악성코드의 암호화 및 해독 기법

아주대학교 이성욱* · 홍만표**

1. 서 론

스캐닝(scanning)을 통한 시그니쳐 인지(signature recognition)는 현재까지도 가장 보편적으로 사용되고 있는 악성 코드 감지 방식으로, 특정 악성 코드에만 존재하는 특별한 문자열을 데이터베이스화하고 이 문자열의 존재를 탐색함으로써 해당 스크립트의 악성 여부를 진단한다. 시그니쳐란 특정 악성코드에만 존재하며 다른 프로그램에는 존재하지 않는 짧은 문자열로서, 다른 선의의(legitimate) 프로그램들과 악성 코드를 구분하고 그것이 어떤 악성 코드인지를 식별하는데 이용된다[1,2].

적절하게 추출된 시그니쳐를 이용한 스캐닝은 진단 속도가 빠르고 상대적으로 정확한 감지를 수행하며 악성 코드의 종류를 명확하게 구분할 수 있다. 그러나, 각각의 악성 코드에 대응되는 시그니쳐의 검색만으로는 알려지지 않은 새로운 악성 코드에 대한 감지가 어려운 것 또한 사실이다. 따라서, 알려진 악성코드에 공통적으로 나타나거나, 시스템 위해(harm) 행위에 사용될 수 있는 코드 조각들을 검색하여 악성 여부를 판단하는 정적 휴리스틱 분석(static heuristic analysis)이 보조적으로 사용되며, 넓은 의미에서의 시그니쳐는 이에 사용되는 코드 패턴까지를 포함한 개념으로 볼 수 있다[3].

악성 코드의 암호화는 이러한 감지 방식에 대응하기 위해 출현한 것이다. 일반적인 의미에서 암호화(encryption)는 그 의미가 드러나지 않도록 메시지를 인코딩(encoding)하는 절차 또는 기법을 의미한다[4]. 그러나, 컴퓨터 바이러스 또는 악성코드에 있어서의 암호화는 악성코드를 변조(scrambling)함에 의해 바이러스 탐색기(scanner)가 해당 악성코드의 시그니쳐를 찾지 못하도록 숨기고 다형성(polymorphism)을 확보하는 기법을 지칭한다[5].

암호화된 악성코드에서는 악성 코드의 전체 또는 일부가 암호화된 문자열 형태로 존재하며, 실행 시에 해독 루틴이 먼저 수행된다. 해독 루틴은 암호화된 악성 코드를 해독하고 해독된 악성코드 쪽으로 제어를 넘김으로써 의도하는 악성 코드가 실행되도록 한다. 따라서, 악성 코드가 다른 시스템 또는 다른 파일에 자기 복제(self-replication)를 시도할 때, 새로운 키 값을 사용하여 인코딩(encoding)하는 것만으로 전혀 다른 형태를 보이므로 단순한 스캐닝만으로는 감지할 수 없게 된다.

본 고에서는 현재까지 알려진 악성 코드의 암호화 기법들을 살펴보고, 이에 대응하기 위해 제안된 해독 기법들을 소개한다.

2. 암호화 기법

2.1 암호화 바이러스와 다형성 바이러스

암호화된 악성코드는 일반적으로 해독루틴과 키(key) 값, 그리고 암호화된 악성코드로 구성되고, 실행 시에 해독 루틴이 먼저 수행된다. 따라서, 해독 루틴은 주어진 키 값을 이용하여 암호화된 악성 코드를 해독하고, 해독된 악성코드 쪽으로 제어를 넘김으로써 악성 코드가 실행되도록 한다. 해독 루틴이 암호화된 악성 코드 본체를 해독하는 과정을 도시하면 그림 1과 같다[5].

즉, 이러한 암호화 바이러스는 자기 복제 시마다 매번 다른 키 값을 사용함으로써, 외형상 전혀 다른 악성 코드를 생성할 수 있게된다. 그러나, 이러한 경우 키 값은 매번 변하지만 동일한 암호 해독 루틴이 이용되므로, 안티바이러스가 해당 악성 코드의 해독

* 학생회원
** 종신회원

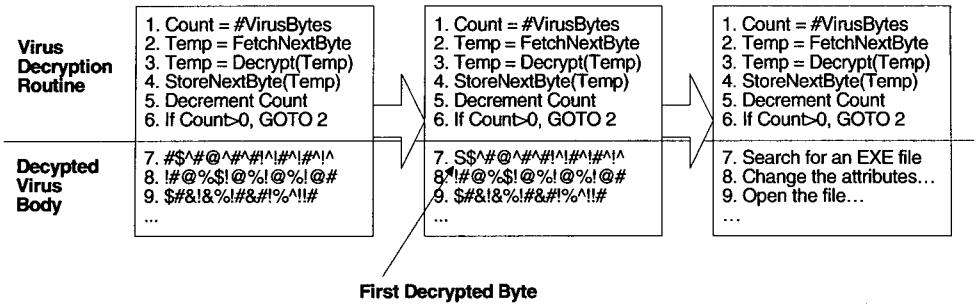


그림 1 암호화된 악성 코드의 해독 과정

루틴에 대한 약간의 정보만 가지고 있으면, 쉽게 암호화된 문자열을 해독할 수 있었다. 이에 대응하기 위해 출현한 것이 다형성 바이러스이다.

다형성 바이러스는 암호화 바이러스와 마찬가지로 암호화된 바이러스 본체와 해독 루틴을 가지고 있으며, 실행 시 해독 루틴이 제어를 넘겨받아 바이러스 본체에 대한 해독 작업을 수행한다는 점도 동일하다. 그러나, 다형성 바이러스는 뮤테이션 엔진(mutation engine)을 포함하고 있어, 자기 복제를 수행할 때마다 임의의 새로운 해독 루틴을 생성하게 된다. 뮤테이션 엔진과 바이러스 본체는 암호화된 형태로 존재하며, 사용자가 감염된 프로그램을 실행할 때, 해독 루틴이 제어를 넘겨받고 암호화된 바이러스 본체와 뮤테이션 엔진을 해독한다. 해독 루틴이 바이러스에 제어를 넘겨주면 바이러스는 자기 자신과 뮤테이션 엔진의 복사본을 메모리에 생성하고, 뮤테이션 엔진을 호출한다. 뮤테이션 엔진이 이전의 해독 루틴과는 다른 형태의 해독 루틴을 생성하면, 바이러스는 바이러스 본체와 뮤테이션 엔진을 암호화하여 다른 프로그램에 이를 덧붙임으로써 감염을 시도한다. 결과적으로 암호화된 바이러스 본체 뿐 아니라 바이러스 해독 루틴까지도 매 번의 감염마다 다른 형태를 보이게 되며, 시그니쳐, 키 값, 암호 해독 루틴 등 외형상 아무런 공통점이 없어 보이는 새로운 복제본이 생성되므로, 안티바이러스가 특정한 해독 루틴을 식별하는 것이 매우 어렵게 된다[5].

2.2 악성 스크립트의 암호화 기법

현재 보편적인 스크립트 악성코드의 암호화 패턴은 크게 두 가지로 나타난다. 첫째는 이진 형태의 악성 코드와 같이 악성코드 전체가 하나의 문자열로 암호화되어 있는 경우로, 그림 2와 같은 형태를 가지고 있다. 이것은 VBS/VBSWG.T로 명명된 악성 스크립트로서, "execute" 문장의 "snphhuatvsbkwuj" 함수에 인자(parameter)로 주어진 것이 암호화된 악성 코드이다. 마이크로소프트 비주얼 베이직 스크립트(이하 비주얼 베이직 스크립트라 칭함)의 "execute" 문장은 인자로 주어진 문자열을 프로그램 코드로 보고 실행하므로, 먼저 해독함수를 호출하게 된다[6]. 따라서, 전체 코드가 완전히 해독된 후에 실행을 개시하게 된다.

```
Execute(snphhuatvsbkwuj("&Wcr/Wcrvf/H,Vnsl/@w ..... doe!gtobuhno"))

Function snphhuatvsbkwuj(zwbyjntbpmhqqgh)
    For vvpzxfszgnczrao = 1 To Len(zwbyjntbpmhqqgh)
        ccuhbhjhyzkheeq = Mid(zwbyjntbpmhqqgh, vvpzxfszgnczrao, 1)
        If Asc(ccuhbhjhyzkheeq) = 7 Then
            ccuhbhjhyzkheeq =Chr(34)
        End If
        If Asc(ccuhbhjhyzkheeq) <> 35 and Asc(ccuhbhjhyzkheeq)
        <> 34 Then
            If Asc(ccuhbhjhyzkheeq) Mod 2 = 0 Then
                ccuhbhjhyzkheeq = Chr(Asc(ccuhbhjhyzkheeq) + 1)
            Else
                ccuhbhjhyzkheeq = Chr(Asc(ccuhbhjhyzkheeq) - 1)
            End If
        End If
        snphhuatvsbkwuj = snphhuatvsbkwuj & ccuhbhjhyzkheeq
    Next
End Function
```

그림 2 하나의 문자열로 암호화된 악성 스크립트의 예(VBS/VBSWG.T)

두 번째 패턴은 그림 3과 같이 프로그램에서 사용하는 일부 문자열을 암호화한 경우이다. 이러한 유형의 악성 스크립트는 하나 또는 그 이상의 해독 함수

```
...
Set Roy =
Maggie.CreateTextFile(Maggie.BuildPath(Maggie.GetSpecialFolder(2),V("7594E44554D405E2458545")),
                                    True)
Roy.WriteLine(V("E402") & Maggie.BuildPath(Maggie.GetSpecialFolder(2),V("7594E44554D405E245D405")))
Roy.WriteLine("E 0100" & H("4D5AE701630001006002406FFFF5F0C"))
Roy.WriteLine("E 0110" & H("000200000001F0FF570000000132504B"))
...
...
Function V(Van)
    For Kirk = 1 To Len(Van) Step 2
        V = V & Chr("&h" & Mid(Van,Kirk + 1,1) & Mid(Van,Kirk,1))
    Next
End Function

Function H(Houten)
    For Luann = 1 To Len(Houten) Step 2
        H = H & " " & Mid(Houten,Luann,2)
    Next
End Function
```

그림 3 일부 문자열들이 암호화된 악성 스크립트의 예(VBS/TripleSix)

를 가지며, 함수의 인자 또는 대입문의 우변값(r-value) 등에 사용되는 임의의 문자열이 암호화된 형태를 보인다. 따라서, 첫 번째 경우와 달리 실행이 진행되면서 필요한 시점에 필요한 문자열이 해독되는 방식으로 동작하게 된다.

3. 암호화된 악성 코드의 해독

3.1 기존의 해독 기법

암호화 악성 코드를 해독하기 위해 보편적으로 사용되는 방법으로는 엑스레잉(X-raying)과 에뮬레이션(emulation) 기법이 알려져 있다[7].

엑스레잉 기법은 해당 악성 코드에서 찾아야 하는 시그니쳐와 악성 코드가 사용하는 해독 알고리즘에 대한 사전 지식을 이용하여 그 범위를 좁힌 후, 남겨진 모든 경우를 시도(brute-force decryption)하면서 시그너쳐의 존재 여부를 판단하는 방법을 말한다[8]. 그러나, 이러한 방법은 찾고자 하는 악성코드의 암호화 기법과 특성이 면밀히 분석되어 충분한 사전 지식이 얻어진 후에 실행 가능하므로, 알려지지 않은 새로운 악성코드에는 적용하기 어렵다는 단점을 가지게 된다.

에뮬레이션 기법은 그 명칭이 의미하는 바와 같이, 악성 코드를 가상 기계에서 에뮬레이션 하여 해독된 코드를 얻는 방법을 말한다. 이 때, 모든 해독이 완료된 코드를 얻어내려 한다면 가상기계의 각 메모리 유닛을 감시하여 코드 부분의 메모리에 더 이상 변화가 일어나지 않는 시점까지 실행을 계속하여야 하며, 시그니쳐 기반의 감지 방법과 결합하여 사용하는 경우에는 시그니쳐가 위치한 메모리 값들의 해독이 완료되면 그 즉시 에뮬레이션을 중단하고 시그니쳐 비교를 실시하게 된다. 이 과정을 도시하면 그림 4와 같다[5]. 그림에서 숙주 프로그램(host program)은 바이러스가 감염된 일반 프로그램을 의미하며, 해독 루프(decryption loop)는 암호 해독 루틴을 지칭한다. 가상 기계 상에서 바이러스에 감염된 프로그램을 실행시키면 가장 먼저 해독 루프로 제어가 넘겨지며, 해독 루프가 실행되면서 암호화되어 있던 바이러스 몸체와 뮤테이션 엔진이 해독된다. 따라서, 실행 중에 발생하는 메모리 값의 변화를 표시하며 주어진 프로그램을 실행시키면서 더 이상의 변화가 없거나, 시그니쳐가 대조되어야할 위치의 메모리 값이 모두 표시되어있을 때 에뮬레이션을 중단할 수 있다.

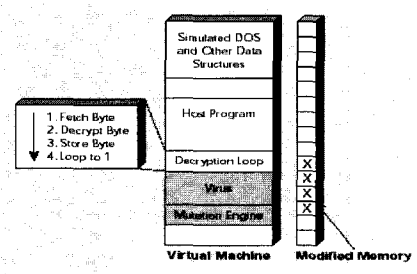


그림 4 에뮬레이션을 통한 암호화 악성 코드의 해독 과정

이 같은 에뮬레이션 기법은 주어진 악성 코드에 대한 상세한 사전 지식 없이 암호화된 악성 코드를 해독할 수 있으나, 가상 기계의 구현에 어려움이 존재하는 것이 사실이다. 즉, 에뮬레이션을 위해서는 대상 코드가 실행될 수 있는 모든 환경을 가상적으로 만들어 주어야 하는데, 마이크로소프트 윈도우즈와 같은 최근의 운영체제에서, 해당 프로그램이 사용하는 각종 객체와 제반 환경들을 모두 에뮬레이션 하는 것이 현실적으로 매우 어려우며 부하 또한 크다고 알려져 있다[9]. 또한, 최근 스크립트 악성 코드에서 흔히 나타나는 부분 문자열 암호화에서는, 단순한 에뮬레이션으로 모든 암호화된 문자열을 해독하려면 프로그램이 완전히 종료될 때까지 에뮬레이션을 수행하여야 하며, 한번의 에뮬레이션으로 프로그램의 모든 제어 흐름을 추적할 수 없다는 에뮬레이션의 근본적 단점으로 인해 여전히 암호화되어 있는 문자열들이 남아 있는 경우가 발생하게 된다.

3.2 알려지지 않은 악성 코드의 감지와 암호화된 악성 코드의 해독

이미 알려져 있는 악성 코드의 경우에는 해당 악성 코드의 특성과 행위가 안티바이러스 제작자에 의해 사전에 분석되므로, 암호화 기법의 패턴과 해독 방법을 정의하고 기존의 해독 방법들을 병용함으로써 적절한 시간 내에 해독작업을 완료할 수 있다. 그러나, 알려지지 않은 악성 코드는 암호화 기법 또한 알려지지 않은 상태이므로, 새로운 암호화 기법의 출현에 영향을 적게 받으면서도 적정 시간 내에 해독 작업을 완료할 수 있는 범용의(generic) 해독 기법이 필요하게 된다.

이것은 현재 알려지지 않은 악성 코드의 감지를 위해 널리 사용되고 있는 기법들이 주어진 프로그램을 실행하기 전에 감지를 완료하는 형태임에 기인한다. 즉, 암호화된 악성 코드의 해독 과정은 악성 코드의 감지를 위한 일종의 전처리 작업이므로, 전체 악성 코드 감지 과정에 부담을 주지 않으면서 해당 감지 기법이 처리하기 용이한 형태로 암호화된 악성 코드를 변환하여야 한다.

암호 해독 기법과 악성 코드 감지 기법과의 관계를 살펴보려면, 기존의 악성 코드 감지 기법들에 대한 상세한 언급이 필요하다. 이를 정리하면 그림 5와 같다.

감지 시점에 의한 분류는 실행 전에 해당 코드를 분석하여 악성 여부를 판단하는 직접적 방식(direct method)과, 실행 중 또는 후에 나타나는 악성 행위 및 결과를 관측하여 판단하는 간접적 방식(indirect method)으로 악성 코드 감지 방식을 크게 구분한 것이다[7]. 이와는 다른 관점에서의 전통적인 분류는 악성 여부를 판단하는 근거에 의한 것으로, 코드의 스캐닝을 통해 특정 패턴을 검색하는 스캐너, 에뮬레이션 또는 실제 실행을 통해 대상 코드의 행위 패턴을 감시하는 행위 감시기, 그리고 화일의 변형을 검사하는 무결성 검사기로 악성 코드 감지 기법을 분류하였다[10].

시그너쳐 탐색은 앞서 밝힌 바와 같이 진단 속도가 빠르고 악성 코드의 종류를 명확하게 구분할 수 있다는 장점을 가지고 있다. 그러나, 알려지지 않은 악성 코드에 대해서는 전혀 대응할 수 없으므로, 안티바이러스 업체에서 해당 악성코드의 시그너쳐와 치료방법을 포함하는 새로운 악성 코드 데이터베이스를 배포하기 전까지 많은 사용자들이 악성 코드에 그대로 노출될 수밖에 없다. 따라서, 최근에는 휴리스틱 분석, 행위 차단 기법 등을 이용하여 알려지지 않은 악성 코드를 감지하기 위한 많은 연구가 이루어지고 있다.

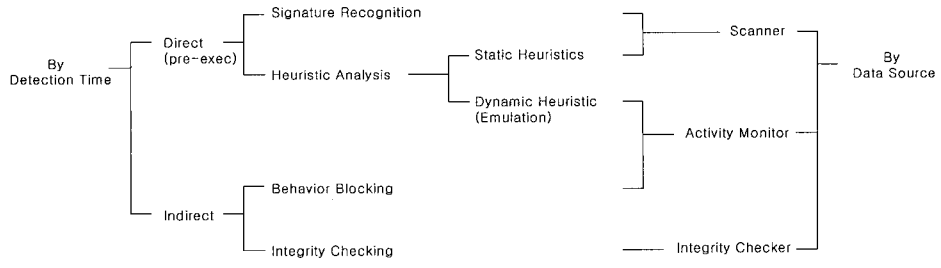


그림 5 악성 코드 감지 기법의 분류

휴리스틱 분석은 기본적으로 새로운 악성 코드의 출현은 빈번하게 이루어지나, 새로운 악성 행위 기법의 출현은 매우 드물게 이루어진다는 데에 착안한 것이다[2,7,8,11]. 일반적인 프로그램에 있어서, 특정 기능을 수행하기 위한 새로운 기법의 개발은 몇몇 선도적인 프로그래머 또는 학자들에 의해 이루어지며, 대부분의 프로그래머들은 이렇게 알려진 기법을 이용하여 프로그램들을 작성하게 된다. 악성 코드 또한 프로그램이므로 선도적인 역할을 수행하는 일부 악성 코드 제작자들에 의해 새로운 악성 행위의 기법이 공개되고, 그 뒤로 이를 이용한 다수의 악성 코드들이 출현하게 된다. 따라서, 이미 알려진 악성 행위의 기법에 대한 휴리스틱을 이용하여 주어진 코드를 분석함으로써, 이미 알려진 악성 행위를 포함하고 있는 많은 새로운 악성 코드를 감지할 수 있다.

휴리스틱 분석 기법은 악성 코드 내부에 존재하는 코드 형태에 대한 정적 휴리스틱을 이용하는 것과, 에뮬레이션을 통해 얻어지는 실행 중 발생 행위에 대한 동적 휴리스틱을 이용하는 방법으로 나누어진다. 정적 휴리스틱 분석은 악성 행위에 자주 이용되는 코드 조각들을 데이터베이스화 하여두고 대상 코드를 스캔하여 그 존재 여부나 출현 빈도를 탐색하여 악성 코드를 감지하는 방식이다. 이 방식은 속도가 비교적 빠르고 높은 감지율을 보이나, 긍정 오류가 다소 높다는 단점을 가지고 있다[3]. 동적 휴리스틱 분석은 가상 기계를 구현한 에뮬레이터 상에서 해당 코드를 수행하면서 프로그램 수행 중에 발생하는 시스템 호출과 시스템 자원(resource)들에 발생하는 변화를 감시함으로써 악성 행위를 감지하는 방식이다[2]. 그러나, 이를 위해서는 완전한 가상 기계를 구현하여야 하며, 한번의 에뮬레이션만으로는 모든 프로그램 흐름을 추적할 수 없다는 단점을 가지고 있다. 따라서, 문자열 검색을 통해 위험한 코드를 탐색하고 악성 코드에 존재하는 로직 트릭(logic trick)에 대비하기 위한 프로그램 흐름 분석을 수행하며, 본격적인 분석은 에뮬레이션을 통해 이루어지는 하이브리드 휴리스틱 분석(hybrid heuristic analysis)이 이상적인 대안으로 제시되고 있다.

행위 차단(behavior blocking) 기법은 실제로 대상 시스템에서 코드를 실행시킨다는 점 외에는 동적 휴리스틱을 이용한 감지 방법과 유사한 것으로 생각될 수 있다[12]. 그러나, 에뮬레이션은 아무런 부작용(side effect) 없이 긴 시간 동안의 행위 감시를 통해 대상 코드의 악성 여부를 판별할 수 있는데 반해, 악성 코드를 실제 시스템에서 실행하고 동일한 감시를 수행한다면 악성 행위가 실제로 일어나게 되므로, 디스크 포맷이나 시스템 파일 변형 등과 같이 악성 코드가 실행할 가능성이 높은 각각의 행위가 감지되면 이를 즉각 차단하여야 한다. 따라서, 실질적으로 에뮬레이션에서와 같은 긴 시간 동안의 행위 패턴 감시가 어렵고, 각각의 위험 행위가 발생할 때마다 경고가 주어지므로 매우 높은 긍정 오류(false positive)를 보이게 된다.

무결성 검사(integrity checking)는 로컬 디스크에 존재하는 파일들 전체 또는 일부에 대하여 파일 정보 및 체크섬, 또는 해쉬 값을 기록하여 두었다가 일정 시간이 지난 후 파일들이 변형되었는가를 검사하는 간접적인 악성 코드 대응 방식이다[13]. 이 방식은 지정된 파일의 변형만을 감지하므로 적법한 내용의 변화가 예상되는 파일에 사용할 경우 매우 높은 긍정 오류를 발생시킨다는 단점을 가지고 있다. 따라서, 서버 상에서 악성 코드 또는 시스템 침입(intrusion)에 의한 변형을 감지할 목적으로 일부 시스템 파일들에 대해서 적용하는 것이 일반적이다.

이상과 같은 악성 코드 감지 기법들 중, 현재 비교적 널리 활용되고 있는 것은 휴리스틱 분석 기법의 부류에 속하는 것들이다. 이것은 휴리스틱 기법들이 해당 프로그램의 실행 전에 악성 여부를 판별하므로, 시스템에 악성 코드의 실행으로 인한 부작용이 발생하지 않으며, 감지 후 별도의 복구 작업이 필요 없다는 장점이 크게 작용한 것으로 보인다. 그러나, 이러한 휴리스틱 분석이 원활히 이루어지기 위해서는 암호화되어 있는 악성 코드에 대한 해독 작업이 반드시 선행되어야 한다.

즉, 에뮬레이션에 의한 동적 휴리스틱 분석은 한 번의 실행으로 모든 분석이 완료되기 어렵고 많은 시간을 필요로 한다는 점을 고려할 때 단독으로 이용되기 어려우므로, 현실적으로 유용하게 사용할 수 있는 기법은 정적 또는 하이브리드 형태의 휴리스틱 분석으로 볼 수 있다. 이들 분석 기법은 주어진 코드를 스캔하여 악성 행위를 유발할 수 있는 코드 패턴을 탐색하거나, 프로그램의 흐름을 분석하는 작업을 수행하며, 이러한 작업이 원활하게 이루어지기 위해서는 암호화되지 않은 상태의 악성 코드가 주어져야 한다. 또한, 이를 위해 사용되는 해독 알고리즘은 새로운 암호화 패턴에 유연하게 대응할 수 있는 형태를 가지

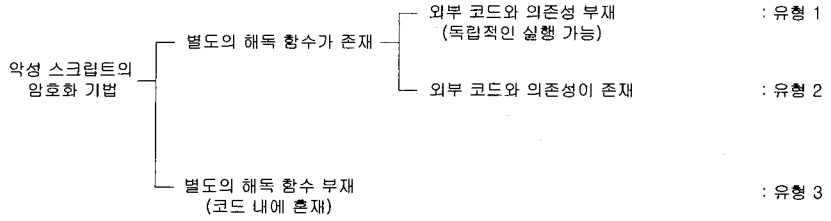


그림 6 악성 스크립트의 암호화 기법

며, 이어지는 분석 과정에 부담을 주지 않도록 짧은 시간 내에 해독 작업을 완료하여야 한다.

3.3 알려지지 않은 암호화된 악성 스크립트의 해독

악성 스크립트에서의 암호화는 넓은 의미에서 이진 파일 형태의 악성 코드와 유사한 형태이나, 그 상세에 있어 다소 다른 형태를 보이고 있다.

첫째, 대부분 해독 루틴이 분리된 함수의 형태로 존재한다. 이것은 주어진 악성 코드를 상세하게 분석하지 않고서도 단순한 스캔만으로 해독 루틴의 후보들을 추출할 수 있도록 한다. 즉, 이론적으로 악성 스크립트의 암호화 기법은 그림 6과 같이 분류될 수 있다.

즉, 악성 스크립트의 암호화 기법은 해독 함수 존재 여부에 따라 크게 두 부류로 나뉠 수 있고, 해독 함수를 가지는 형태는 그 함수가 외부 코드와 의존성을 가지는 경우와 그렇지 않은 경우로 다시 구분될 수 있다. 그러나, 2장에서 밝힌 바와 같이 현존하는 대부분의 악성 스크립트 암호화 기법은 유형1에 속하므로, 이러한 특징을 해독 시에 활용할 수 있다.

둘째, 부분 문자열 암호화의 형태가 존재한다. 일반적인 이진 악성 코드는 해독 루틴이 한번 수행되고 나면 모든 암호화된 문자열이 해독된다. 그러나, 부분 문자열 암호화의 경우에는 해당 부분의 코드가 실행되기 직전에 암호화된 문자열이 해독되는 형태를 취하고 있다. 따라서, 이러한 형태로 암호화된 스크립트를 단순한 에뮬레이션으로 해독하려면 모든 문장이 수행될 때까지 에뮬레이션을 시행하여야 한다.

궁극적으로 볼 때, 암호화된 악성 스크립트의 해독 과정은 어느 정도의 에뮬레이션을 필요로 한다. 그러나, 단순한 전체 에뮬레이션에는 많은 문제가 있으며, 이러한 문제를 회피하기 위해서는 해당 스크립트의 암호를 해독하는데 필요한 부분만을 발췌하여 실행하는 방법이 필요하게 된다. 특히, 알려지지 않은 새로운 스크립트를 대상으로 한다면 다음과 같은 두 가지 문제에 부딪히게 된다.

첫째, "대상 스크립트가 암호화되어 있는가"를 판단하는 문제이다. 이미 안티바이러스 개발자들에 의해 분석된 악성 코드는 암호화 여부와 암호 해독 방법이 알려져 있다. 그러나, 새롭게 등장한 악성 코드의 경우에는 아무런 사전 지식 없이 대상 스크립트의 분석만으로 암호화 여부를 판단하여야 한다.

둘째, "대상 스크립트의 암호 해독 루틴이 어떤 것인가"를 탐색하는 문제이다. 암호화된 스크립트에는 해독 함수 외에도 많은 함수들이 정의되어 있을 수 있다. 따라서, 새로운 암호화 기법의 출현에 영향 받지 않도록 특정 코드 패턴에 의존하지 않고 해독 함수의 집합을 찾아내야 한다.

이 같은 문제는 앞서 밝힌 스크립트 암호화 기법의 특징을 고려하고, 현실적인 문제가 되는 유형 1의 암호화로 범위를 좁혀 생각해 보면 그 해답을 얻을 수 있다. 즉, 대상 스크립트가 암호화되었는지, 또는 암호 해독 루틴이 어떤 것인지 분석하기보다는 해당 스크립트에서 상수화 할 수 있는 모든 값을 상수로 대치함으로써 자연적인 암호 해독을 유도하는 것이다[14].

유형 1의 암호화 기법에서 사용하는 해독 함수는 그 정의에 따라 블랙 박스(black-box)와 같은 형태를 가지게 되며, 단지 호출 시에 자신에게 주어진 상수 인자에 따라 계산된 값을 리턴하게 된다. 따라서, 외부와 아무런 자료 의존성이 없는 함수의 실행 결과 값을 대치하는 것만으로 암호화된 내용의 해독이 가능하다. 즉, 이러한 함수들에 대한 호출식(call expression)과 함수 정의만이 기술된 임시 스크립트를 생성하고 이것을 실행 또는 에뮬레이션하여 얻어진 값을 원래의 호출식이 있던 자리에 대치하여 해독된 스크립트를 얻을 수 있게 된다.

4. 결 론

최근의 많은 악성 코드들은 안티바이러스의 탐지를 회피하기 위해 암호화된 형태로 존재하며, 단순한 인코딩의 수준을 넘어 다양한 형태의 암호화 기법들을 채용하고 있다. 이미 알려진 악성 코드의 경우에는 인력에 의한 분석에서 얻어진 결과를 충분히 활용하여 기존 기법들을 결합함으로써 해독 작업을 수행하고 있다. 그러나, 최근 알려지지 않은 새로운 악성 코드를 감지하기 위한 많은 연구가 이루어지고 있음을 생각해 볼 때, 이러한 연구가 실효를 거두기 위해서는 알려지지 않은 암호화된 악성 코드를 적정 시간 내에 분석 가능한 형태로 변환하는 현실적인 범용 암호 해독 기법에 관한 연구가 지속적으로 이루어져야 할 것이다.

참고문헌

[1] Sandeep Kumar, Eugene H.Spafford, "A Generic Virus Scanner in C++," Purdue University Technical Report CSD-TR-92-062, 1992. 9.

[2] Baudouin Le Charlier, Morton Swimmer, Abdelaziz Mounji, "Dynamic detection and classification of computer viruses using general behaviour patterns," Fifth International Virus Bulletin Conference, Boston, September 20-22, 1995.

[3] Symantec, "Understanding heuristics-symantec's bloodhound technology", Symantec White Paper, 1998.

[4] Charles P. Pfleeger, Security in Computing, pp.22, prentice hall, 1997.

[5] Symantec, "Understanding and Managing Polymorphic viruses," Symantec White Paper, 1999.

[6] Microsoft, VBScript User's Guide, http://msdn.microsoft.com/library/default.asp ?url=/library/en-us/script56/html/vbstutor.asp, Microsoft, 2001.

[7] Francisco Fernandez, "Heuristic Engines," Virus Bulletin Conference, 2001. 9.

[8] Igor Muttik, "Stripping down an AV Engine," Virus Bulletin Conference, 2000. 9.

[9] Gabor Szappanos, "VBA Emulator Engine Design," Virus Bulletin Conference, 2001. 9.

[10] Eugene H. Spafford, "Computer Viruses as Artificial Life," Journal of Artificial Life, MIT Press, 1994.

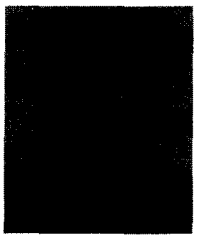
[11] Alex Shipp, "Heuristic Detection of Viruses within Email," virus bulletin conference, 2001. 9.

[12] Tim Hollebeek and Dur Berrier, "Interception, Wrapping and Analysis Framework for Win32 Scripts," DISCEX2(DARPA Information Survivability Confrence and Exposition), 2001. 6.

[13] Gene H. Kim, Eugene H. Spafford, "The Design and Implementation of Tripwire: A File System Integrity Checker," ACM Conference on Computer and Communications Security, 1994.

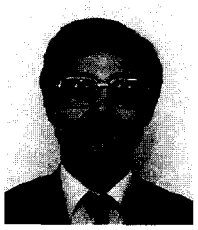
[14] 이성욱, 홍만표, "알려지지 않은 악성 암호화 스크립트에 대한 분석 기법", 한국정보과학회 논문지(게재예정)

이 성 욱



1994 아주대학교 컴퓨터공학과 졸업
1996 아주대학교 교통공학과 석사
1996~1997 기아정보시스템 자능형교통시스템팀.
1997~현재 아주대학교 컴퓨터공학과 박사 과정
관심분야 : 병렬처리, 컴퓨터 보안
E-mail : suleeip@yahoo.co.kr

홍 만 표



1981 서울대학교 자연과학대학 계산통계학과 석사
1983 서울대학교 자연과학대학 계산통계학과 박사
1991 서울대학교 자연과학대학 계산통계학과
1983~1985 울산공과대학 전자계산학과 전임강사
1985~1999 아주대학교 정보 및 컴퓨터공학부 교수
1993~1994 미네소타대학 전자공학과 교환교수
1999~현재 아주대학교 정보통신전문대학원 교수
2000~2001 조지와싱턴 대학교 컴퓨터과학과 교환교수
관심분야 : 병렬처리 및 컴퓨터 보안
E-mail : mphong@ajou.ac.kr