이학석사 학위논문

생년월일 기반 비밀번호를 가지는 보안명세서에 대한 전수조사공격

2020년 2월

서울대학교 대학원

수리과학부 수리과학 전공

김민

생년월일 기반 비밀번호를 가지는 보안명세서에 대한 전수조사공격

지도교수 홍진

이 논문을 이학석사 학위논문으로 제출함
2019년 10월

서울대학교 대학원
수리과학부 수리과학 전공
김민

김민의 석사 학위논문을 인준함
2019년 12월

위 원 장 현동훈 (인)

부위원장 홍진 (인)

위 원 이승진 (인)

국문초록

통신사 및 카드회사들은 암호화된 이용요금 명세서를 e-mail을 통하여 고객들에게 전송하는 서비스를 제공하고 있다. 여기서 주로 사용하는 첨부파일의 형식은 PDF와 HTML형식이며, 고객의 생년월일 6자리의 숫자로 암호화하여 고객이 아닌 제3자가 열어볼 수 없도록 접근을 제한하고 있다. 본 논문에서는 PDF와 HTML의 파일 형식을 고려한 전수조사 공격(Brute Force Attack)방법을 알아보고, 대량의 명세서 파일이 유출되었을 때, 심각한 개인정보 유출로 인한 피해가 발생할 수 있음을 알아본다.

주요어(Keywords): 전수조사공격(Brute Force Attack), 보안명세서, PDF , HTML
학 번 : 2018-23801

목 차

표 목 차

그 림 목 차

코드 목 차

제 1 장 서론

이동통신 및 신용카드 사용자들은 많은 경우 해당 서비스 제공자로부터 주기적으로 요금명세서를 이메일의 암호화된 첨부파일로 받고 있다. 이러한 보안명세서는 대부분 현실적인 이유로 고객의 생년월일 6자리로 암호화되어 있다.

개인정보보호에 대한 국민의 관심이 높아지면서 대략 10년 전부터 널리 사용되기 시작한 보안명세서는 초기에는 Active-X에 기반한 HTML 첨부파일이었다. 현재는 자바스크립트로 복호화 기능을 제공하는 HTML 첨부파일 형태의 솔루션을 여러 보안업체에서 각기 다른 방식으로 구성하여 제공하고 있으며, 한편으로는 PDF 표준 자체의 보안 기능으로 암호화한 PDF 명세서 파일을 첨부하는 방식도 사용되고 있다.

초기 보안명세서는 명세서 암호화를 위한 비밀번호를 고객이 사전에 등록하는 방식이었으나, 곧 이러한 번거로움을 피할 수 있는 주민등록번호 뒤 7자리의 비밀번호 사용이 널리 퍼지게 되었다. 이후 개인정보보호법 개정으로 주민등록번호의 사용이 어려워지면서 현재와 같이 생년월일 6자리를 비밀번호로 사용하는 형태가 일반화되었다.

생년월일에 대한 전수조사 공격은 그 복잡도가 $2{,}320 \approx 2^{4.5}$에 불과하므로 보안명세서가 특정 개인을 목표로 설정한 APT(Advanced Persistent Threats)공격에 대하여 의미 있는 안전성을 제공할 것으로 기대할 수 없다. 그보다는 서비스 제공자가 고객에게 보안명세서를 발송하는 것은 그 목적이, 보이스피싱을 위한 준비 작업과 같이, 요금명세서를 대량으로 확보하려는 공격자에 대한 방어에 있다고 해석하는 것이 합당할 것이다.

본 논문에서는 이러한 낮은 수준의 안전성마저도 생년월일로 암호화된 보안명세서로는 확보할 수 없음을 살펴본다. 현재 널리 사용되고 있는 몇몇 보안명세서 방식에 대하여 전수조사 공격을 구현 및 수행하여 실제로 소요되는 대략적인 공격시간을 제시하고, 결과적으로 보안명세서가 그 내용에 더하여 생년월일을 추가로 공격자의 손에 넘겨주는 역효과를 가져옴을 보인다. 또한, 생년월일과 함께 노출된 보안명세서가 심각한 개인정보유출을 야기할 수 있음을 지적한다.

제 2 장 이론적 배경

먼저 PDF파일의 비밀번호를 판단하는데 사용한 해시함수와 HTML파일의 분석에서 등장한 HMAC기법, PBKDF2에 대하여 간략히 알아본 후, PDF파일의 형식과 비밀번호판단알고리즘을 살펴본다.

2.1 MD5 해시함수

<u>정의1</u> 해시함수는 임의의 길이의 입력을 특정한 길이의 출력으로 내보내는 함수이다. 암호학적 해시함수(Cryptographic Hash Function)

$: \{0,1\}$ $\rightarrow \{0,1\}^r, r \in$ 는 해시함수의 일종으로 다음의 성질을 만족하는 함수이다[1,3].

① 주어진 임의의 길이의 입력 에 대해서 $h(x)$를 빠르게 계산할 수 있어야 한다.

② 역상저항성(Preimage Resistance): 주어진 $y \in \{0,1\}^r$에 대하여 $y = h(x)$를 만족하는 $x \in \{0,1\}^*$를 찾는 것이 계산적으로 불가능해야 한다.

③ 충돌저항성(Collision Resistance): $h(x_1) = h(x_2) \in \{0,1\}^r$, x_1 x_2 만족하는 $x_1, x_2 \in \{0,1\}^*$를 찾는 것도 계산적으로 불가능해야한다.

MD5(Message-Digest algorithm 5)는 MIT의 Ronald Rivest 교수가 1991년에 개발하였고, RFC 1321(Request For Comments: 1321) 표준에 규정되어 있는 암호학적 해시함수이다. 2004년에 해시충돌이 발견되어, 더 이상 보안용도로 MD5를 사용하는 것은 바람직하지 않지만, 여전히 많은 곳에서 MD5를 사용하고 있다[7].

MD5는 Merkle-Damgard 구조를 이용하여 설계되었고 128비트의 출력을 가지며, 입력 값 은 512비트의 배수가 되도록 패딩된다. 패딩된 입력 값을 512비트 단위로 나누어 메시지블록이라고 하고 M, M_2, ..., M_n (각각 512 비트)로 표기하자. 압축함수의 i-번째 출력 값을 IHV_i라고 하고, 압축함수를 CF로 표기하면, MD5의 알고리즘은 다음과 같다[6].

HV = fixed value

$IHV_1 = CF\,IHV_0, M_1)$

$IHV_2 = CF(IHV_1, M_2)$

$IHV_3 = CF(IHV_2, M_3)$

:

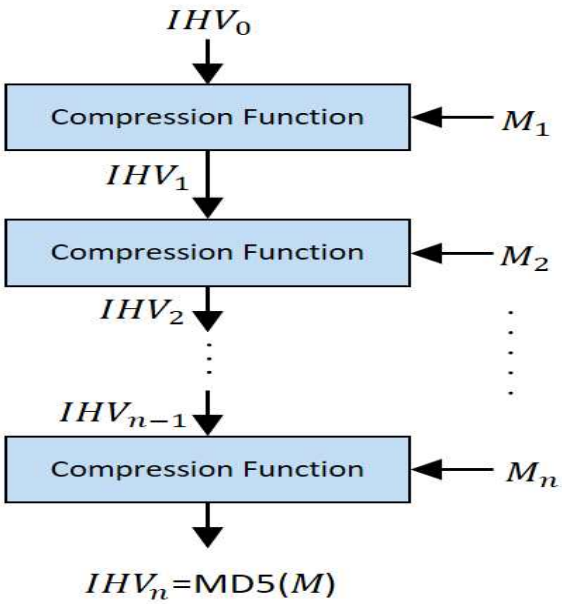
:

:

$IHV_{n-1} = CF(IHV_{n-2}, M_{n-1})$

$IHV_n = CF(IHV_{n-1}, M_n)$

MD5 hash = IHV_n



[그림 1] MD5 알고리즘

여기서 암축함수 $CF: \{0,1\}^{m+t} \rightarrow \{0,1\}^t$는 두 개의 고정 길이 입력 ($m = |M_i|, t = |IHV_i|$)을 고정 길이 출력으로 변환하는 함수이다.

2.2 HMAC

해시함수의 대표적인 응용 중의 하나는 HMAC(Hash-based Message Authentication Code)기법이다. 먼저, MAC(Message Authentication Code)이란 송신자가 보낸 메시지를 수신자 입장에서 송신자가 보낸 메시지가 맞는지(Authentication 인증)그리고 수신 받은 메시지가 변형되지 않았는지(Integrity 무결성)확인하는 방법이다. 이러한 MAC에 해시함수를 응용한 방법이 HMAC이다. HMAC은 2002년 FIPS(Federal Information Processing Standards Publication 미국 연방 정보처리 표준)으로 선정되었다[4].

정의2 HMAC(K, m)=H((K′⊕o-pad) || H((K′⊕i-pad) || m))으로 정의하고, K′= $\begin{matrix} H(K), K\text{ 길이가 블록의 길이보다 클 때} \\ K, K\text{의 길이가 블록의 길이와 같을 때} \end{matrix}$ 이다[1,3]. 여기서,

H= 암호학적 해시함수

m= 메시지

K= 비밀키

|| =연접

⊕= bit-wise XOR

o-pad= 블록길이 만큼의 0x5c가 반복된 패딩

i-pad= 블록길이 만큼의 0x36가 반복된 패딩

2.3 PBKDF2

키 생성함수란 패스워드와 같은 비밀값으로부터 암호키 재료를 유도하는 데 사용되는 알고리즘이다. 본 논문에서 언급할 키 생성함수의 한 종류인 PBKDF2(Password Bases Key Derivation Function 2)에 대해서 살펴본다. PBKDF2는 기반이 되는 의사난수함수(Pseudorandom Function)와 반복 횟수(c)의 선택에 의해 정의된다. PBKDF2의 실행을 위한 입력은 패스워드(P), 솔트(S), 생성할 키(K)의 바이트 길이 정보($dkLen$)이다. 여기서 앞서 소개한 HMAC을 의사난수함수 대신에 사용할 수 있다. PBKDF2는 PKCS(Public-Key Cryptography Standards) #5로 등록된 표준이다[5].

- **입력**
  1. P: 패스워드
  2. S: 솔트
  3. : 반복횟수
  4. $dk\ en$: 생성하고자 하는 키의 길이

- **처리 과정**

  1. 만약 $dkLen > (2^{2}-1)\times hLen$)이면 키의 길이 경고 반환.
     여기서 en은 의사난수함수 출력의 길이

  2. $l = \lceil dkLen/hLen \rceil$, r $= dkLen-(l-1)\times hLen$

---

3. = F (P, S, , 1)

T_2 = F (P, S, c, 2)

.
.

T_i = F (P, S, c, i)

.
.
.

T_l = F (P, S, c, l)을 계산한다.

여기서 F (P, S, c, i) = U_1 U_2 ‥ U_c 을 나타낸다.

U_1 = PRF(P, S||int(i))

U_2 = PRF(P, U_1)

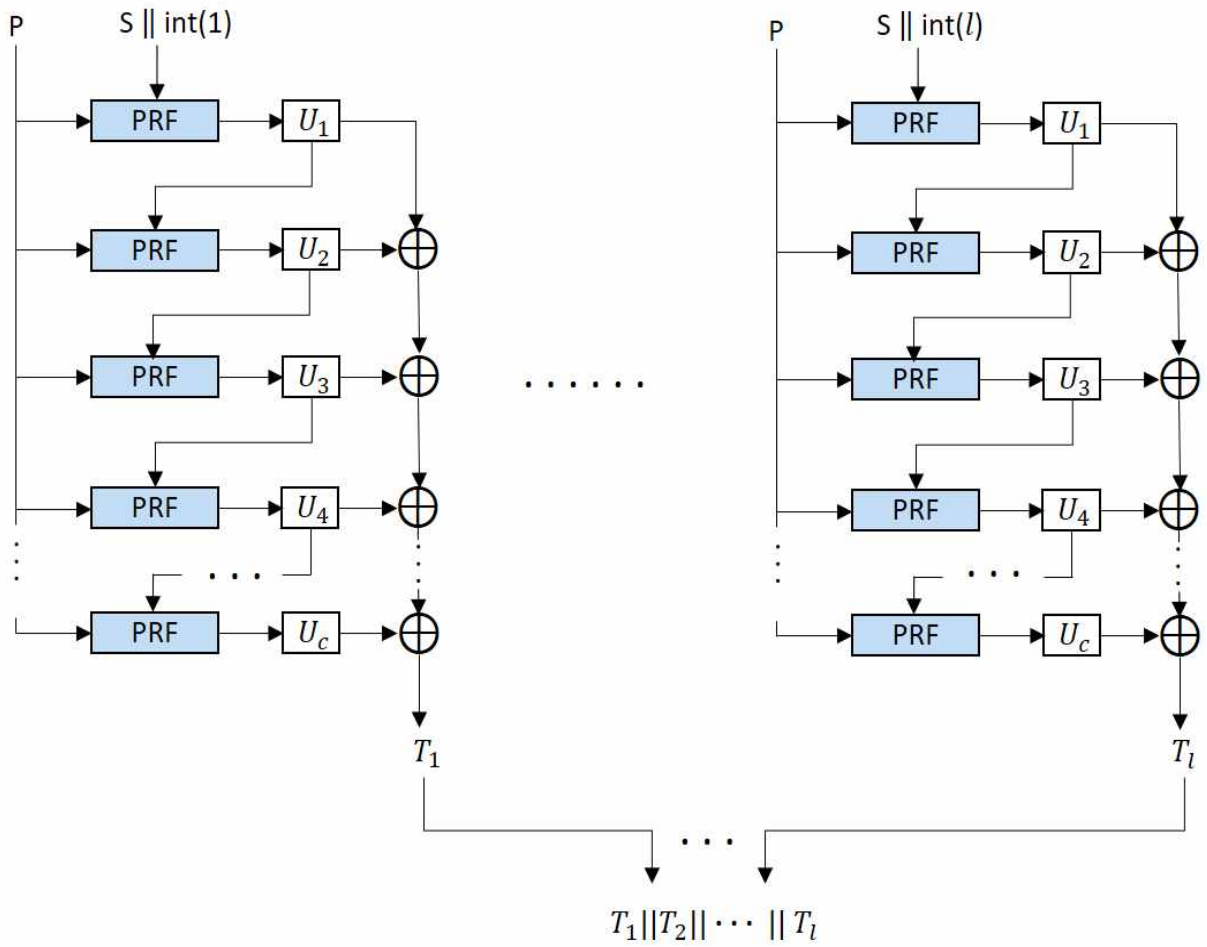
.
.
.

U_c = PRF(P, U_{c-1})

int(i) = 정수i의 32비트 이진수열

4. T_1, T_2, T_3 ,…, T_{l-1} 그리고 T_l에서 처음 r바이트를 추출한 결과를 연접해서 키(K)를 얻는다.

- **출력** : K= 생성된 키



[그림 2] PBKDF2 키 생성과정

2.4 PDF

PDF(Portable Document Format)는 1993년 어도비사(社)가 PDF 1.0버전과 Acrobat 1.0 버전을 발표하면서 세상에 등장하였다. 어도비사에서 Acrobat Reader를 무료배포하고, 적극적으로 PDF를 지원하기 시작하면서 PDF는 인쇄용 문서의 실질적 산업 표준으로 자리 잡았다.

PDF의 가장 큰 특징은 문서 형식이나 제작 기술이 모두 공개되어 있다는 것이다. PDF는 2008년에 ISO(International Organization for Standardization)32000 오픈 포맷으로 표준화되었다. 또한 어도비 시스템즈에서 개발한 일러스트레이터, 포토샵, In-Design 뿐 아니라 다양한 상용 프로그램 및 무료 프로그램에서 PDF형식을 지원하고 있다. 대부분의 문서형식들이 PDF 변환을 지원하고 사용권을 다양하게 부여할 수 있다는 점이 다른 문서 형식과 비교할 때 PDF의 장점이다.

PDF의 암호는 두 가지 종류가 있다. 하나는 권한암호(Owner Password)이고 다른 하나는 사용자암호(User Password) 혹은 파일열기암호이다. 권한암호를 통해서 PDF파일의 수정, 편집 및 인쇄 등의 작업을 제한할 수 있다. 사용자암호는 단순히 PDF파일을 열기 위한 암호이다. 사용자암호가 설정된 PDF파일을 클릭하면, 사용자암호를 물어보는 이벤트가 발생하고 이벤트 창에 사용자암호를 입력하면 PDF파일을 열 수 있다.

PDF의 파일을 열어서 정보를 탈취하는 것이 목적이므로 본 논문에서는 사용자암호를 확인하는 알고리즘을 집중적으로 살펴본다. 어도비사는 PDF Reference[8]를 통해서 PDF의 스펙을 공개하고 있다. 임의의 사용자가 암호를 입력하면, 입력한 암호가 설정된 사용자/권한암호와 같은 암호인지 계산 및 판단하는 알고리즘 역시 공개하고 있다.

PDF 1.6 이상의 버전에서 사용자 암호가 설정되어 있는 PDF파일의 경우, PDF파일 내부에 표준암호사전개체(Standard Encryption Dictionary Object)를 가진다. 먼저, PDF의 사전 개체(Dictionary Object)란 PDF 문서의 주요 구성 요소인데, 일반적으로 문서의 글꼴이나 페이지와 같은 복잡한 개체의 속성을 수집하고 묶는데 사용된다. 이러한 사전개체 중에서 표준암호사전개체는 입력된 암호가 옳은 암호인지 판단하는데 사용할 값들을 저장하고 있는 사전 개체의 한 종류이다. 표준암호사전개체의 주요 구성요소로는 V, U, O, P등의 값이 있다. V-값은 어떤 암/복호화 알고리즘을 사용할지 나타내는 값으로 0부터 4까지의 값을 부여한다. U-값은 옳은 사용자암호를 판정하는데 사용되며, O-값은 옳은 권한암호를 판정하는데 사용된다. P-값은 파일이 열렸을 때, 할 수 있는 작업을 나타내는 값이다.

PDF파일을 열면, 사용자암호를 입력하라는 이벤트창이 발생한다. 사용자는 이벤트 창에 사용자암호(파일열기암호)를 입력한다. 입력 받은 암호를 기반으로 U-값을 계산한다. 계산한 U-값과 PDF파일의 표준암호사전개체에 있는 U-값을 비교하여, 두 값이 같으면 입력 받은 암호를 해당 PDF파일의 옳은 사용자암호로 판단한다.

본 논문에서 다룰 PDF파일들은 PDF1.5 이후의 파일들이다. 이러한 경우 V-값은 4인데, V=4일 때 사용하는 알고리즘들은 PDF Reference[4]의 Algorithm 2: Computing an encryption key, Algorithm 5: Computing the encryption dictionary's U (user password) value (Security handlers of revision 3 or greater), Algorithm 6: Authenticating the user password에 해당한다. 본 논문에서는 비밀번호가 생년월일 6자리인 파일들을 다루기 때문에 이에 맞추어서 위의 알고리즘들을 간략화 하였다. 각 알고리즘은 PDF Reference의 알고리즘들과 순서를 맞추어서 [알고리즘 1], [알고리즘 2], [알고리즘 3]으로 하였다.

[알고리즘 1] 암호화키 계산

(a) 입력한 비밀번호(생년월일 6자리)가 32바이트가 되도록 아래의 32바이트 패딩열의 7번째 바이트부터 가져와서 연접한다. yymmdd를 주어진 비밀번호라고 하면, 32바이트로 패딩된 결과값은 < y y m m d d 8A 41 64 00 4E 56 FF FA 01 08 2E 2E 00 B6 D0 68 3E 80 2F 0C A9 FE 64 53 69 7A >가 된다.

PDF Reference에 정의된 패딩열:
< 28 BF 4E 5E 4E 75 8A 41 64 00 4E 56 FF FA 01 08 2E 2E 00 B6 D0 68 3E 80 2F 0C A9 FE 64 53 69 7A >

(b) MD5 해시함수를 초기화 한다.

(c) 표준암호사전개체의 O-값을 (a)의 결과와 연접한다.

(d) 표준암호사전개체의 P-값을 부호가 없는 2진수로 바꾼 후 low-order바이트를 먼저 넣어서, (c)의 결과와 연접한다.

(e) PDF파일의 ID-값의 첫 번째 부분을 (d)의 결과와 연접한다. ID-값은 PDF파일의 가장 마지막부분에 위치한다.

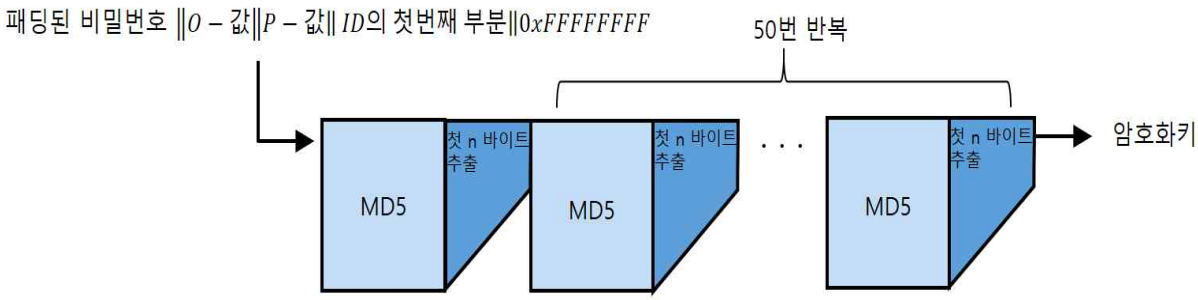
[알고리즘 1] 암호화키 계산

(f) PDF파일의 메타데이터가 암호화 되어 있지 않을 경우에는 (e)의 결과에 0xFFFFFFFF을 연접한다.

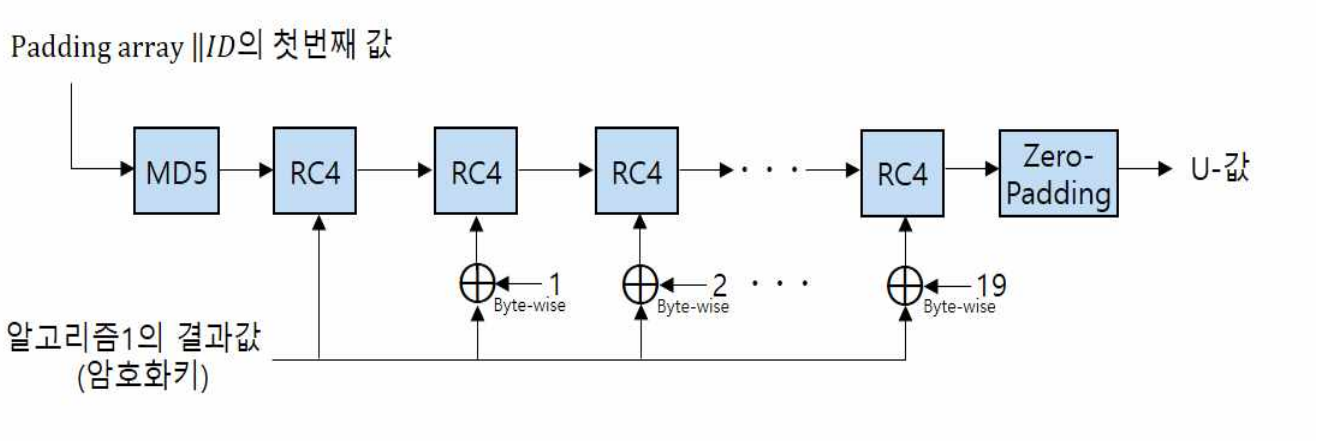
(g) (f)의 결과를 MD5에 입력하여 해시값을 얻는다.

(h) MD5를 50번 반복한다: 여기서 n을 표준암호사전개체의 Length-값이라 고할 때, 이전 MD5 해시값의 처음 n 바이트를 다음 MD5의 입력으로 집어넣는다. 이러한 과정을 50번 반복한다

(i) n을 표준암호사전개체의 Length-값이라고 할 때, (h)의 결과값에서 처음 n바이트를 암호화키로 사용한다.



[그림 3] 알고리즘 1 : 암호화키 생성과정



[그림 4] 알고리즘 2 : U-값 계산

[알고리즘 2] 암호사전의 U-값 계산

(a) 비밀번호(생년월일 6자리)를 기반으로 [알고리즘 1]을 이용하여 암호화키를 생성한다.

(b) MD5를 초기화한다.

(c) [알고리즘 1]에서 언급한 32바이트 패딩열과 PDF파일의 ID-값의 첫 번째 부분을 연접하여 MD5의 입력으로 집어넣어 MD5 해시값을 얻는다.

(d) (c)의 결과를 평문, (a)의 결과를 키로 하여 RC4을 계산한다.

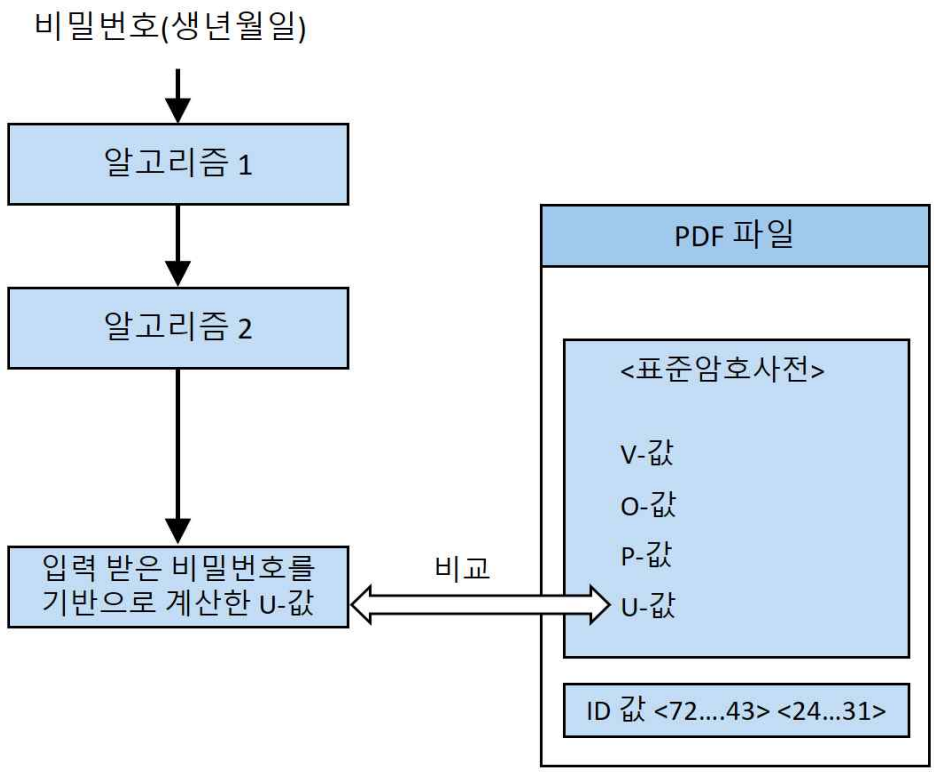
(e) RC4를 19번 반복한다: 평문은 이전 RC4의 결과값을 사용하고, 키는 (a)의 결과값인 암호화키의 각 바이트와 반복횟수 i를 XOR한 값을 사용한다.

(f) (e)의 결과값에 16바이트의 0을 덧붙여서 U-값을 얻는다.

[알고리즘 3] 사용자암호 판정

(a) 입력한 비밀번호를 바탕으로 [알고리즘 2]를 수행한다.

(b) (a)의 결과값이 표준암호사전개체의 U-값과 같다면 입력한 비밀번호를 옳은 사용자암호로 판정한다.



[그림 5] 알고리즘 3 : 계산한 U-값과 내부의 U-값 비교

제 3 장 전수조사공격

3.1 PDF파일에 대한 전수조사공격

3.1.1 준비과정

PDF파일에 대한 전수조사 공격을 수행하기 위한 프로그래밍 환경을 다음과 같이 설정한다. 운영체제는 Microsoft사의 Windows 10 64비트를 사용하였으며, CPU는 Intel사의 Core(TM) i5-6200U CPU @ 2.30 GHz 듀얼코어 4-논리프로세서, 8GB의 메모리를 사용하였다. 3.1.3에서 대량의 PDF파일을 다루기 위한 C-programming은 Microsoft Visual Studio 2019에서 수행하였다.

3.1.2 하나의 파일에 대한 전수조사공격

QPDF는 무료 command line 프로그램으로, 선형화(웹 최적화 또는 빠른 웹 보기라고도 한다), 암호화 및 PDF파일의 암호 복호화와 같은 다양한 변환을 수행할 수 있다. QPDF에는 PDF 파일 중 한 파일에서 다른 PDF파일로 개체를 복사하고 PDF파일의 페이지 목록을 조작하는 기능을 통해 PDF를 회전, 병합 및 분할이 가능하다. 본 논문에서는 PDF파일의 비밀번호를 찾기 위하여 QPDF8.3.0의 복호화 기능을 사용한다. QPDF는 복호화 명령어에 생년월일(비밀번호)와 복호화 하고자 하는 PDF파일의 이름을 입력하면, 비밀번호가 제거된 PDF파일을 얻을 수 있다. 반면에 옳지 않은 비밀번호를 입력한 경우에는 command창에 invalid password라는 알림을 띄운다.

하나의 PDF파일의 옳은 비밀번호와 decrypted file을 얻기 위해서 공격범위를 설정하였다. 공격범위은 2019년을 기준으로, 휴대전화를 사용하는 20대부터 80대까지로 하였다. 따라서 생년은 40년부터 99년까지, 생월은 1월부터 12월까지, 마지막으로 생일은 1일부터 31일까지를 범위로 가지는 변수로 설정하였다. 공격대상파일은 2019년 2월에 발행된 KT사의 통신요금명세서 PDF일로 하였다. 간단한 DOS 배치파일을 이용하여 40년 1월 1일부터 99년 12월 31일까지 차례대로 모든 가능한 경우의 수에 대하여 QPDF를 실행하였다.

QPDF의 복호화 명령어

qpdf --password=생년월일 6자리 --decrypt '복호화할 파일이름'.pdf '복호화 후 저장할 파일이름'.pdf

[그림 6] QPDF의 복호화 기능 사용방법

위의 [그림 6]의 예에서는 복호화 할 파일의 이름은 kimmin_kt.pdf 이고 복호화 한 후에 저장할 파일의 이름은 decrypted_kimmin_kt.pdf 로 하였다.

QPDF를 이용해서 하나의 공격대상 PDF파일을 복호화 하는데 약 4분 30초~22분이 소요되었다.

파일 \ 관찰	1	2	3	4	평균
비밀번호 500101인 PDF	260s	275s	279s	280s	273s
비밀번호 600101인 PDF	660s	672s	657s	677s	665s
비밀번호 700101인 PDF	847s	859s	850s	831s	846s
비밀번호 800101인 PDF	1127s	1135s	1124s	1132s	1129s
비밀번호 900101인 PDF	1269s	1271s	1260s	1274s	1268s

[표 1] QPDF를 이용한 생년월일별 복호화 소요시간

3.1.3 대량의 파일에 대한 전수조사공격

앞 절에서 살펴본 바와 같이 배치파일과 QPDF만을 이용하여서는 하나의 PDF파일을 복호화 하는데 평균 14분이 소요되어서 대량의 파일에 대한 효과적인 전수조사 공격이 불가능 하였다. 대량의 PDF파일에 대한 효과적인 전수조사공격을 위하여 C-programming을 통하여 [알고리즘 1], [알고리즘 2], [알고리즘 3]을 구현하였다. 특히, MD5와 RC4를 사용하는 [알고리즘 1]과 [알고리즘 2]는 OpenSSL1.0.2를 이용하여 구현하였다. 하나의 생년월일에 대해서 [알고리즘 1], [알고리즘 2], [알고리즘 3]을 통하여 U-값을 계산한다. 한편, PDF파일의 표준암호화사전을 찾고 표준암호화사전 내부에 있는 U-값을 찾는다. 계산한 U-값이 PDF파일 내부의 U-값과 같은지 비교한다. 만약, 두 값이 같을 경우 시도한 생년월일이 PDF파일의 옳은 비밀번호이다. 앞 절과 마찬가지로 생년월일은 40년 1월 1일부터 99년 12월 31일까지 차례대로 대입하였다.

비밀번호의 조사구간이 40년부터 99년이므로 그 중간 값인 70년대 비밀번호를 가지는 파일을 기준으로 C-programming을 통해서 옳은 비밀번호를 찾는데 약 0.5초가 소요되었다. 또한 10개의 파일의 옳은 비밀번호를 찾는데 약 5초의 시간이 소요되었다. 따라서 대량의 PDF보안명세서가 피싱업자의 수중에 들어가면, 하나의 노트북만 사용하여도, PDF파일의 경우 10만개의 파일을 복호화 하는데 겨우 13시간이 소요된다. 복호화를 통해서 오히려 사용자의 생년월일 정보가 유출되어 보안기능으로 작동하지 않을 뿐만 아니라 보안약점으로 작용할 수 있음을 알 수 있다.

파일 \ 관찰	1	2	3	평균
비밀번호 500101인 PDF	0.284s	0.294s	0.282s	0.286s
비밀번호 600101인 PDF	0.389s	0.383s	0.375s	0.382s
비밀번호 700101인 PDF	0.473s	0.472s	0.475s	0.473s
비밀번호 800101인 PDF	0.560s	0.562s	0.565s	0.562s
비밀번호 900101인 PDF	0.665s	0.655s	0.676s	0.665s

[표 2] C-programming을 이용한 생년월일별 복호화 소요시간

[코드 1] 알고리즘 1 C-code

```
MD5_Init(&ctx);// MD5 initialize

MD5_Update(&ctx, user_password, 6);// pass the user password

MD5_Update(&ctx, padding, 26);// pass the padding array

MD5_Update(&ctx, O, 32);// pass the O

MD5_Update(&ctx, P, 4);// pass the P

MD5_Update(&ctx, ID, 16);// pass the ID

MD5_Update(&ctx, meta_false_padding, 4);// pass 0xFFFFFFFF

MD5_Final(output_MD5, &ctx);// finish the hash
```

[코드 2] 알고리즘 2 C-code

```
---------알고리즘2--step(a)----------
for (i = 0; i < 16; i++){
key_data[i] = output_MD5[i];}

--------알고리즘2--step(b)----------
MD5_Init(&ctx);// initialize

--------알고리즘2--step(c)-----------
MD5_Update(&ctx, padding, 32);
MD5_Update(&ctx, ID, 16);
MD5_Final(output_MD5, &ctx);

--------알고리즘2--step(d)-----------
RC4_set_key(&rc4_key, 16, key_data);
RC4(&rc4_key, 16, output_MD5, final);

--------알고리즘2--step(e)-----------
for (k = 1; k <= 19; k++)
{ for (j = 0; j < 16; j++)
   {
     key_RC4[j] = key_data[j] ^ k;
   }
    RC4_set_key(&rc4_key, 16, key_RC4);
    RC4(&rc4_key, 16, final, final);
}
```

주의 [알고리즘 2]에서 주의할 점은 (e)과정이다. RC4를 19번 반복할 때, 각 i번째 반복에서 사용할 RC4의 키는 (a)의 결과값과 i를 XOR해서 구한다. 즉, T := MD5(padding || ID)

for i = 0 to 19

{ K_i := K_u ^ i

T := RC4(K_i, T) }

3.2 HTML파일에 대한 전수조사공격

3.2.1 준비과정

HTML보안명세서는 Active-X 플러그인을 설치해야 보안명세서를 열어 볼 수 있었다. 따라서 사용자의 불편함이 큰 문제였다. 하지만, 여러 보안메일 솔루션 업체들이 Active-X 플러그인의 설치 없이도 열람이 가능하도록 서비스를 제공하고 있다. 적지 않은 공공기관, 카드회사, 이동통신사들이 보안명세서를 HTML 형식으로 제공하고 있기 때문에 HTML 보안명세서에 대한 전수조사공격 역시 시도하였다.

HTML보안명세서의 경우 회사들이 각자 자신들의 보안메일솔루션을 사용하여 명세서를 암호화하기 때문에 PDF와는 달리 복화하는 방법 역시 모두 달랐다. 먼저 공공기관, 통신사, 카드회사의 경우를 대상으로 사용하는 보안메일솔루션들을 알아보고 그에 대한 전수조사공격을 시도 하였다. 대표적으로 신한카드사의 이용대금명세서에 사용된 UnisafeMail, 서울시 지방세고지서에 사용된 Vestmail, LGU+의 요금명세서에 사용된 XecureExpress를 대상으로 하였다.

HTML파일에 대한 전수조사공격은 PDF파일에서와 마찬가지로 생년, 생월, 생일을 변수로 하여 각각 40년~99년, 1월~12월, 1일~31일까지를 차례대로 대입하여 옳은 비밀번호를 확인하였다. 위의 세 경우의 보안메일 솔루션에 대한 전수조사공격을 위하여 사용한 자바스크립트 코드는 아래와 같다.

[코드 3] 자바스크립트 코드

```
function brute()
   {
     for (var y=40; y<100; y++) //y=생년
      {
         for (var m=1; m<=12; m++) //m=생월
          {
            var tmpo = m+"";
            if (tmpo. length == 1)
                 tmpo = tmpo+ "0";
            m = tmpo;
            for (var d=1; d<=31; d++) //d=생일
               {  var tmpo = d+"";
                if (tmpo. length == 1)
                    tmpo = tmpo+ "0";
                    d= tmpo;
                보안메일솔루션들의 비밀번호 판정함수부분을 대입
                if (비밀번호 판정함수!=NULL)
                { console.log(y+m+d);
                  return y+m+d;
                 }
               }
           }
        }
      }
```

3.2.2 UnisafeMail

2019년도 3월에 발송된 신한카드사의 보안명세서에 사용된 UnisafeMail에 대한 전수조사공격을 알아본다. 신한카드 보안명세서의 코드를 확인하면, 사용한 알고리즘을 알아보기 힘들게 하기 위해서 함수이름과 내용이 난독화 되어 있다. PDF의 경우와 마찬가지로, HTML명세서 역시, 입력받은 생년월일 비밀번호를 기반으로 값을 계산하고, 그 값이 옳은 값인지 판단하는 과정을 거친다. 따라서 전수조사공격을 위해서는 앞서 언급한 판단하는 부분을 찾아서 전수조사공격을 수행한다.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ko" lang="ko">
<head>
	<title></title>
	<meta http-equiv="Content-Type" content="text/html; charset=euc-kr" />
<script type="text/javascript" charset="UTF-8">
/* <![CDATA[ */
/* 20140701 */

var _0xcdb6=["₩x73₩x74₩x72₩x32₩x62₩x69₩x6E","₩x62₩x69₩x6E₩x32₩x73₩x74₩x72","₩x68₩x65₩x78₩x32₩x62₩x69₩x6E","₩x62₩
65₩x6E₩x74₩x73₩x42₩x79₩x54₩x61₩x67₩x4E₩x61₩x6D₩x65","₩x73₩x63₩x72₩x69₩x70₩x74","₩x63₩x72₩x65₩x61₩x74₩x65₩x45
₩x65₩x73₩x6F₩x66₩x74₩x2E₩x63₩x6F₩x6D₩x2E₩x20₩x41₩x6C₩x6C₩x20₩x72₩x69₩x67₩x68₩x74₩x73₩x20₩x72₩x65₩x73₩x65₩
5F₩x74₩x65₩x73₩x74","₩x6D₩x65₩x73₩x73₩x61₩x67₩x65₩x20₩x64₩x69₩x67₩x65₩x73₩x74","₩x66₩x37₩x38₩x34₩x36₩x66₩x3
53₩x69₩x7A₩x65₩x2E","₩x28₩x45₩x72₩x72₩x6F₩x72₩x29₩x65₩x6E₩x63₩x72₩x79₩x70₩x74₩x53₩x65₩x65₩x64₩x32₩x20₩x3A₩
x4F₩x50₩x59₩x52₩x49₩x47₩x48₩x54","₩x43₩x6F","₩x70₩x79₩x72","₩x69₩x67","₩x68₩x74₩x28","₩x43₩x29₩x20₩x32","₩x31","₩x3
} ;this[_0xcdb6[3]]=function (_0x9319x2){return _0x9319xb(_0x9319x2);} ;this[_0xcdb6[4]]=function (_0x9319x2){return _0x9319xc(_0x9319
6[17]](/[^A-Fa-f0-9]/g,_0xcdb6[11]);if(_0x9319xa[_0xcdb6[9]]%2){return ;} ;var _0x9319x7=[];for(var _0x9319x5=0;_0x9319x5<_0x9319xa[_(
2>>6);_0x9319x16=_0x9319x12&63;if(isNaN(_0x9319x11)){_0x9319x15=_0x9319x16=64;} else {if(isNaN(_0x9319x12)){_0x9319x16=64;} ;}
-Za-z0-9₩+₩/₩=₩/n]/g,_0xcdb6[11]);do{_0x9319x13=_0x9319xe[_0xcdb6[14]](_0x9319xd[_0xcdb6[22]](_0x9319x5++));_0x9319x14=_0x9
0x9319x1c[_0xcdb6[9]];_0x9319x1e++){var _0x9319x18=_0x9319x1c[_0xcdb6[10]](_0x9319x1e);if(_0x9319x18<128){_0x9319x1d+=String[_
```

[그림 7] 신한카드 보안명세서 부분코드

[코드 4] UnisafeMail 비밀번호 판단하는 부분코드

```
this['unisafe_smail_process'] = function(_0x9319xd9)
    {     var _0x9319xda = 17;
          _0x9319xd7 = this.COPYRIGHT();
          var _0x9319xdb = _0x9319xdf(_0x9319xd9, _0x9319xda);
          var _0x9319xdc = _0x9319xe7(UNISAFESMAIL_DATA, 15);
          if (_0x9319xdc['length'] < 1) {
               var _0x9319xdd = unisafe_smail_checkValue(_0x9319xdc);
               _0x9319xdc = _0x9319xdd;
          };
          var _0x9319xde = _0x9319xe6();
          if (_0x9319xde['substring'](0, 10) == '<!--LIC_TO') {
               _0x9319xe5(_0x9319xd6);
          }  else {aler(_0xcdb6[123])};
      };
```

UnisafeMail().unisafe_smail_process을 계산하여 옳은 비밀번호를 판단한다. 따라서 앞서 작성한 자바스크립트코드에 UnisafeMail().unisafe_smail_process을 대입한 후, 전수조사공격을 수행하여 옳은 비밀번호를 찾을 수 있었다. 또한, 신한카드 보안명세서의 비밀번호를 분석 및 크랙하는 블로그 글을 참고하여 교차검증을 수행하였다[2]. 이 경우 한 파일의 비밀번호를 알아내는데 약 65초가 소요되었다.

관찰 / 파일	생년월일=900922
1	65.32s
2	63.73s
3	64.50s
4	67.12s
5	68.55s
평균	65.42s

[표 3] UnisafeMail을 이용한 HTML 파일의 복호화 소요시간

3.2.3 Vestmail

2019년도 서울시 지방세 보안명세서에 사용된 Vestmail에 대한 전수조사공격을 알아본다. 방법은 UnisafeMail의 경우와 유사하게, 입력한 비밀번호가 옳은 비밀번호인지 체크하는 부분을 찾은 후 전수조사공격을 수행한다.

[코드 5] Vestmail 비밀번호 판단하는 부분 코드

```
function Q(b)
 {    var k = x.b.l(s[b]);
       l[b] = x.y.n(k, z, { c: new x.c.l(x.p.ba), g: G });
         if (null == l[b])
           { try { vestmail_onend(!1)}
              catch (p) { L(!1)}
              alert(vestmail_msg_wrong_password);
               H = !0
           } else C = setTimeout(O, 10)
      }
```

Vestmail의 경우에는 함수의 이름을 단순화하고, 각 함수의 역할을 알기 힘들게 난독화를 해두었다. 따라서 크롬이 지원하는 자바스크립트 디버깅 기능을 이용하여 입력한 비밀번호가 들어가는 함수들을 추적하여 비밀번호를 판단하는 부분을 유추하였다.

앞서 작성한 자바스크립트 코드에 [코드5]의 코드를 대입한 후, 전수조사공격을 수행하여 옳은 비밀번호를 찾을 수 있었다. 생년월일의 중간값인 70년대의 생년월일이 옳은 비밀번호일 때, 한 파일의 비밀번호를 알아내는데 약 10초가 소요되었다.

관찰 / 파일	생년월일=720102
1	10.32s
2	10.57s
3	10.42s
4	10.51s
5	10.59s
평균	10.48s

[표 4] Vestmail을 이용한 HTML 파일의 복호화 소요시간

3.2.4 XecureExpress

LGU+ 보안명세서에 사용된 XecureExpress에 대한 전수조사공격을 알아본다. 공격대상파일은 2019년 9월에 발행되었으며, 90년대 생년월일을 비밀번호로 가지는 LGU+ 보안명세서를 대상으로 하였다. 분석방법은 앞의 두 경우와 유사하게, 입력한 비밀번호가 옳은 비밀번호인지 체크하는 부분을 찾은 후 전수조사공격을 수행한다.

먼저, 가장 쉽게 알 수 있는 XecureExpress.decryptFunc함수에 비밀번호를 입력해서 보안메일을 복호화 할 수 있었다. 하지만, 80년대~90대의 생년월일을 계산하는데 소요시간은 약 16분으로 앞의 두 경우보다 시간이 많이 소요되었다. 따라서 많은 시간이 소요되는 원인을 찾기 위해서 decryptFunc의 내부를 분석하였다.

decryptFunc의 내부에는 enveloptedPwd함수와 encryptedPwd함수가 있고 디버깅을 통하여 그 중에서 enveloptedPwd함수를 사용함을 확인하였다. enveloptedPwd함수 내부에서는 입력받은 비밀번호를 기반으로 PBKDF2를 사용하여 derivation key를 생성한다. 생성한 derivation key를 기반으로 HMAC 알고리즘을 사용하는 것을 알 수 있었다.

시간이 소요되는 부분을 파악하기 위해서 PBKDF2의 반복횟수( )를 살펴보았다. PBKDF2는 반복횟수를 최소한 1000이상으로 하기를 권고한다[4]. 따라서 1000번 이상의 반복 횟수를 예상하였으나 분석결과 반복횟수가 100번에 불과하였다. 또한, 반복횟수의 변화에 따른 함수의 소요시간이 큰 차이가 없었기 때문에, HMAC알고리즘을 수행하는데 시간이 소요됨을 알 수 있었다.

관찰 / 파일	10년간의 생년월일
1	975s
2	971s
3	981s
4	985s
5	973s
평균	977s

[표 5] XecureExpress를 이용한 HTML 파일의 복호화 소요시간

[코드 6] XecureExpress 비밀번호 판단하는 부분 코드

```
plainText = XecureExpress.decryptFunc(inputText);
if (plainText != null) {
        var userAgent = navigator.userAgent;
        var xemframe = document.getElementById("XEMFrame");
        if (!xemframe) contentType = 2;
        if(contentType==2||userAgent.match(/iPhone|iPod|iPad|
          Android|Windows CE|BlackBerry|Symbian|Window)!=
          null||userAgent.match(/LG|SAMSUNG|Samsung/) != null)
          { onEnd();
            document.write(plainText);
            document.close(plainText)}
         else { xemframe = document.getElementById("XEMFrame");
                xemframe.style.display = "block";
                xemContent = xemframe.contentWindow.document ||
                    xemframe.contentDocument;
                xemContent.open();
                xemContent.write(plainText);
                xemContent.close();
              }
        }
else {  onEnd();
        var errorMsg = null;
        if (pwdType == 1) errorMsg = "invalidInputPWD";
        else if (pwdType == 2) errorMsg = "idOrPwdInputFail";
        alert(errorCallBack(errorCodeList[errorMsg]));
    }
```

[코드 7] Softforum의 HMAC이 사용된 부분코드

```
var salt =
recipientInfos.sub[0].sub[2].sub[1].sub[1].sub[0].sub[1].sub[0].content
().split(" ")[2];
var iterationCount =
recipientInfos.sub[0].sub[2].sub[1].sub[1].sub[0].sub[1].sub[1].content
();
var iv =
recipientInfos.sub[0].sub[2].sub[1].sub[1].sub[1].sub[1].content().split("
")[2];
var encryptedKey = recipientInfos.sub[0].sub[3].content().split(" ")[2];
var derivationKey =
algorithmList[oids[recipientInfos.sub[0].sub[2].sub[1].sub[0].content()].d
](SofoJS.SHA1(pwd), SofoJS.enc.Hex.parse(salt), { keySize: 128 / 32,
                    iterations: iterationCount }); undefined(iterationCount);
var hMacKeyAndDecryptedKey =
algorithmList[oids[recipientInfos.sub[0].sub[2].sub[1].sub[1].sub[1].sub[
0].content()].d].decrypt({ciphertext:SofoJS.enc.Hex.parse(encryptedKey)},
derivationKey, {iv: SofoJS.enc.Hex.parse(iv)});
var hMacKey =
SofoJS.enc.Hex.parse(hMacKeyAndDecryptedKey.toString().slice(32));
var decryptedKey =
SofoJS.enc.Hex.parse(hMacKeyAndDecryptedKey.toString().slice(0, 32));
var encryptedData =
SofoJS.enc.Latin1.parse(encryptedDataInfos.sub[2].contentRaw());
iv = encryptedDataInfos.sub[1].sub[1].content().split(" ")[2];
var hMacAndPlainText =
algorithmList[oids[encryptedDataInfos.sub[1].sub[0].content()].d].decrypt
({ciphertext: encryptedData}, decryptedKey, {iv: SofoJS.enc.Hex.parse(iv)
});
var hMacData = hMacAndPlainText.toString().slice(-40);
plainText =
SofoJS.enc.Hex.parse(hMacAndPlainText.toString().slice(0,-40));
var hashValue = SofoJS.HmacSHA1(plainText, hMacKey);
if (hashValue.toString() != hMacData.toString()) {
                    plainText = null;
                    throw errorMsg = "invalidInputValue"; }
```

제 4 장 결론

4.1 PDF파일과 HTML파일에 대한 공격

생년월일을 비밀번호를 가지는 보안명세서 파일들의 복호화가 수 초 이내로 가능한 쉬운 일이라는 것을 알아보았다. 복호화를 통해서 오히려 사용자의 생년월일 정보가 유출되어 보안기능으로 작동하지 않을 뿐만 아니라 보안약점으로 작용할 수 있음을 알 수 있었다.

대량의 보안명세서가 피싱업자의 수중에 들어가면, 하나의 노트북만 사용하여도, PDF파일의 경우 10만개의 파일을 복호화 하는데 약 13시간 이면 충분하다. 비교적 복호화 시간이 더 소요되는 HTML파일의 경우에는 1만개의 파일을 복호화 하는데 약 160시간이 소요된다. 따라서 이러한 보안명세서의 생년월일 암호화는 충분한 컴퓨팅 파워를 가지고 있는 피싱업자의 공격에 대해 전혀 안전하지 않다.

명세서파일은 사용내역, 사용위치, 사용금액등 개인정보를 유추할 수 있는 정보들을 포함하고 있다. 따라서 이러한 전수조사공격으로 인하여 유출된 생년월일 정보는 명세서파일의 내용에 더하여 보이스피싱 등에 악용 될 수 있다.

4.2 제안

가장 단순한 개선방법은 비밀번호를 늘려서 사용하는 것이다. 생년월일과 휴대전화번호의 뒷자리를 합하여 총 10자리의 비밀번호를 사용한다면, 가능한 비밀번호의 경우의 수는 223,200,000이므로 하나의 70년대 생년월일을 가지는 PDF파일을 기준으로 약 1시간 38분이 소요된다.

XecureExpress의 경우와 같이 HMAC 기법을 활용하거나, PBKDF2의 반복횟수를 늘려서 계산시간을 증가시킬 수도 있다. 하지만 계산시간이 증가하거나 파일의 크기가 커지면 모바일 기기에서 사용하는데 불편함이 따르므로 적절한 균형을 찾아야한다.

마지막으로, PROOF OF WORK개념을 응용하는 방법도 있다. KT에서 명세서를 발급할 때, 4자리 난수를 발생시켜서 PDF파일을 전송한다. 올바른 사용자는 자신의 6자리이외에 난수 4자리만 시도해보면 보안명세서를 열 수 있다. 하지만, 공격자는 공격할 생년월일 6자리와 난수 4자리, 총10자리를 시도해야한다. 이 방법의 경우 올바른 사용자의 입장에서 난수4자리를 일일이 대입할 수 없으므로 명세서파일에 특화된 PDF리더를 배포해야한다는 단점이 있다.

참 고 문 헌

[1] 김명환, 수리암호학 개론, 경문사, 2019.

[2] 임준오, “신한카드 이용대금 명세서 개인 비밀번호 분석 및 크랙”, 2019년 10월 1일 접속, https://imjuno.com/2018/01/01/

[3] D. R. Stinson, Cryptography Theory and Practice, Champman & Hall/CRC, 2018.

[4] H. Krawczyk, M. Bellare, R. Canetti, HMAC keyed-hashing for message authentication. RFC 2104, 1997.

[5] K. Moriarty, B. Kaliski, A. Rusch, PKCS #5:password-based cryptography specification version 2.1. RFC 8018, 2017.

[6] R. Rivest, The MD5 message-digest algorithm. RFC 1321, 1992.

[7] X. Wang, H. Yu, How to break MD5 and other hash functions, Advances in Cryptology-Eurocrypt 2005, volume 3494 of Lecture Notes in Computer Science, pp.19-35, Springer, 2005.

[8] ISO 32000-1:2008, Document management－Portable document format－Part 1:PDF 1.7, 2008.

Abstract

Brute Force Attacks on encrypted billing statements with password based on birth date

Min Kim

Department of Mathematical Sciences

The Graduate School

Seoul National University

Telecommunications companies and credit card companies provide services to notify a customer by attaching encrypted billing statement file to customer's e-mail. The formats of attachment used here are PDF and HTML, and the billing statement file is encrypted with 6 digits of the customer's birth date(YY-MM-DD), thus limiting access to third parties other than the customer. In this paper, we introduce the Brute Force Attack methods, which are used to decrypt the billing statement files, and we will see that if large volumes of the statement files are leaked, damage from serious personal information leakage can occur.

keywords : Brute Force Attack, billing statement, PDF, HTML

Student Number **: 2018-23801**