Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.28, NO.2, Apr. 2018 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2018.28.2.397

macOS 운영체제에서 화이트리스트 구축을 위한 신뢰 프로세스 수집 연구*

윤 정 무,† 류 재 철‡
충남대학교

A Method to Collect Trusted Processes for Application Whitelisting in macOS*

Jung-moo Youn,† Jae-cheol Ryu‡
Chung-Nam National University

요 약

악성코드로 의심되는 프로세스를 효과적으로 탐지하기 위해서 블랙리스트 기반으로 제작된 도구들이 가장 보편적으로 사용되고 있다. 블랙리스트 기반의 도구는 기존에 발견된 악성코드의 특징을 추출한 후 이를 이용하여 악성행위를 하는 것으로 추정하는 프로세스와 비교한다. 그러므로 기존에 알려진 악성코드를 탐지하기에는 가장 효과적이지만 악성코드 변종을 탐지하는 것은 한계가 있다. 이런 문제를 해결하기 위해서 블랙리스트와 반대개념인 화이트리스트기반 도구의 필요성이 대두되었다. 화이트리스트기반의 도구는 악성코드 프로세스의 특징을 추출하는 것이 아닌, 신뢰할 수 있는 프로세스를 수집해놓고, 검사하는 프로세스가 신뢰할 수 있는 프로세스인지를 확인한다. 즉, 악성코드가 신규 취약점을 이용해 만들어지거나 변종 악성코드가 등장하더라도 신뢰 프로세스목록에 없기 때문에 효과적으로 악성코드를 탐지해낼 수 있다. 본 논문에서는 macOS 운영체제에서 신뢰할 수 있는 프로세스를 수집하는 연구를 통해 효과적으로 화이트리스트를 구축하는 방법을 제시하고자 한다.

ABSTRACT

Blacklist-based tools are most commonly used to effectively detect suspected malicious processes. The blacklist-based tool compares the malicious code extracted from the existing malicious code with the malicious code. Therefore, it is most effective to detect known malicious codes, but there is a limit to detecting malicious code variants. In order to solve this problem, the necessity of a white list-based tool, which is the opposite of black list, has emerged. Whitelist-based tools do not extract features of malicious code processes, but rather collect reliable processes and verify that the process that checks them is a trusted process. In other words, if malicious code is created using a new vulnerability or if variant malicious code appears, it is not in the list of trusted processes, so it can effectively detect malicious code. In this paper, we propose a method for effectively building a whitelist through research that collects reliable processes in the macOS operating system.

Keywords: Whitelist, Detection, Ransomware

Received(02. 26. 2018), Modified(03. 28. 2018), Accepted(04. 02. 2018)

* 이 논문은 2017년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(R0190-17-2009, 화이트리스트와 상황인지 기술을 이용한 엔드포인트 보호기술 개발)

† 주저자, jmstar1@cnu.ac.kr

‡ 교신저자, jcryou@home.cnu.ac.kr(Correspondingauthor)

I. 서 론

컴퓨터를 사용하는 사람이 증가할수록 사용자 컴퓨터에 몰래 침입해서 악성행위를 하는 악성코드도 증가하고 있다. 최근에는 사용자 몰래 특정 파일들을 암호화 알고리즘을 사용하여 암호화하고 정상 복구를 대가로 돈을 요구하는 랜섬웨어가 유행하고 있다. 과거에 해커들은 주로 자기능력을 과시하기 위한 목적으로 악성코드를 제작했지만 최근에는 악성코드를 수익모델로 간주하고 제작하는 경향이 있다. 랜섬웨어 악성코드의 경우에도 RaaS(Ransomware as a Service)라는 단어가 생겨날 정도로 해커의 많은 관심을 받고 있다[1].

랜섬웨어가 많이 발견되고 있는 만큼 랜섬웨어를 효과적으로 탐지하고 차단하는 기술도 많이 등장하고 있다. 랜섬웨어를 탐지하는 가장 대표적인 방법으로는 블랙리스트기반의 탐지방법이 있다. 알려진 랜섬웨어의 특징을 추출해서 데이터베이스화를 한다. 이렇게 수집된 정보를 이용하여 랜섬웨어로 의심되는 프로세스가 해당 특징을 가지고 있는지 확인함으로써 랜섬웨어를 탐지하는 것이다. 이 방법은 기존의 랜섬웨어를 탐지하는 가장 좋은 방법이다. 하지만 랜섬웨어 변종이 발생할 경우 다시 특징을 추출해야하기 때문에 정보를 수집하는 동안 변종 랜섬웨어를 탐지할 수 없는 문제가 발생한다. 특히, 랜섬웨어는 한번 감염되면 정상적으로 복구할 수 있는 방법이 거의 없기 때문에 변종 랜섬웨어를 탐지할 수 없다는 것은 매우 치명적이다[2].

변종 랜섬웨어를 효과적으로 탐지하기 위해서 등장한 기법은 블랙리스트 기법의 반대개념인 화이트리스트 기법으로 랜섬웨어를 탐지하는 것이다. 기존에는 랜섬웨어로 의심되는 프로세스들을 수집한 랜섬웨어 특징과 비교했지만, 화이트리스트 기반의 탐지방법은 처음부터 신뢰할 수 있는 프로세스만 화이트리스트에 등록해놓고 이 화이트리스트 안에 비교하는 프로세스가 포함되어 있는지를 확인하는 것이다. 즉, 랜섬웨어로 의심되는 프로세스가 신뢰할 수 있는 프로세스인지를 비교하는 것이다. 엄밀히 말하면 랜섬웨어 프로세스뿐만 아니라 일반 악성코드 프로세스, 기존에 설정해놓은 특정행위를 하는 프로세스 모두를 대상으로 비교를 한다. 블랙리스트 기반의 악성코드 탐지도구에 비해 비교횟수가 많아진다는 단점이 있지만, 신뢰할 수 있는 프로세스 목록만 잘 구축한다면 효과적으로 변종 악성코드를 탐지할 수 있는 장점이 있다. 그러므로 화이트리스트 기반의 악성코드 탐지도구는 신뢰할 수 있는 프로세스 목록을 얼마나 잘 수집하느냐에 성능 및 신뢰도가 결정된다.

PC운영체제마다 사용하는 프로세스가 다르며 심지어는 같은 PC 운영체제계열이라도 세부버전이 다른 경우 사용하는 프로세스가 달라질 수 있다. 기본적으로 운영체제를 동작시키는 시스템 프로세스는 신뢰할 수 있는 프로세스이므로 시스템 프로세스를 어떤 방식으로 잘 수집하고, 해당 프로세스가 해커에 의해 변조되지 않았는지를 어떻게 보장할 것인지를 판단해야했다. 신뢰할 수 있는 프로세스라는 것은 좁은 의미로 봤을 때 운영체제에서 기본적으로 제공하는 프로세스로 한정지을 수 있지만 넓은 의미로 보면 사용자가 사용하는 일반적인 프로세스도 포함할 수 있다. 즉, 넓은 의미의 신뢰 프로세스는 악성행위를 하지 않는 프로세스이며, 본 논문에서 수집하는 화이트리스트도 마찬가지로 넓은 의미에서 신뢰 프로세스를 의미한다.

본 논문은 Apple사에서 개발한 macOS PC운영체제에서 신뢰할 수 있는 프로세스를 수집하는 연구를 수행했다. macOS 운영체제에서 제공하는 보안 기능들을 활용하여 화이트리스트로 구축한 프로세스에 대한 무결성을 보장했다. macOS 운영체제는 Windows 운영체제 다음으로 많이 사용되는 운영체제이며, 타 운영체제와 비교했을 때 보안 기능에 더 초점을 맞춘 운영체제라고 평가된다. 하지만 macOS 운영체제역시 악성코드는 꾸준히 발견되고 있으며, 랜섬웨어도 2016년 3월에 KeRanger가 등장하는 등 절대적으로 안전한 PC운영체제는 아니다. 국내에서는 macOS 운영체제의 보안에 대한 연구가 많이 수행되고 있지 않으며, 보안업체 또한 많은 관심을 가지고 있지 않기 때문에 국외의 보안업체와 비교했을 때 보유기술의 수준이 낮은 것으로 판단한다. 하지만 국내에서 macOS는 꾸준히 사용되고 있기 때문에 국내의 사용자들도 잠재위협에 노출되어 있는 상황이다. 그러므로 앞으로 발생할 잠재위협에 효과적으로 대응하기 위해서 선행적으로 macOS 운영체제 보안연구를 수행했다.

II. 관련 연구

2.1 랜섬웨어

2.1.1 랜섬웨어의 개념

랜섬웨어란 악성 URL이나 e-mail의 첨부파일을 통해 사용자의 PC를 감염시키고, 사용자 몰래 특정 확장자의 파일들을 암호화 한다. 암호화가 완료되면 사용자에게 감염파일의 정상복구를 대가로 금전을 요구하는 악성코드다. 넓은 의미로 보면 랜섬웨어는 사용자의 정상적인 파일접근을 막고 정상 사용을 대가로 금전을 요구하는 악성코드로 해석할 수도 있다.

랜섬웨어는 Windows운영체제에서 가장 많이 발견되었다. 많은 사용자를 감염시킬수록 해커의 입장에서는 금전적인 이득을 취할 확률이 높기 때문이다. 분명한 것은 Windows 운영체제뿐만 아니라 macOS 운영체제에서도 꾸준히 랜섬웨어가 발견되고 있다는 것이다[3].

2.1.2 macOS에서의 랜섬웨어

macOS 운영체제에서 발견된 최초의 랜섬웨어는 2013년 7월에 발견된 "FBI Ransomware"이다[4]. 단순히 웹사이트의 시작페이지를 FBI로 위장한 사이트로 변경한 후 다른 웹페이지에는 접근하지 못하도록 하였고, 수상한 행위를 감지했기 때문에 벌금을 내라는 허위 안내문을 통해 계좌로 300달러를 입금할 것을 요구했다. 비록 암호화알고리즘을 사용하여 파일을 암호화하진 않았지만 사용자는 정상적으로 웹서비스를 사용하지 못하였다.

macOS 운영체제에서 공개키 암호화 알고리즘을 사용하여 제대로 만들어진 최초의 랜섬웨어는 2016년 3월에 발견된 "KeRanger"이다. 이 랜섬웨어는 BitTorrent사이트의 취약점을 이용하여 해당 기업의 개발자서명을 훔쳤고, BitTorrent 설치파일에 악성 실행파일을 추가로 삽입하여 웹사이트에 배포했다. 정상적으로 다운받아 사용한 사용자들은 자신도 모르게 악성 실행파일이 실행되었다. 악성 실행파일은 해커의 Command & Control서버와 연결을 맺고, 3일간 잠복기를 거친다. 3일이 지난 후 악성 실행파일은 해당 MacOS 운영체제 버전을 C&C서버로 전송한다. C&C서버에서는 RSA키와 협박파일을 보내준다. 키와 파일을 전달받은 악성 실행파일은 특정 확장자의 파일들을 탐색하여 암호화를 시작한다. 이 때, AES CBC모드를 사용하여 파일을 암호화 한 후 암호화키를 RSA공개키로 암호화함으로써 Brute-Force와 같은 전수대입으로는 암호화키를 확보할 수 없도록 했다. 파일 암호화가 완료되면 사용자에게 협박안내파일을 읽도록 유도했고, 정상복구를 대가로 가상화폐의 일종인 비트코인 1개를 요구했다. 2016년 3월 당시 1비트코인은 한화로 약 40만원의 가치가 있었다.

2.2 macOS 운영체제 소개 및 역사

macOS 운영체제는 Apple기업이 제작한 운영체제로, 1984년 1월부터 Apple 컴퓨터를 이끌어 왔던 맥 오에스 9의 뒤를 잇는 운영체제다. 이전에는 OS X이라는 이름을 가졌고, X는 10을 뜻하는 의미로 로마숫자 X를 사용했다. 이 운영체제는 1996년 12월에 Apple기업이 인수한 NeXT의 기술력으로 만들어 졌으며 유닉스기반으로 제작되었다.

마하커널과 BSD를 기반으로 제작된 MacOS 운영체제는 다윈 운영체제위에 아쿠아 그래픽계층과 응용 프로그램 계층을 올렸고, 이를 위해 쿼츠, 코코아, 카본, 로제타등과 같은 기술을 활용했다[5].

macOS는 최초 "Mac OS X"로 명명되었지만 2016년 6월 세계 개발자 회의(The Apple Worldwide Developers Conference)에서 iOS나 워치OS, tvOS등의 다른 브랜드에 맞춰

Table 1. The operating Systems release date

Operating System	Release date
Mac OS X public Beta	2000. 12. 13
Mac OS X 10.0	2001. 3. 24
Mac OS X 10.1	2001. 12. 25
Mac OS X 10.2	2002. 8. 24
Mac OS X 10.3	2003. 10. 24
Mac OS X 10.4	2005. 4. 29
Mac OS X 10.5	2007. 10. 26
Mac OS X 10.6	2009. 8. 28
Mac OS X 10.7	2011. 7. 20
OS X 10.8	2012. 7. 25
OS X 10.9	2013. 10. 22
OS X 10.10	2014. 10. 16
OS X 10.11	2015. 12. 30
macOS 10.12	2016. 12. 20
macOS 10.13	2017. 12. 25

macOS로 변경되었다. Netmarketshare에 따르면 2017년 12월 PC운영체제 점유율은 macOS의 경우 9.02%로 점유율2위를 차지하였으며 1위는 Windows로 88.51%를 차지했다.

2.3 macOS 운영체제의 보안 기능

2.3.1 Codesign

macOS 운영체제는 Application에 대한 개발자 서명 기능을 제공한다. 개발자 정보는 Apple에서 관리하며, Apple에 개발자로 등록한 개발자만이 App Store나 웹에서 Application배포가 가능하다. Application은 사용자에게는 단일 파일처럼 보이지만 실제로는 Bundle이라는 묶음으로 파일의 묶음형태이며, Apple에서는 개발자의 공개키와 개인키를 관리한다. 개발자가 자신이 개발한 Application에 개발자서명을 한다는 것은 사전에 Apple과 키를 분배한 후, 자신의 개인키로 Application을 암호화하여 전자서명을 하는 것이다. 즉, 동일한 전자서명 값이 나오기 위해서는 반드시 해당 개발자의 개인키가 필요하다. 공개키 암호화 알고리즘을 사용하기 때문에 개발자 서명이 된 Application은 무결성을 보장할 수 있다[6].

2.3.2 System Integrity Protection

System Integrity Protection은 SIP라고 불리기도 한다. /System, /sbin, /bin, /usr 경로 및 하위경로에서 정상적으로 파일을 추가하거나 수정, 제거할 수 있는 프로세스를 제한하였다. Apple의 개발자 서명이 되어있고 사전에 승인된 시스템 프로세스만 해당 폴더에서 정상적인 작업이 가능하도록 했다. 그러므로 위의 4가지 경로에 있는 파일들은 시스템 프로세스에 의해 생성되었기 때문에 타 경로의 파일과 비교했을 때 더 신뢰성이 높다. 사용자가 관리자권한으로 해당 경로에 파일을 추가할 수도 없다. 반드시 해당 경로에 특정 파일을 추가해야하는 경우, 안전모드로 부팅해서 Terminal창에 사용자가 직접 SIP기능을 꺼야한다[7].

2.3.3 Gate Keeper

macOS 운영체제에서는 기본적으로 실행파일에 개발자서명을 하도록 한다. Application형태로 제작된 실행파일의 경우 반드시 개발자서명이 되지만, 유닉스기반으로 제작된 macOS 운영체제에서 실행파일은 Application형태가 아닌 경우도 있다. 즉, 일반적으로 만든 exec 실행파일의 경우 개발자 서명이 없는 경우가 있다. 이런 경우, 개발자가 직접 개발자 서명을 할 수도 있지만, 하지 않는 경우도 있기 때문에 결과적으로 실행파일이 개발자 서명이 되어있지 않은 경우가 발생한다. 보안상 위험할 수 있기 때문에 Apple에서는 파일을 실행할 때 개발자 서명이 되어있지 않은 실행파일의 경우 실행을 하지 못하도록 막을 수 있다. Gate Keeper가 이러한 역할을 수행하는데, 사용자가 관리자권한으로 개발자 서명이 안 된 실행파일도 실행하도록 설정할 수 있다[8].

2.3.4 Sandbox

Sandbox는 실행파일마다 적용되며, 자신의 영역에서만 정상적으로 활동할 수 있도록 프로세스를 관리한다. 특정 Application은 자신의 영역에 있는 정보만 가져올 수 있으며 현재 실행중인 타 Application이 가지고 있는 정보를 가져올 수 없다. 키보드나 마우스이벤트를 수집할 때에도 해당 Application 내에서 발생한 이벤트는 정상 수집 가능하지만 Application 외부에서 발생한 이벤트는 수집할 수 없다. 이 경우 관리자권한으로 해당 프로세스를 신뢰하는 프로세스로 변경해야 가능하다.

III. 제안 연구 방법

화이트리스트는 두 개의 리스트로 구분할 수 있다. 하나는 시스템 프로세스로 구성된 시스템 프로세스 화이트리스트고 나머지 하나는 시스템 프로세스를 제외한 프로세스로 구성된 일반 프로세스 화이트리스트다. 랜섬웨어 등 악성행위를 하는 것으로 의심되는 프로세스를 식별한 경우 화이트리스트와 비교한다. 즉, 의심받는 프로세스는 실제로 악성행위를 하는 프로세스일 수도 있지만 반대로 정상적인 작업을 수행하는 프로세스일 가능성이 충분히 존재한다. 그러므로 프로세스의 잦은 화이트리스트 비교를 고려했을 때 가장먼저 시스템에서 사용하는 프로세스인지를 빠르게 검사한 후 일반 프로세스 중 신뢰하는 프로세스인지 검사하도록 했다.

3.1 시스템 프로세스 화이트리스트 구축

시스템을 운용하기 위해 사용되는 프로세스들은 대표적인 신뢰할 수 있는 프로세스다. 프로세스는 실행파일에 의해 실행되며, 실행파일들은 기본적으로 개발자서명이 되어있다. 구체적인 개발자서명 값을 확인하기 위해서는 macOS에 기본적으로 내장되어 있는 codesign 명령어를 사용한다.

Table 2.는 macOS에서 제공하는 미리보기기능을 수행하는 Preview.app의 개발자서명 정보 값이다. codesign의 옵션 중 verify를 의미하는 -v옵션을 추가할 경우 대상파일의 개발자서명 유효정보를 확인할 수 있다. 개발자서명정보 중 Identifier는 일반적인 도메인주소의 반대형태로 명명하며, Apple사에서 제작한 파일의 경우 "com.apple."으로 시작하게 된다. Preview Application의 경우 "com.apple.Preview"의 값을 가진다. 일반적인 App의 경우 실행파일 뿐만 아니라 설정파일, 실행에 필요한 비 실행파일이 포함된 Bundle형태로 구성된다. 그러므로 실행파일뿐만 아니라 App에 포함된 모든 파일이 개발자서명 값을 가진다.

시스템 프로세스를 수집하는 가장 기본적인 방법은 PC내 모든 파일들의 개발자서명 값을 확인하는 것이다. 단, 실행파일만 수집해야하기 때문에 파일의 header값을 같이 확인해야 한다. 파일의 header를 확인하기 위해서 macOS에 내장되어있는 otool 도구를 활용했다. otool은 파일 또는 라이브러리를 구체적인 정보를 확인할 때 사용한다. 옵션 중 -h 옵션을 사용하면 대상파일의 Mach header값을 확인할 수 있다. macOS실행파일은 Mach-O 형태의 포맷을 가지고 있으며, Mach header값 중 magic 값은 "0xfeedfacf"를 가진다. Windows의 실행파일인 PE의 처음 값이 "MZ"로 시작하는 것과 동일한 개념이며 값 자체가 가지는 의미는 없다. otool 도구를 통해 확인할 수 있는 Preview의 Mach header값은 Table 3.과 같다.

PC의 모든 파일을 대상으로 Mach header값을 확인하고, 해당 파일이 실행파일인 경우 개발자서명 값을 확인해서 Identifier가 "com.apple."으로 시작하는지 확인함으로써 PC내 모든 시스템 프로세스를 수집할 수 있다. 하지만 Identifier가 "com.apple."으로 시작하더라도 시스템 프로세스가 아닐 가능성이 존재한다. 기본적으로 개발자 Identifier의 경우 개발자마다 중복해서 사용할 수 없지만 예외상황이 있다. macOS의 App개발도구인 Xcode.app을 통해 사용자가 새로운 App또는 Command Line Tool을 제작할 때, 개발자 Identifier를 임의로 변경해서 제작할 수 있다. 제작한 App을 배포할 때는 사전에 신청한 개발자 Identifier를 반드시 사용해야 하지만 로컬에서는 Identifier중복검사를 수행하지 않는다. 즉, Xcode를 통해서 얼마든지 "com.apple."으로 시작하는 Identifier를 가진 실행파일을 만들 수 있다. 이 경우, 악성코드가 로컬에서 임의로 실행파일을 생성할 때 "com.apple."으로 시작하도록 할 수 있는 문제가 발생한다. 그러므로 본 논문에서는 macOS의 고유한 보안기능인 System Integrity Protection을 활용했다. SIP는 시스템 파일과 폴더를 수정할 가능성을 사전에 차단함으로써 악성 소프트웨어로부터 운영체제를 보호하는 역할을 한다. SIP에 의해 /System, /sbin, /bin, /usr 경로 내에 있는 모든 파일들은 보호받기 때문에 해당 경로 내 실행파일

Table 2. Preview's codesign Info

Category	Value
Executable	/Applications/Preview.app/Contents/MacOS/Preview
Identifier	com.apple.Preview
format	app bundle with Mach-O thin
platform identifier	1
signature size	4105
Info.plist entries	35

Table 3. Preview's header Info

Category	Value
magic	0xfeedfacf
cputype	16777223
cpusubtype	3
caps	0x80
filetype	2
ncmds	45
sizeofcmds	4776
flags	0x00200085

중 개발자서명 Identifier값이 “com.apple.”으로 시작하는 경우 시스템 프로세스 화이트리스트에 추가했다. SIP기능은 관리자권한을 가진 사용자에 의해 기능을 On/Off할 수 있기 때문에 실행파일을 수집하기 전에 macOS에서 SIP기능이 작동하고 있는지를 확인한다. csrutil status 명령어를 통해 현재 SIP기능이 작동하고 있는지 확인할 수 있다. SIP기능은 사용자가 직접 작동을 중지시킬 수 있지만 반드시 안전모드로 부팅해야만 한다. 현재까지 악성코드가 macOS의 SIP기능을 강제로 중지시킨 사례는 없다.

스크립트언어인 파이썬을 이용해 otool과 codesign명령어를 사용하여 시스템 프로세스 화이트리스트를 구축했다. /System의 하위 경로와 /sbin, /bin, /usr의 하위경로에 있는 실행파일을 검사한 결과는 Table 4.와 같다.

Table 4.는 SIP 보안기능에 의해 보호받는 4가지 경로의 하위 실행파일 개수다. macOS 10.11.6 El Capitan 운영체제 버전에서 실행파일명 및 전체경로를 수집했으며, /System/하위에 실행파일이 가장 많았다. 하지만 수집한 실행파일을 모두 화이트리스트로 구축할 수 없었다. 왜냐하면 /System/폴더 안에는 스크립트 언어 관련 실행파일도 혼재하기 때문이다. macOS 운영체제에서는 스크립트 언어를 기본적으로 제공해주기 때문에 /System 경로 하위에 파이썬 관련 라이브러리가 설치되어 있는 것을 확인했다. 파이썬은 버전별로 설치되어있으며, 관련 실행파일은 총 452개가 있었다. 또 다른 스크립트 언어인 펄의 경우 관련 실행파일이 238개 수집되었다. 그러므로 /System 하위의 실행파일은 모두 시스템 프로세스지만 스크립트 언어관련 실행파일을 제외한 3,230개를 시스템 프로세스 화이트리스트로 수집할 수 있었다. /usr/경로의 경우 SIP에 보호받는 경로지만 하위경로 중 예외경로인 /usr/local/경로 하위로 수집한 실행파일 1,381개를 제외한 1,479개를 시스템 프로세스 화이트리스트로 추가했다.

Table 4. Executable Files using SIP

Path	count
/System/	3,920
/sbin/	40
/bin/	16
/usr/	2,860

Table 5. Executable Files except exception Files

Path	count
/System/	3,230
/sbin/	40
/bin/	16
/usr/	1,479

최종적으로 구축한 시스템 프로세스 화이트리스트에 포함된 실행파일은 Table 5.와 같이 총 4,765개다. 수집한 프로세스가 실제로 Apple사에서 제작한 시스템 프로세스인지 확인하기 위해서 개발자서명을 확인했다. Apple사의 개발자 Identifier는 “com.apple.”으로 시작하기 때문에 macOS 운영체제에 내장되어있는 codesign 도구를 활용해 해당 실행파일에 대한 개발자 Identifier를 확인했다. 그 결과, 시스템 프로세스 화이트리스트로 수집한 실행파일의 Identifier는 모두 “com.apple”으로 시작하는 것을 확인했다.

3.2 일반 프로세스 화이트리스트 구축

시스템 프로세스의 경우 악성코드에 의해 변조될 가능성이 매우 적지만 시스템 프로세스 외 프로세스들은 악성코드에 의해 변조될 가능성이 충분하다. 특히, 최근 몇 년간 발견된 악성코드의 경우 대부분 개발자 서명을 훔쳐서 해당 Application에 몰래 악성실행파일을 추가해놓는 방법으로 악성코드를 PC에 감염시켰다. 그렇기 때문에 아무리 Application이 개발자 서명이 되어있고 애플에 의해 승인받았다고 하더라도 무조건적으로 화이트리스트로 추가할 수 없다.

macOS에서 발견된 랜섬웨어 중 하나인 KeRanger의 경우 최초 감염된 Application을 사용자가 실행했을 때 3일간 잠복기를 가진다. 그러므로 실행직후 이상증상을 발견할 가능성이 매우 낮다. 실행하는 순간에 해당 Application에 의해 동작하는 프로세스가 악성행위를 하는지 안하는지 판단할 수 없기 때문이다.

하지만 대부분의 랜섬웨어가 수익성을 위해 문서파일들을 주 표적으로 삼기 때문에 문서파일에 접근하는 일반 Application의 경우 조금 더 까다로운 조건을 추가한다면 화이트리스트로 구축할 수 있다. 예를 들어, 랜섬웨어가 잠복기를 끝내고 문서파일들

을 암호화하기 시작하는 순간에는 사용자가 문서파일에 접근한 흔적이 없을 것이다. 즉, 문서파일을 사용자가 직접 열어본 흔적이 있다면 그 시점에서 실행된 프로세스는 신뢰할 수 있다.

일반 프로세스 화이트리스트는 사용자가 입력장치를 통해 문서파일을 실행했을 때 실제로 실행되는 프로세스를 수집했다. 입력장치의 입력이 발생했는지 여부를 확인하기 위해서 NSEvent API를 활용했다 . addGlobalMonitorForEventMatchingMask 함수를 NSEventMaskLeftMouseDown 인자 값으로 주는 경우 마우스의 왼쪽 버튼의 입력이 발생했을 때 콜백함수를 호출해준다. 실제로 사용자가 문서파일을 실행할 때 마우스 왼쪽 버튼을 연속으로 두 번 빠르게 클릭한다. 짧은 시간동안의 반복적인 마우스 입력은 clickCount변수에 저장된 정수 값으로 횟수를 측정할 수 있다. clickCount가 1인 경우 마우스 왼쪽 버튼을 한 번 클릭한 것이고, clickCount가 2인 경우 마우스 왼쪽 버튼을 두 번 클릭한 것이다. addGlobalMonitorForEvent MatchingMask 함수에 NSKeyDownMask를 인자로 준 경우에는 키보드 입력 창치에서의 입력 값을 확인할 수 있다. 사용자가 문서파일을 실행하기 위해서 키보드를 이용한 경우, Command 키와 아랫방향의 키를 동시에 입력해야한다. Command 키는 특수키로, 11534600의 상수 값을 가지고, 아래 방향키는 126의 상수 값을 가진다.

위에서 사용한 파일실행방법 이외에도 기타 입력장치를 통해 사용자가 직접 문서파일을 실행할 수 도 있지만, 본 연구에서는 일반적으로 가장 많이 사용하는 키보드와 마우스 입력장치에 한해서 관련 정보를 수집했다. 문서파일의 실행에 필요한 입력장치의 입력 값이 발생한 경우 그 시점에서 실행된 프로세스의 경로를 수집하는 방식으로 일반 프로세스 화이트리스트를 수집했다.

IV. 실 험

본 연구에서 제안한 시스템 프로세스 화이트리스트 수집방법은 MacOS 운영체제에서 제공하는 보안 기능인 SIP를 활용한다. /System, /sbin, /bin, /usr 경로에는 신뢰할 수 있는 실행 파일만 존재한다는 가정 하에서 화이트리스트를 구축하였다. 하위의 예외 경로를 제외한 나머지 경로에서는 애플의 개발자 서명이 된 파일만 존재한다. 단, 운영체제의 버전에 따라서 해당 경로에서 수집되는 실행파일은 달라진다. 또한, 똑같은 운영체제 버전이라도 초기에 수집한 화이트리스트와 어느 정도 사용자가 사용한 후 수집한 화이트리스트는 다르다.

Table 6.은 사용 중인 macOS에서 각각의 경로별 실행파일을 수집하고, 6개월이 지난 후 다시 경로별 실행파일을 수집한 것이다. /System/경로의 실행파일을 비교했을 때, 애플에서 승인된 시스템프로세스에 의해 해당 경로에 실행파일들이 더 추가된 것을 확인할 수 있었으며 오직 실행파일만 들어있는 /sbin/, /bin/의 경우 큰 차이가 없는 것을 확인할 수 있다. 이를 통해 화이트리스트는 사용하고 있는 PC또는 운영체제 세부버전마다 수집되는 프로세스가 다를 수 있으며 같은 운영체제, 같은 버전을 사용하더라도 사용하는 Application에 따라 화이트리스트가 다를 수 있다는 것을 확인했다. 또한, 일정시간이 지난 후 화이트리스트를 수집할 경우 새롭게 추가하거나 배제해야할 실행파일도 발생한다.

수집한 화이트리스트를 사용하여 실제로 랜섬웨어를 탐지할 수 있는지를 테스트 해봤다. 실험환경은 macOS 10.10.5 Yosemite 버전에서 수행했으며 메모리는 2GB, CPU는 Dual Core 2.67GHz로 가상환경을 구축했다. 랜섬웨어의 경우 파이썬을 활용하여 샘플을 제작하였다. 암호화 방식은 AES CBC모드로 구현하였으며, Users/Desktop/pdf/ 경로의 하위에 있는 PDF파일을 암호화하도록 제작했다. 해당 경로에는 PDF파일 150개를 넣어놓고 랜섬웨어를 동작시켰다. KeRanger의 경우 C&C 서버와 통신을 하면서 동작하기 때문에 재연에 한계가 있으므로 이와 유사한 동작방식으로 암호화하는 랜섬웨어 샘플을 제작했다. 다만 KeRanger의 경우 최종적으로 AES키를 RSA키로 다시 암호화하지만 실험목적상 샘플은 AES암호화 알고리즘으로 파일을 암호화하는 과정까지만 제작했다.

Table 6. Process change information

	Basic	0.5Y Later
/System/	3653	4440
/sbin/	42	40
/usr/	1437	1634
/bin/	36	35
sum	5168	6149

Table 7.은 화이트리스트를 구축한 상태에서 랜섬웨어 샘플을 동작시켰을 때 탐지 여부를 확인하는 실험을 진행한 것이다. 파이썬으로 제작된 랜섬웨어 샘플에 의해 문서파일에 대한 의심스러운 행위가 발견되면, 가장먼저 해당 프로세스의 경로를 확인하여 시스템 프로세스 화이트리스트로 수집된 경로와 비교한다. 파이썬은 시스템 프로세스 화이트리스트에 없기 때문에 일반 프로세스 화이트리스트와 비교한다. 사용자의 입력 값이 동반된 프로세스만 일반 프로세스 화이트리스트에 추가되므로 랜섬웨어 샘플의 경우 일반 프로세스 화이트리스트에 없다. 결국 해당 프로세스는 랜섬웨어로 간주, 차단된다.

랜섬웨어 샘플을 100회 동작시켰을 때 매번 탐지에 성공했지만 지연시간이 발생했다. 결국 화이트리스트에는 경로가 저장된 문자열과 비교하기 때문에 화이트리스트에 추가된 프로세스가 많을 경우 랜섬웨어를 차단하기까지 지연시간이 발생한다. 구축한 화이트리스트를 이용하여 실험했을 때, 지연시간은 0.3초 이내로 발생하는 것을 확인했다.

Table 7. Ransomware detection experiment

Item	Value
encryption algorithm	AES CBC mode
encrypted folder	Users/Desktop /pdf
file to encrypt	PDF File
number of measurements	100

V. 결 론

랜섬웨어로부터 발생하는 피해가 점점 증가하고 있는 상황에서 랜섬웨어 변종을 탐지하기 위한 연구는 반드시 필요하다. 화이트리스트를 이용할 경우 블랙리스트기반의 랜섬웨어 탐지방법에 비해 더 많은 프로세스를 검사하기 때문에 연산시간이 더 소모된다. 그럼에도 불구하고 화이트리스트기반으로 랜섬웨어를 탐지할 경우 변종 랜섬웨어를 효과적으로 탐지해낼 수 있는 장점이 있다.

화이트리스트를 구축할 때 macOS 버전마다, 수집하는 PC마다 수집되는 실행파일이 다를 수 있다. 하지만, SIP기능은 버전, 사용시점에 관계없이 계속 적용되고 있는 macOS 고유 보안 기능이기 때문에 SIP를 활용하여 특정 경로의 하위 실행파일을 수집한다면 효과적으로 화이트리스트를 구축할 수 있다. 다만, 시스템 프로세스가 아닌 일반 프로세스를 화이트리스트로 구축하기 위해서 문서 편집 App등 사용자에 의한 수동적인 화이트리스트 구축이 필요하다. 이렇게 구축한 화이트리스트를 활용하기 위해서는 악성행위를 하는 프로세스의 행위에 대한 기준을 정하는 것이 중요하고, 어떤 시점에서 특정 프로세스를 화이트리스트와 비교할 것인지 정해야한다. 본 연구는 화이트리스트를 구축하기 위한 연구이므로 악성 프로세스 식별 관련 내용은 논문에서 제외했지만, 궁극적으로는 악성행위 프로세스 식별연구뿐만 아니라 프로세스의 행위를 제어할 수 있는 연구가 지속적으로 필요하다. 또, 정상적인 문서열람 프로세스에 Code Injection을 하는 랜섬웨어가 제작될 수도 있기 때문에 macOS에서의 Hooking / Code Injection에 관련된 연구도 지속적으로 필요하다.

References

[1] Trend MICRO, "Ransomware as a Service", https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/ransomware-as-a-service-ransomware-operators-find-ways-to-bring-in-business, Sep. 2016

[2] "Ransomware on the Rise, Exposing Vulnerability of EHRs", Biomedical safety & standards, Vol. 46 No. 10, pp.73-75, May. 2016

[3] Ji-yo Park, "A Study on Malicious Behavior Detection of Ransomware in WIndows", A master's thesis, Konkuk University, Aug. 2016

[4] "Objective-See", https://objective-see.com/blog/blog_0x0F.html

[5] "macOS", https://ko.wikipedia.org/wiki/MacOS

[6] "Code Signing Tasks", https://developer.apple.com/library/content/documentation/Security/Conceptual/CodeSigningGuide/Procedures/Procedures.html

[7] "System Integrity Protection", https://en.wikipedia.org/wiki/System_Integrity_Prot

ection
[8] Jonathan Levin, MacOS and iOS Internals, Volume 3: Security & Insecurity, Paperback, 2016

〈저자소개〉

윤 정 무 (Jung-moo Youn) 학생회원
2013년 2월: 충남대학교 컴퓨터공학과 졸업
2018년 3월: 충남대학교 컴퓨터공학과 석사
〈관심분야〉 정보보호, 시스템보안

류 재 철 (Jae-cheol Ryou) 종신회원
1985년 2월: 한양대학교 산업공학과 졸업
1988년 5월: Iowa State University 전산학 석사
1990년 12월: Northwestern University 전산학 박사
1991년 2월~현재: 충남대학교 컴퓨터공학과 교수
〈관심분야〉 정보보호, 네트워크보안, 암호학, 보안프로토콜