) rather than file's content (signature).

There are two main types of file analysis [34], dynamic (behavioral) analysis and static (code) analysis. File-based sandboxes facilitate dynamic analysis along with few static analysis abilities and decides about the legitimacy or maliciousness of a file sample. File-based sandboxes are further classified into two types, virtualization-based sandboxes and emulation-based sandboxes.

Malware authors use various techniques to obstruct these analysis techniques and bypass detection. These evasion techniques may include human interaction evasion, environment specific evasion and configuration specific evasion. Among configuration specific evasion techniques, sleep calls are most important. As sandboxes analyzes files for a specified period of time, hence malware authors try to delay the execution of the malware (send sleep calls) long enough that the sandbox gets timeout. Also to execute the malware in a specific period of time, time triggers are used by malware authors.

A better way to detect evasive behavior with a high level of confidence is to analyze the execution of the malware both in sandbox and on client side. Because malware is designed to run on target machine rather than sandbox so difference in execution pattern will clearly indicate the maliciousness of the file.[35]

6.4.2 Malware Detection Using Memory Forensics

Memory forensics is very useful and novel concept for malware detection. Because everything running in the operating system whether they are processes, malwares (rootkits), threads, open files, network sockets, IP addresses, encryption keys, registry keys, events logs and hardware/software configuration resides in memory.

This method is advantageous if the attackers have succeeded in infecting the target system. The memory data of the target system is collected and an analysis is performed for significant elements and finally evidence is recovered about the malware. Physical memory data (from RAM) and page file data are taken in an ideal analysis[36].

In [37] a new method of memory forensics is introduced. In this approach, a comparison of data of various structures of memory management is used simultaneously for malware detection. This data is specifically picked up from kernel and user space of memory management. Malware information concerning changes in registry, library files calls and operating system function are gathered and evaluated. Among this information samples are ranked based upon certain chosen characteristics. This technique's best results included 98% detection rate and 16% false positives rate.

6.5 Conclusion

In this chapter various techniques that advance malwares authors can use to inject/deploy their malwares on target systems are highlighted. Furthermore, countermeasures for detecting such malwares even before getting successful in evasion and detection after their evasion are also discussed.

Chapter 7

CONCLUSION AND FUTURE DIRECTION

7.1 Introduction

This chapter concludes the thesis. Some thoughts on possible improvements of the work have been presented in section 7.3.

7.2 Conclusion

The Flame's Collision Attack is yet ambiguous with repsect to it's algorithms and actual techniques. As it was a variant of Chosen prefix Collision Attack of literature.

Four partial replacement differential paths bearing the attributes of Flame's differential paths have been constructed using Stevens's open source code by making few modifications in the code.

Four specified starting and ending segments of partial backward replacement differential paths are also constructed. Each of these segments are constructed using the message block differentials and intermediate hash value differences (IHVs) observed in Flame's differential paths found in literature.

A complexity analysis of Flame's chosen prefix collision attack is performed and suggestions are given for possible improvement of the attack. In order to thwart malwares like Flame which exploited a weak cryptographic primitive (MD5 Hashing Algorithm) a new preventive measure Weak Cryptographic Primitive Detection is proposed along with other counter-measures.

7.3 Future Directions

The proposed paradigm for detection of cryptographic could be realized and implemented in future in order to prevent the abuse of cryptographic primitives. Partial upper replacement differential paths may also be constructed. The bit-conditions in replacement differential paths may be reduced thus making them more feasible than Flame's actual differential paths. Proper tunnels with maximum achievable strengths may be used in partial forward differential paths, hence making more efficient paths.

Appendix A

USER MANUAL

1. The Boost C++ libraries are needed to be build in order to run the framework (version 1.52.0 can be downloaded and build)

2. As the source code uses program options from Boost C++ libraries so in order to set options it is better to run, compile and build the programs from command prompt in administrative mode.

3. The command *"msbuild hasclash.sln /p:configuration=debug* has to be used to build the project.

4. After successful build of the differential path construction programs, directory must be changed to *"Debug"* where exe files are placed and then md5diffpathhelper.exe has to be run.

5. The input that must be given to *md5diffpathhelper.exe* includes, the name of the working directory (where constructed initial upper and lower paths file would be stored), and message differences *diffm4*, *diffm11* and *diffm14*.

6. The IHV input should be in decimal form while message differences should be in BSDR form (0-32).

7. After the construction of the initial lower and upper differential paths (starting and ending segments) two file *lowerpath.txt.gz* and *upperpath.txt.gz* will be stored in the given working directory.

8. Next *md5diffpathforward.exe* hash to be run.

9. The input for *md5diffpathforward.exe* file includes the name of working directory (*workdir*), the input file name (here for forward differentiate paths it would be *lowerpath.txt.gz*), *tstep* = (first step to start), *trange* = (range of more steps to be performed), *condtbegin* = (from where to start defining the differentials in terms of bitconditions), *maxweight* = (By default maximum SDRs weight limit is 14) *maxsdrs* = (By default maximum SDRs limit is 160, could be decided how many SDRs should be used, *autobalance* = (Number of differential paths to be generated, maximum limit is 1000000. Generating large number of differential paths is essential but it requires more memory.), *threads* = (Number of worker threads to be assigned to a program. The differential paths construction program are multi-threaded.)

10. The *"Include Directories"* and *"Library Directories"* must include the directories having Boost C++ libraries and headers. They can be linked from Visual Studio 2010 Professional as well as from command prompt.

BIBLIOGRAPHY

[1] M. Stevens, A. Lenstra, and B. De Weger, "Chosen-prefix collisions for md5 and colliding x. 509 certificates for different identities," in *Advances in Cryptology-EUROCRYPT 2007*, pp. 1–22, Springer, 2007.

[2] R. Rivest, "The md5 message-digest algorithm," 1992.

[3] I. B. Damgård, "A design principle for hash functions," in *Advances in Cryptolo gy-CRYPTO89 Proceedings*, pp. 416–427, Springer, 1990.

[4] B. den Boer and A. Bosselaers, "Collisions for the compression function of md5," in *Advances in CryptologyEUROCRYPT93*, pp. 293–304, Springer, 1994.

[5] H. Dobbertin, "Cryptanalysis of md5 compress, 1996," *Rump Session of EuroCrypt*, vol. 96, 1996.

[6] H. Dobbertin, "The status of md5 after a recent attack," *CryptoBytes*, vol. 2, no. 2, 1996.

[7] X. Wang, D. Feng, X. Lai, and H. Yu, "Collisions for hash functions md4, md5, haval-128 and ripemd.," *IACR Cryptology ePrint Archive*, vol. 2004, p. 199, 2004.

[8] X. Wang and H. Yu, "How to break md5 and other hash functions," in *Advances in Cryptology–EUROCRYPT 2005*, pp. 19–35, Springer, 2005.

[9] P. Hawkes, M. Paddon, and G. G. Rose, "Musings on the wang et al. md5 collision.," *IACR Cryptology ePrint Archive*, vol. 2004, p. 264, 2004.

[10] D. Kaminsky *et al.*, "Md5 to be considered harmful someday.," *IACR Cryptology ePrint Archive*, vol. 2004, p. 357, 2004.

[11] A. K. Lenstra, X. Wang, and B. de Weger, “Colliding x. 509 certificates.,” *IACR Cryptology ePrint Archive*, vol. 2005, p. 67, 2005.

[12] M. Daum and S. Lucks, “Attacking hash functions by poisoned messages, the story of alice and her boss, june 2005.”

[13] J. Yajima and T. Shimoyama, “Wang’s sufficient conditions of md5 are not sufficient.,” *IACR Cryptology ePrint Archive*, vol. 2005, p. 263, 2005.

[14] V. Klima, “Finding md5 collisions on a notebook pc using multi-message modifications.,” *IACR Cryptology ePrint Archive*, vol. 2005, p. 102, 2005.

[15] V. Klima, “Tunnels in hash functions: Md5 collisions within a minute.,” *IACR Cryptology ePrint Archive*, vol. 2006, p. 105, 2006.

[16] M. Stevens, A. K. Lenstra, and B. de Weger, “Target collisions for md5 and colliding x. 509 certificates for different identities.,” *IACR Cryptology ePrint Archive*, vol. 2006, p. 360, 2006.

[17] M. M. J. Stevens *et al.*, *Attacks on hash functions and applications*. Mathematical Institute, Faculty of Science, Leiden University, 2012.

[18] M. Stevens, A. Sotirov, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, and B. De Weger, “Short chosen-prefix collisions for md5 and the creation of a rogue ca certificate,” in *Advances in Cryptology-CRYPTO 2009*, pp. 55–69, Springer, 2009.

[19] S. A. TEAM *et al.*, “skywiper: A complex malware for targeted attacks,” 2012.

[20] M. Stevens, “Counter-cryptanalysis,” in *Advances in Cryptology–CRYPTO 2013*, pp. 129–146, Springer, 2013.

[21] M. J. Fillinger, *Reconstructing the Cryptanalytic Attack Behind the Flame Malware*. PhD thesis, Universiteit van Amsterdam, 2013.

[22] E. Biham and A. Shamir, *Differential cryptanalysis of the data encryption standard*. Springer Science & Business Media, 2012.

[23] P. C. Van Oorschot and M. J. Wiener, "Parallel collision search with cryptanalytic applications," *Journal of cryptology*, vol. 12, no. 1, pp. 1–28, 1999.

[24] Maher, "Identification of a new targeted cyber-attack," tech. rep., May 2012.

[25] S. A. TEAM *et al.*, "skywiper: A complex malware for targeted attacks," tech. rep., Technical Report, 2012.

[26] K. A. TEAM *et al.*, "Kaspersky lab. the flame: Questions and answers, may 2012. securelist blog," tech. rep., Technical Report, 2012.

[27] M. Fillinger and M. Stevens, "Reverse-engineering of the cryptanalytic attack used in the flame super-malware," in *Advances in Cryptology–ASIACRYPT 2015*, pp. 586–611, Springer, 2014.

[28] F. Mendel, C. Rechberger, and M. Schläffer, "Md5 is weaker than weak: Attacks on concatenated combiners," in *Advances in Cryptology–ASIACRYPT 2009*, pp. 144–161, Springer, 2009.

[29] "Hashclash webpage:." `https://marc-stevens.nl/p/hashclash/`.

[30] J. MENN, "Exclusive: Secret contract tied nsa and security industry pioneer," tech. rep., Reuters, 2013.

[31] R. Gallagher and G. Greenwald, “Nsa helped british spies find security holes in juniper firewalls,” tech. rep., The Intercept, 2015.

[32] D. Kushner, “The real story of stuxnet,” *IEEE Spectrum*, vol. 50, no. 3, pp. 48–53, 2013.

[33] J. Tang, “Analysis of cve-2015-2360? duqu 2.0 zero day vulnerability,”

[34] L. Zelster, “Introduction to malware analysis,” pp. 01–36.

[35] H. Mourad, “Sleeping your way out of the sandbox,” pp. 01–23.

[36] H. Pomeranzs, “Detecting malware with memory forensics,” pp. 01–27.

[37] M. Aghaeikheirabady, S. M. R. Farshchi, and H. Shirazi, “A new approach to malware detection by comparative analysis of data structures in a memory image,” in *Technology, Communication and Knowledge (ICTCK), 2014 International Congress on*, pp. 1–4, IEEE, 2014.