# Cryptographic Service Provider(CSP) 모듈의 개인키/비밀키 노출 취약점에 대한 실험적 연구

박진호* · 조재익* · 임을규*

**요 약**

Windows 운영체제는 응용프로그램이 쉽고 편리하게 다양한 암호화 알고리즘을 사용할 수 있도록 CSPs(Cryptographic Service Providers)를 제공하고 있으며, 이 응용프로그램은 CryptoAPI (Cryptographic Application Program Interface)를 통하여 선택적으로 다양한 CSP 모듈을 이용할 수 있다. 이 때 CryptoAPI와 CSP 모듈 사이에서 함수로 전달되는 파라미터의 분석을 방지하기 위해 핸들을 이용한 안전한 데이터 접근 방식을 사용하고 있다. 본 논문에서는 새롭게 제시한 메모리 역추적 기법을 통해 위와 같은 보안기법에서도 개인키/비밀키와 같은 중요한 데이터가 노출되는 취약점이 존재한다는 것을 실험을 통해 증명하였다.

# An Experimental Study of Private Key and Secret Key Disclosure Vulnerability in Cryptographic Service Provider(CSP) Module

Jin Ho Park* · Jae Ik Cho* · Eul Gyu Im*

**ABSTRACT**

In Windows operating system, CSPs(Cryptographic Service Providers) are provided for offering a easy and convenient way of using an various cryptographic algorithms to applications. The applications selectively communicate with various CSPs through a set of functions known as the Crypto API(Cryptographic Application Program Interface). During this process, a secure method, accessing data using a handle, is used in order to prevent analysis of the passing parameters to function between CryptoAPI and CSPs. In this paper, our experiment which is using a novel memory traceback method proves that still there is a vulnerability of private key and secret key disclosure in spite of the secure method above-mentioned.

Key words : Cryptographic Service Provider(CSP), Cryptographic Application Program Interface(CryptoAPI)

* 한양대학교 전자컴퓨터통신공학과

## 1. 서 론

컴퓨터와 네트워크의 급속한 발전은 언제 어디서나 자신이 원하는 정보를 얻을 수 있는 유비쿼터스 시대의 초석이 되었다. 그러나 이러한 발전으로 인해 타인의 컴퓨터에 원격으로 접근하는 방법도 발달하였으며, 이를 이용하여 타인의 컴퓨터에 저장된 중요한 데이터를 훔치는 악의적인 목적의 크래킹도 증가 추세이다. 최근 미국에서 진행된 연구에 따르면, 악의적인 목적을 가진 공격자들이 매 39초마다 거의 일정한 주기로 인터넷 접속을 통한 컴퓨터 공략에 나서고 있다는 분석도 있다[8]. 이에 컴퓨터 보안의 중요성이 강조되는 가운데, 컴퓨터의 운용에 필수적인 운영체제의 보안 기능들도 강화되고 있다.

Microsoft사의 Windows 운영체제는 시스템 및 응용프로그램에 보안 기능을 제공하기 위하여 다양한 암호화 알고리즘이 구현되어 있는 CSP(Cryptographic Service Provider) 모듈과 CryptoAPI(Cryptographic Application Program Interface)를 제공한다. Windows 기반의 응용프로그램들은 CryptoAPI를 통하여 CSP에 구현되어 있는 암호화 알고리즘을 사용할 수 있으며, 이를 이용하여 응용 프로그램은 쉽게 인증 및 암호화 과정을 수행할 수 있다. Microsoft에서는 보안성 강화를 위해 CryptoAPI와 CSP 사이에 중간 계층을 두고, 서로 다른 핸들을 사용하게 함으로써 함수가 호출되는 과정에서 암호화 관련 데이터들이 노출되지 않게 설계했다. 그러나 본 논문에서는 저자가 새롭게 고안한 메모리 역추적 기법 사용하여 Windows 운영체제에서 사용하는 보안 기법이 키 값을 노출시키는 보안 취약성을 가지고 있다는 것을 실험을 통해 증명하였다.

본 논문에서는 먼저 CryptoAPI와 CSP에 대해서 제 2장에서 살펴보고, 제 3장에서는 CSP의 취약점 분석 기법에 대해서 설명한 후, 제 4장에서는 제안한 방법을 이용하여 데이터 노출 과정을 실험을 통해 증명하며, 제 5장에서 본 논문의 제약조건을 언급한 후, 제 6장에서 결론을 서술한다.

## 2. 관련 연구

### 2.1 CryptoAPI

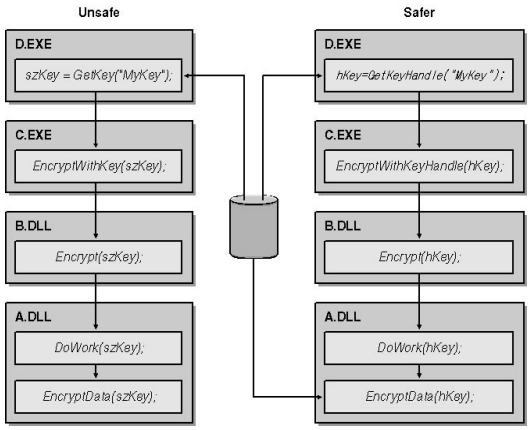
CryptoAPI는 Microsoft Windows 운영체제의 일부분으로 제공되는 응용 프로그램 인터페이스로써, 이를 통하여 응용 프로그램은 쉽게 사용자 키 데이터 보호와 다양한 암호화 서비스를 제공 받을 수 있으며, 이는 실제로 Windows에 등록되어 있는 CSP 모듈 안에서 수행된다. CryptoAPI의 주요 기능은 다음과 같다.

- 암호 알고리즘에 대한 인터페이스 제공
- 스마트 카드와 같은 하드웨어 컴포넌트에 대한 인터페이스 제공
- 사용자 개인 키를 사용한 전자 서명 등의 경우, 사용자와 직접 통신할 수 있는 인터페이스 제공
- Win32 응용프로그램에 암호를 추가할 수 있는 서비스에 대한 인터페이스 제공

CryptoAPI는 인자값을 전달할 때 보안성 향상을 위해 변수값이 아닌 핸들값을 전달하며, (그림 1)은 CryptoAPI가 사용하는 안전한 데이터 접근 방법(그림의 오른쪽)과 안전하지 못한 방법(그림의 왼쪽)을 비교하고 있다.

(그림 1)에서 볼 수 있듯이 좌측의 방법은 함수에서 함수로, 실행파일에서 실행 파일로 암호 문자열 자체가 전달된다. GetKey 함수는 암호를 읽어들여 EncryptWithKey, Encrypt, DoWork, 그리고 결국 EncryptData 함수까지 매개변수로 전달한다. 각각의 함수에서 패스워드와 같은 중요데이터가 디버거 등의 툴을 통해 유출될 수 있으므로, 안전하지 못한 설계 방법이다. 그러나 CryptoAPI는 우

측의 안전한 데이터 접근 방식을 사용하는데, Get KeyHandle 함수가 패스워드의 핸들 값을 리턴 하여, 핸들 값을 원래 패스워드가 저장된 위치에서 읽어 들이는 EncryptData 함수까지 계속 전달한다. 중간의 어떤 함수가 유출되더라도, 공격자는 패스워드가 아닌 핸들 값에만 접근이 가능하다[2].

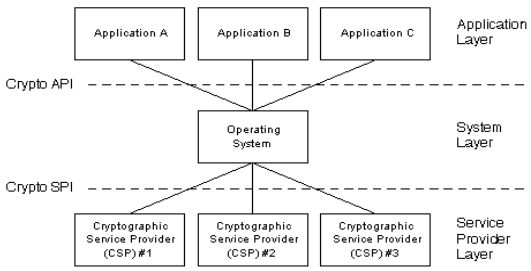



(그림 1) 중요 데이터 접근 방법 비교

## 2.2 CSP(Cryptographic Service Provider)

Microsoft 암호화 시스템은 응용프로그램, 운영체제 시스템, CSP의 세 가지 구성요소로 이루어져 있다. (그림 2)는 세 가지 구성요소가 포함된 암호화 시스템 구조도이다. 응용프로그램은 CryptoAPI로 알려진 함수들의 집합을 통해 운영체제 시스템과 통신하며, 운영체제 시스템은 CryptoSPI(Cryptographic Service Provider Interface)로 알려진 함수들의 집합을 통해 CSP와 통신한다[3].

응용프로그램이 암호화 시스템을 이용할 때 몇 가지 제약 조건이 있는데, 먼저 CSP와 직접 통신할 수 없다는 것이다. 따라서 모든 암호화 함수 호출은 CryptoAPI를 통해 호출되며, 각 CryptoAPI 함수내의 파라미터는 CSP가 실제 암호화 연산을 수행하기 위해 사용하는 운영체제에 있는 값을 가리킨다. 또한, 응용 프로그램은 특정 CSP의 특징을 이용하면 안 된다. 예를 들어, Microsoft RSA Base Provider는 40비트의 비밀키와 512비트의 공개키를 사용한다. 이 때 응용프로그램이 키를 저장하기 위해 요구되는 메모리의 양을 위와 같이 가정하게 될 경우, 사용자가 다른 CSP를 사용하게 될 경우, 에러를 발생시킬 수 있다.




(그림 2) 암호화 시스템 구조

CSP 모듈 자체에 대한 보안성과 호환성을 위해 디자인 규칙을 다음과 같이 정하고 있다.

- 안전하고 이식 가능한 애플리케이션 작성의 편의를 위해 애플리케이션이 암호화 내부에 접근하는 빈도를 제한한다.
- 응용프로그램은 CSP에서 사용되는 키 관련 데이터를 직접 접근할 수 없다. 모든 키 관련 데이터는 CSP 내부에서 생성되고, 불투명한 핸들을 통해 응용프로그램에 전달되어 사용되기 때문에, 응용프로그램이나 그와 관련된 DLL에 키 관련 데이터가 누설될 위험이 없다.
- 응용프로그램은 암호화 연산의 세부사항을 지정할 수 없다. 다만 알고리즘 선택과 같은 광의적사항만을 지정할 수 있다. 즉, 암호화 연산의 실질적 구현은 CSP가 전적으로 책임진다.
- 사용자 인증 데이터는 응용프로그램에서 처리하지 않고, CSP에 의해 처리된다. 이러한 방식은 응용프로그램이 인증방식에 의존적이지 않게 하므로, 새로운 인증 기법(예 : biometric inputs and data keys)을 응용프로그램의 변화 없이 추가할 수 있게 된다.

CSP는 최소한 하나의 동적 연결 라이브러리(DLL)와 서명된 파일로 구성된다. 이 서명된 파일은 Windows 운영체제가 CSP를 인식 하는데 사용된다. CSP 모듈의 무결성을 보증하기 위해 Windows 운영체제는 이 서명을 주기적으로 검증한다.

CSP는 Key BLOB을 통하여 외부로 키를 저장하는 수단을 제공한다. CryptExportKey 함수를 호출하여 CSP안에 있는 키를 외부(Key BLOB)에 저장할 수 있으며, Key BLOB을 다시 CSP 안에 CryptImportKey 함수를 호출하여 저장시킬 수 있다. 이렇게 Key BLOB을 이용하여 키를 다른 시스템에 복사 할 수도 있다. 종류에는 Simple Key BLOB, Public Key BLOB 그리고 Private Key BLOB 등이 있다.

## 3. CSP 취약점 분석

CSP 내부에서 사용되는 암호화 관련 데이터들을 CSP의 취약점을 이용하여 관찰하기 위해 다음과 같이 3가지 방법을 고려해 보았다.

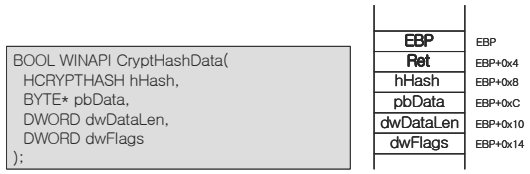
### 3.1 매개변수 관찰 기법

CryptoAPI에서 리턴값은 보통 BOOL형식으로 성공/실패를 나타낸다. 따라서 리턴값을 확인하는 것은 의미가 없으며, 함수에 전달되는 매개변수들을 먼저 확인해 보았다. 매개변수의 정보를 모니터링하려면 함수 호출 규약에 대한 이해가 필요한데, <표 1>은 여러 가지 함수 호출 규약을 정리한 것이다.

<표 1>의 모든 함수 호출 규약에서는 매개변수를 함수에 전달할 때 역순으로 스택에 저장한다. 즉, 마지막 매개변수가 스택의 맨 하단에 저장된다. Windows에서 기본적으로 사용하는 함수 호출 규약은 "__stdcall"이다[7]. 이 호출 규약에서는 함수가 리턴될 때는 호출된 함수에서 스택에 있는 매개변수 등을 제거한 후 리턴한다. (그림 3)은 Windows에서 스택에 저장되는 매개변수를 보여주고 있다.

〈표 1〉 함수 호출 규약

| 호출 규약 | 스택 정리[1] | 특징 |
|---|---|---|
| __cdecl | 호출한 곳 | C언어 표준, 가변 인자 지원 |
| __fastcall | 호출당한 곳 | 파라미터 2개를 ecx, edx 레지스터를 통해서 전달, 빠름 |
| __stdcall | 호출당한 곳 | Windows 표준 |
| __thiscall | 호출당한 곳 | C++ 클래스 멤버 함수, ecx를 통해서 this 포인터를 전달 |




(그림 3) 스택에 저장된 매개변수

CryptHashData 함수에서 맨 뒤에 있는 매개변수가 스택에 가장 먼저 저장된 모습을 볼 수 있다. 이런 방식으로 첫 번째 매개변수 까지 저장한 다음, 함수 종료 후 다음에 실행할 코드의 주소를 저장하고, EBP 레지스터 값을 저장한다.

이와 같이 매개변수를 역순으로 저장하기 때문에 (그림 3)에서와 같이 각 매개변수의 위치를 EBP 값을 기준으로 계산 할 수 있다. 즉, CryptHashData 함수가 실행된 후 EBP + 0x8의 값을 확인하면 hHash 값을 확인할 수 있다.

매개변수에는 암호화 관련 데이터 정보가 어떤 형식으로든 포함되기 때문에 매개변수를 확인하면 해쉬 함수나 암호화 함수에 전달되는 평문과 패스워드 정보, 해쉬 결과 값 등을 확인 가능 했다.

1) "스택 정리"는 함수 수행이 끝나고 스택에 있던 함수 관련 정보를 없애는 것을 의미함.

그러나 이러한 방법으로 Windows에서 사용되는 개인키 또는 비밀키의 확인은 불가능하다. CryptoAPI에 사용된 핸들과 CSP에서 사용하는 핸들 값이 다르기 때문에 CryptoAPI 함수에 전달된 핸들 값으로는 CSP안에 있는 값에 접근할 수 없었다. 이 문제의 해결을 위해 다음과 같은 두 번째 방법을 시도했다.

### 3.2 메모리 역추적 기법

우선 프로세스의 덤프 데이터를 저장한 다음, 찾고자 하는 데이터를 덤프 데이터 파일에서 검색하여 상대적 가상 주소(Relative Virtual Address)를 얻고, 이것을 다시 가상 주소(Virtual Address)로 계산하여 이 위치에 접근하는 명령어의 포인터(Instruction Pointer)를 찾는다. 즉, 이 방법은 메모리에서 특정 데이터의 위치를 찾고, 더 나아가 그 데이터에 접근하는 명령어의 포인터를 획득함으로써 추후 그 데이터 값의 변화를 관찰할 수 있다. 또한 각광받고 있는 코드 혼잡(Code Obfuscation) 기술이 적용된 파일에도 이론상으로는 큰 문제없이 적용될 수 있다. 그러나 분석 대상 프로그램의 구조상 데이터에 접근하는 명령어 코드가 여러 곳일 수 있고, 또한 덤프 데이터 파일에 찾고자하는 데이터가 여러 곳에 있을 수 있기 때문에 여러 번의 수행 착오를 거쳐야 하는 문제점도 있다. 하지만 이 분석 과정을 프로그램으로 자동화 시킬 수도 있을 것이다. 본 논문의 제 4장에서는 단지 디버거 툴로 실험을 수행했다.

### 3.3 메모리 할당/삭제 관찰 기법

메모리 할당 함수를 호출하는 명령어의 포인터를 찾고, 그 위치를 감시하여 새로운 메모리를 할당할 때 마다 할당된 메모리의 주소를 관찰하는 방법이다. 본 논문에서는 이 방법에 대하여 구체적으로 연구해보지는 않았지만 유용한 방법이라 생각된다. 특히, 비밀키 데이터와 같은 정보가 메모리에 생성되고 삭제되는 것을 관찰할 수도 있을 것이다.

## 4. 데이터 노출 실험 결과 및 분석

이번 장에서는 앞장에서 설명한 내용을 토대로 Windows XP Pro 운영체제에서 기본적으로 제공하고 있는 Enhanced CSP를 기준으로 몇 가지 암호화 관련 중요 데이터에 대한 실험 내용을 기술한다.

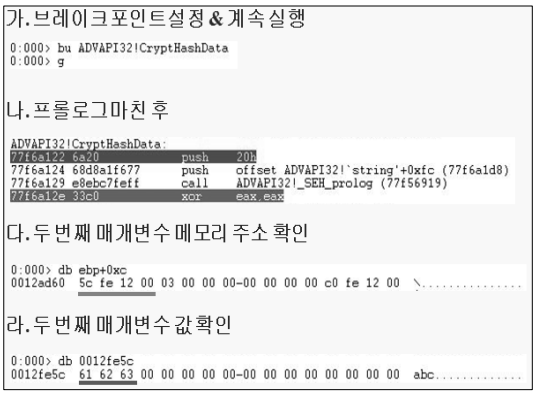
### 4.1 해쉬 함수

CryptoAPI에서는 CryptHashData 함수를 이용하여 해쉬값을 생성한다. 이 때 전달되는 매개 변수를 확인하면 해쉬 함수에 전달되는 평문 데이터를 확인할 수 있다. 특히 패스워드로부터 대칭키를 유도하여 이 키를 통해 암/복호화하는 경우, 대칭키를 유도할 때 CryptDeriveKey 함수를 이용한다. 이 함수의 입력 값은 평문이 아닌 해쉬값이다. 즉, 패스워드의 해쉬값을 얻어 사용하게 된다. 따라서 CryptHashData 함수를 관찰하면 경우에 따라서 패스워드 까지도 확인할 수 있다. CryptHash Data 함수의 정의는 (그림 4)와 같다.

```
BOOL WINAPI CryptHashData(
  HCRYPTHASH hHash,
  BYTE* pbData,
  DWORD dwDataLen,
  DWORD dwFlags
);
```

(그림 4) CryptHashData 함수 정의

CryptHashData 함수로 전달되는 평문 데이터를 확인하려면, 해쉬할 평문 데이터를 담고 있는 두 번째 매개변수 pbData 값을 확인하면 되는데, (그림 5)와 같은 방법으로 CryptHashData 함수의 pbData 값을 확인할 수 있다.




(그림 5) Hash 함수에서 평문 데이터 확인

"abc" 문자열을 해쉬하는 테스트 프로그램에 대한 실험 결과 CryptHashData 함수에 전달된 평문 데이터의 값이 정확히 61 62 63(abc)로 일치함을 알 수 있다.

## 4.2 암호화 함수

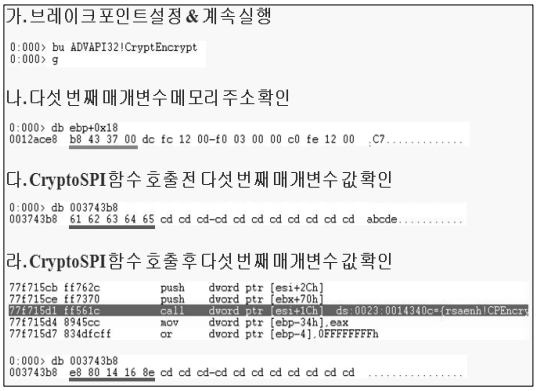
CryptoAPI에서는 CryptEncrypt 함수로 평문을 암호화하고 CryptDecrypt 함수로 암호문을 다시 복호화 한다. 이 때 해쉬 함수와 동일한 방법으로 전달되는 파라미터를 확인하여 평문과 암호문을 확인할 수 있다.

CryptDecrypt 함수는 CryptEncrypt 함수 방법과 동일하기 때문에 과정 설명을 생략하였다. Crypt Encrypt 함수의 정의는 (그림 6)과 같다.

```
BOOL WINAPI CryptEncrypt(
  HCRYPTKEY hKey,
  HCRYPTHASH hHash,
  BOOL Final,
  DWORD dwFlags,
  BYTE* pbData,
  DWORD* pdwDataLen,
  DWORD dwBufLen
);
```

(그림 6) CryptEncrypt 함수 정의

다섯 번째 매개변수 값(pbData)은 입출력 버퍼로 사용되는데 호출 시에 평문 데이터를 저장하고 리턴되었을 때 암호문 데이터가 저장된다. 따라서 CryptoSPI의 CPEncrypt 함수를 호출하기 전과 후의 값을 확인하였다. 다섯 번째 매개변수는 EBP + 0x18 메모리 위치를 확인하여 값을 볼 수 있다. (그림 7)의 방법으로 값을 확인할 수 있다.




(그림 7) CryptEncrypt 함수에서 평문과 암호문 확인

이 과정을 통해 평문 데이터는 61 62 63 64 65 (abcde)이며 해당 평문에 대한 암호문은 e8 80 14 16 8e 임을 확인 가능했다.

## 4.3 비밀키

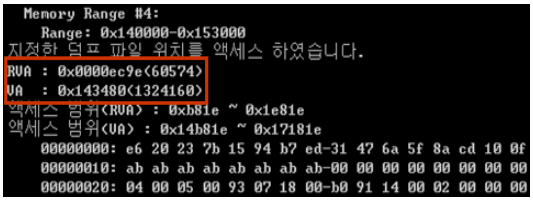
암호화 시스템에서 대칭키 암호화 방법을 사용할 때, 송·수신자가 암호화 및 복호화 과정에 동일한 키를 사용하게 된다. 이 때 사용되는 키를 비밀키(대칭키 혹은 세션키)라고 한다. Windows에서는 파일을 암호화 저장하기 위해 각 파일에 대해 FEK(File Encryption Key)라는 비밀키를 생성하여 사용한다. 비밀키의 노출은 해쉬 함수와 암호화 함수 취약점 분석 시에 사용했던 방법(3.1절)으로는 불가능하다. 그래서 저자가 새롭게 고안한 메모리 역추적 기법(3.2절)을 이용하여 다음의 3단계 과정을 통해 비밀키 값을 확인할 수 있었다.

#### 4.3.1 테스트 프로그램 작성

평문 형태의 비밀키를 CSP로부터 꺼내는(export) 기능을 가진 프로그램 구현한다.

#### 4.3.2 디버거를 사용하여 비밀키에 접근하는 EIP 획득

1) 디버거 실행.
2) CryptExportKey 함수에 중단점을 설정하고, 디버거를 중단점까지 실행.
3) 비밀키를 CSP로부터 꺼내는(export) 지점까지 실행.
4) 덤프 파일 생성.
5) 비밀키를 덤프 파일에서 검색하여 키가 위치한 RVA(Relative Virtual Address)를 획득.
6) RVA값을 이용하여 Virtual Address 계산[1]. (그림 8) 참조.
7) 디버거에서 Virtual Address에 중단점을 설정하고, 디버거를 중단점까지 실행.
8) 중단점에서 EIP값을 얻는다.
9) 바로 이 시점에서 키 값을 확인할 수도 있다.

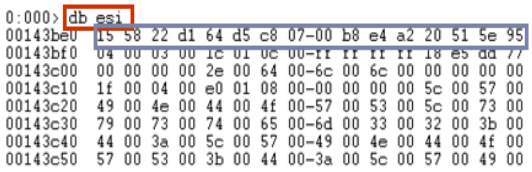



(그림 8) RVA값을 이용하여 Virtual Address 계산

#### 4.3.3 EIP를 이용하여 키 확인

1) 테스트 프로그램 실행.
2) 디버거를 실행하여 실행중인 테스트 프로그램을 선택한 후 attach 한다.
3) 앞에서 얻은 EIP에 중단점을 설정하고, 디버거를 중단점까지 실행.
4) 중단점에서 비밀키 확인.

(그림 9)에서 실험을 통해 노출된 비밀키를 확인할 수 있다.




(그림 9) 노출된 비밀키

### 4.4 개인키

공개키/개인키 쌍은 전자 서명 또는 비밀키와 같이 비교적 적은 양의 데이터를 강도 높은 수준으로 안전하게 암호화하여 송·수신하기위해 사용된다[5]. 공개키는 일반적으로 모든 이들에게 공개되기 때문에 공개키 값을 확인하는 것은 큰 의미를 가지기 어렵다. 따라서 본 논문에서는 개인키 노출 방법에 대해서만 언급한다.

공개키로 암호화된 비밀키를 복호화 할 때 개인키가 사용된다는 점에 착안하여 암호화된 비밀키를 CSP에 넣는 과정에서 비밀키에 접근하는 부분을 찾는 방식으로 키 노출을 확인하였다. 또한, Enhanced CSP는 개인키를 메모리에 암호화하여 저장하므로 단순히 CryptImportKey 함수에 중단점을 설정하고 그 위치에서 메모리 덤프하면 암호화된 개인키만 볼 수 있다. 그렇기 때문에 개인키가 복호화 된 시점에서 덤프해야 한다. 복호화 되는 시점에서 콜 스택(Call Stack)은 (그림 10)과 같았다.

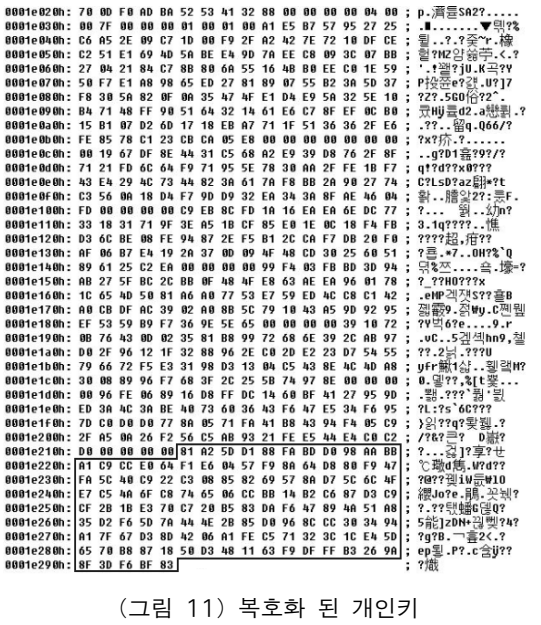
```
ADVAPI32!SystemFunction041
rsaenh!MyRtlDecryptMemory+0x12
rsaenh!RSAPrivateDecrypt+0xce
rsaenh!PKCS2Decrypt+0x5b
rsaenh!RSADecrypt+0x130
rsaenh!CPImportKey+0x6c1
ADVAPI32!CryptImportKey+0xcb
decrypt!DecryptFileA+0x512
decrypt!main+0x1c2
decrypt!mainCRTStartup+0x170
kernel32!BaseProcessStart+0x23
```

(그림 10) 개인키가 복호화 된 시점의 Call Stack

(그림 10)에 나열된 함수들의 기능에 대하여 다음과 같이 분석되었다. CryptImportKey 함수는 키를 CSP 안으로 복사하기 위한 CryptoAPI이고, CPImportKey 함수는 CryptImportKey 함수에 대해 실제 CSP에 구현된 CryptoSPI이다. RSADecrypt 함수는 키를 복호화하기 위한 함수이고, PKCS2D ecrypt 함수는 PKCS 2타입 형식[6]의 키를 복호화 하며, RSAPrivateDecrypt 함수는 개인키의 복호화 기능을 담당 한다. MyRtlDecryptMemory 함수에서 복호화 함수 SystemFunction041를 호출하는데, 함수가 종료된 후에 덤프하면 복호화 된 개인키를 얻을 수 있다. 이렇게 복호화 된 개인키를 덤프하게 되면 개인키에 접근하는 EIP를 찾을 수 있으며, 다음의 방법으로 개인키를 확인했다.

1) 자신의 개인키를 평문 형태의 파일로 저장하고, 자신의 공개키로 암호화된 비밀키를 CSP에 넣는 테스트 프로그램 작성.
2) CryptImportKey 함수에 중단점을 설정하고, 디버거를 중단점까지 실행.
3) 중단점에서 멈추었다면 MyRtlDecryptMemory 함수에 중단점 설정 & 계속 실행.
4) 지정한 중단점에서 디버거가 멈추면 한 스텝씩 코드를 계속 실행하여 SystemFunction041 함수 호출 코드까지 계속 실행.
5) 덤프 파일을 생성.
6) 평문 형태의 개인키가 저장되어있는 파일을 이용, 개인키를 덤프 파일에서 검색하여 키가 위치한 RVA값을 얻음.
7) RVA값을 이용하여 Virtual Address 계산.
8) Virtual Address에 중단점을 설정하고 계속 실행.
9) 중단점에서 EIP값을 얻는다.
10) EIP에 중단점을 설정하고 키 값 확인.

위 실험 결과, 메모리에서 덤프 받은 복호화 된 개인키 데이터는 대체적으로 Private Key BLOB 원 형태를 유지하고 있지만, BLOB 중간 중간에 4바이트 혹은 8바이트의 NULL값이 채워진 부분이 있었다. 특이했던 점은 8번 과정에서 중단점을 설정할 때 Private Key BLOB의 privateExponent 시작 위치가 아닌 NULL값으로 채워진 4바이트 전의 위치에 설정했더니 성공적으로 중단점에서 디버거가 멈추었으며, EIP값을 확인 할 수 있었다. (그림 11)은 메모리에서 덤프 받은 복호화 된 개인키 부분이다.



(그림 11) 복호화 된 개인키

## 5. 본 논문의 제약 조건

제 4장의 실험을 통해 CSP 모듈에서 평문, 암호문, 비밀키 그리고 개인키를 추출할 수 있는 취약점에 대해 분석했다. 그러나 본 논문에서 언급된 CSP 모듈의 취약점을 일반적으로 적용시키기는 어렵다.

우선 본 연구 내용은 FAT 파일 시스템에서만 적용이 가능하다. FAT 파일 시스템에는 각 파일에

대한 접근 권한 속성이 없지만 NTFS에는 접근 권한 속성이 있어서 다른 계정의 프로세스에 접근이 불가능한 것이 그 이유이다. 현재 Windows 운영체제를 사용하는 대부분의 시스템에서 보안성이 향상된 NTFS를 사용한다는 점을 고려해본다면 본 연구 내용이 적용될 수 있는 시스템은 한정되어진다. 그럼에도, FAT 파일 시스템이 NTFS에 비해 속도적인 측면에서 우위를 보이기 때문에 파일 입출력 속도가 중요한 시스템 또는 USB 드라이브나 플래시 메모리에는 여전히 사용되고 있으며 본 연구 결과를 적용할 수 있을 것이다.

또한 본 논문에서 제시한 방법은 해당 시스템에 대한 접근권한을 충분히 가지고 있어서 디버깅 도구를 해당 시스템에서 직접 실행할 수 있고, 해당 프로세스에 접근이 가능할 경우에만 가능하다.

## 6. 결 론

본 논문에서는 Windows에서 사용되는 암호화 모듈 시스템에 대하여 살펴보고, CSP의 취약점을 이용하여 암호화 관련 중요 데이터들이 노출될 수 있는 문제점까지 분석하였다. 이는 비밀키나 개인키가 다른 사용자에게 노출될 수도 있다는 점에서 심각한 문제점을 가지고 있다. 물론, 앞에서 언급한 제약 조건으로 인해 NTFS 파일시스템 하에서는 다른 계정의 키 확인이 불가능했지만, 향후 NTFS를 우회 할 수 있는 방법이 발견된다면 다른 계정의 키를 엿볼 수도 있을 것이다. 이런 취약점은 바이러스, 웜 등과 같은 악성 코드에 악용될 수 있으며 사용자가 인지하지 못하는 사이에 암호화된 통신 내용을 누군가 엿볼 수도 있다. 최근에는 윈도우즈 암호화 모듈의 취약점을 이용하여 사용자 시스템의 데이터를 암호화하여 인질로 잡는 등의 암호학적 바이러스에 관한 연구도 시도되고 있다[4].

마지막으로, 향후 연구 과제로는 NTFS 파일 기반 시스템에서 키 값이 노출이 될 수 있는지에 대한 분석 연구와 이러한 암호화 모듈의 취약점을 보완할 수 있는 기술 개발이 필요하다.

## 참 고 문 헌

[1] 이호동, “Windows 시스템 실행파일의 구조와 원리”, 한빛미디어, 2005.

[2] Michael Howard and David LeBlanc, “Writing secure code second Edition”, Microsoft Press, September 2002.

[3] “Application Programmer’s Guide : Microsoft CryptoAPI”, Microsoft Corporation, January 1996.

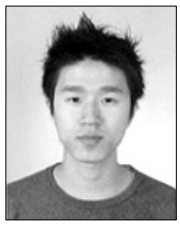
[4] Adam L. Young, “Cryptoviral extortion using Microsoft’s Crypto API”, International Journal of Information Security, March 2006.

[5] R. Rivest, A. Shamir, L. Adleman, “A method for Obtaining Digital Signatures and Public-key Cryptosystems, In CACM, Vol. 21, No. 2, pp. 120-126, 1978.

[6] “PKCS #1 : RSA Encryption Standard”, RSA Laboratories Technical Note, version 1.5, November 1993.

[7] http://msdn.microsoft.com/.

[8] http://www.technologynewsdaily.com/.



**박 진 호**

2006년 용인대학교 컴퓨터 정보처리학과 졸업

2006년~현재 한양대학교 정보통신공학과 석사과정

관심분야 : IPv6, 시스템 보안

**조 재 익**

2005년 한양대학교 정보경영공학과 졸업
2005년~현재 한양대학교 정보통신공학과 석사과정
관심분야 : MANET, 네트워크 보안

**임 을 규**

2002년 University of Southern California 컴퓨터과학 박사
2000년~2002년 WiseNut Inc. Sr. SW Engineer
2002년~2005년 국가보안기술연구소 선임연구원
2005년~2007년 한양대학교 정보통신대학 컴퓨터전공 전임강사
2007년~현재 한양대학교 정보통신대학 컴퓨터전공 조교수
관심분야 : 유무선 네트워크 보안, 정보보안