침해사고 대응 알림 서비스

(Infringement Incident Response Notification Service)

팀　　명 :　목표75KG

지 도 교 수 :　양환석 교수님

팀　　장 :　은정욱

팀　　원 :　김두형, 박형준

2023. 11.

중부대학교 정보보호학과

목 차

1. 서 론

1.1 연구배경

디지털 전환 시대인 현재 많은 중소기업이 여러 가지 시스템을 도입하며 기업 인프라를 변화시키고 있다. 그러나 디지털 전환에 따르는 보안 위협도 날이 갈수록 증가세를 보이고 있다. 중소기업의 정보보호 인식은 중요하다고 인식되고 있으나 실제 대응에 관한 활동은 부족하거나 이루어지고 있지 않다. KISA에서도 중소기업을 대상으로 정보보안 서비스를 제공하고 있으나 이마저도 활용하지 제대로 활용하지 않는 것이 현실이다. 따라서 중소기업들이 부담을 느끼지 않게 접근성이 쉬우며, 적은 리소스로 높은 효율의 보안 수준 유지를 위한 시스템 구축의 필요성을 느꼈다.

1.2 연구 필요성

보안을 위협하는 공격은 계속해서 진화하고 발전한다. 시스템 변경점에 대한 감시가 가능하며 실시간 알림으로 보안 위협에 커뮤니티를 활용한 실시간 의사소통 및 협업을 강화하여 더 빠른 대응이 가능하며 자동화 프로세스를 통해 데이터의 신뢰성과 무결성 유지가 가능한 침해사고 대응 알림 서비스, 실시간으로 시스템 및 네트워크 상태를 모니터링하고 신속한 대응이 가능한 시스템의 구현으로 자본과 인력이 부족한 소규모 단체 및 개인에게 도움이 되고자 한다.

1.3 연구 목적 및 주제 선정

해당 연구의 주된 목적은 GRR과 Slack을 효과적으로 연동하여 보안 감시 및 위협 대응 능력을 향상시키는 서비스를 개발하는 것에 있다.

주제 선정의 경우 이전 목차에서 언급한 소규모, 저자본은 보안성 강화에 한계가 있기에 이러한 대상들에게 최소의 자원으로 높은 효율의 보안성 강화를 위한 주제를 선정하게 되었다.

2. 관련연구

2.1 Windows

Windows는 Microsoft Corporation이 개발한 운영 체제(OS) 패밀리로, 컴퓨터와 서버용 운영 체제를 포함한다. 개인용 컴퓨터, 노트북, 서버, 태블릿, 스마트폰 및 기타 디바이스에서 사용되며, 다양한 용도와 환경에서 많이 활용된다. Windows 운영 체제는 사용자 친화적이며, 다양한 사용자들이 컴퓨터와 디지털 기술을 이용하는 데 사용된다.

2.2 Liunx

리눅스는 컴퓨터 운영 체제 (Operating System)의 하나로, 오픈소스 운영 체제로 널리 알려져 있다. 안정성, 신뢰성, 보안성, 다중 사용자 지원, 다중 작업 처리 및 커스터마이징 가능성 등 다양한 이점을 제공한다. 그 결과, 서버 환경에서 매우 인기가 있으며, 웹 호스팅, 클라우드 컴퓨팅, 임베디드 시스템, 고성능 컴퓨팅, 개발 및 데스크톱 환경 등 다양한

분야에서 사용된다.

2.3 Python

파이썬은 고수준 프로그래밍 언어로 컴퓨터 프로그래밍을 위한 강력하고 다목적 언어이다. 간결하고 읽기 쉬운 문법, 동적 타입 지정 언어로 변수의 타입을 명시적으로 선언하지 않아도 되며 객체 지향 프로그래밍을 지원하며 대화식 인터프리터로 실시간으로 실행과 결과 확인이 가능하며 크로스 플랫폼을 지원하는 등 많은 장점으로 프로그래머에게 인기가 높은 언어이다.

2.4 Ngrok

로컬 서버를 인터넷을 통해 접근 가능하게 만들어주는 터널링(Tunneling) 서비스 및 도구이다. 개발자와 시스템 관리자들이 웹 서비스, API, 웹 애플리케이션 등을 개발하고 테스트할 때 유용하게 활용된다. ngrok을 사용하면 로컬 호스트(개발 중인 서버 또는 애플리케이션)를 외부에서 접근 가능한 주소로 연결할 수 있다.

2.5 GRR

Google에서 개발한 오픈 소스 컴퓨터 보안 및 사이버 조사 플랫폼이다. GRR은 주로 대규모 네트워크 환경에서 사이버 위협을 탐지하고 대응하는데 사용된다. 이 도구는 보안 전문가, 사이버 보안 분석가, 디지털 포렌식 전문가, 조사관 및 사이버 보안 팀에게 유용하다.

2.6 Slack

비지니스와 팀 커뮤니케이션을 위한 클라우드 기반 메시징 및 협업 플랫폼이다. Slack은 채팅, 파일 공유, 팀 간 협업을 쉽게 할 수 있도록 설계되어 있어 많은 기업과 조직에서 사용된다. 대표적인 기능으로는 워크스페이스(Workspace)와 채널(Channel), 간편한 이모지의 활용, 서드파티 앱과의 연동이 있다.

2.7 VT

전세계에서 널리 사용되는 온라인 바이러스 및 악성 소프트웨어 스캔 서비스이다. 파일, URL 또는 Application의 보안을 평가하기 위해 여러 암호화 및 안티 바이러스 엔진을 사용하여 파일을 스캔한다. 이에 제공하는 서비스는 70개의 이상의 악성 코드 탐지 엔진을 사용하여 파일이나 URL을 스캔하고, 악성 코드 탐지, 보안 위협 분석, 사용자 지정 스캔 등의 방식으로 개인 사용자와 기업의 보안 전략에 필수적인 역할을 한다.

2.8 MITER ATT&CK

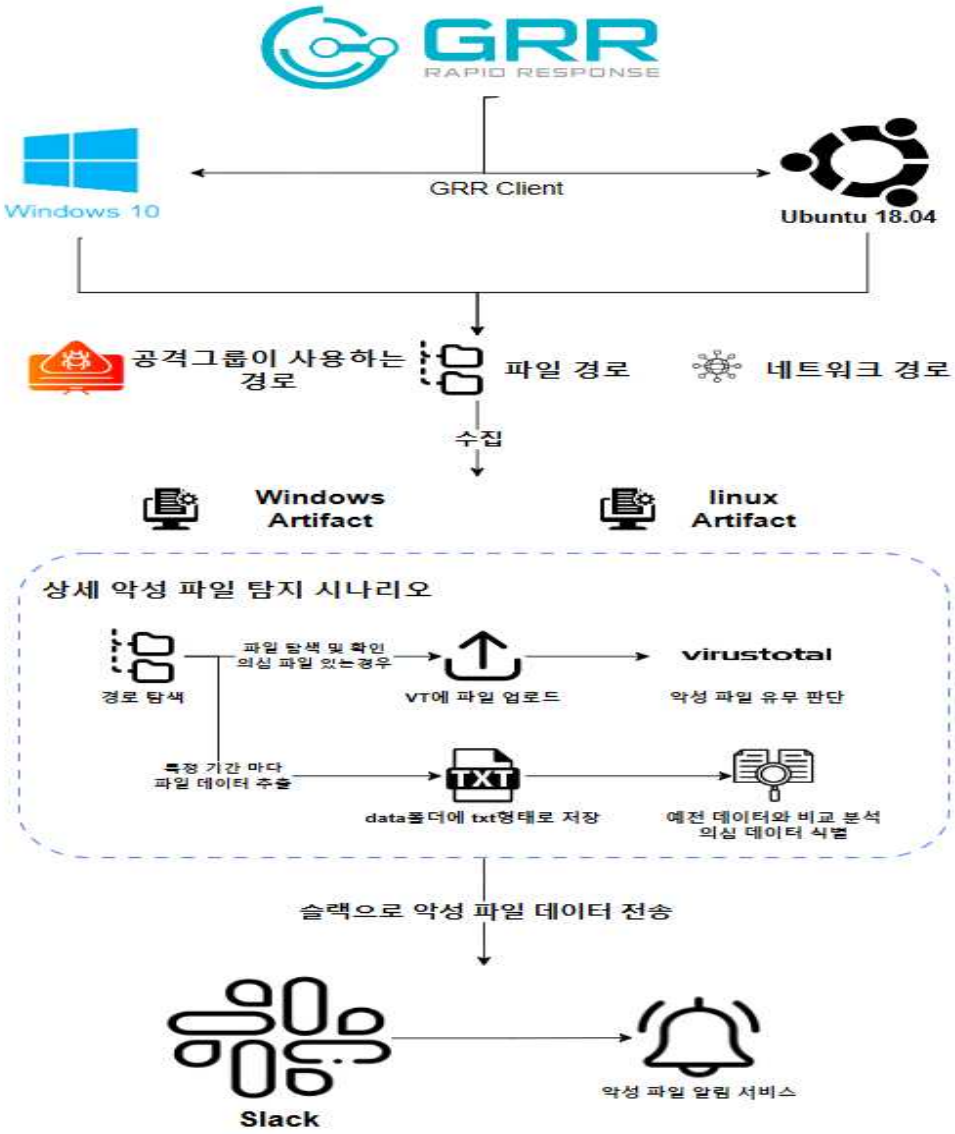
실제 사이버 공격 사례를 분석한 후 공격자가 사용하는 악의적 행위에 대해 전술(Tactics), 기술(Techniques)와 일반적인 지식(Common Knowledge)의 관점으로 분석하여 다양한 공격그룹의 기법에 대한 정보를 목록화 해 놓은 표준적인 데이터이다.
Matrices Mitigations, Groups, Software 등 다양한 정보를 제공하고 있으며, 해당 시스템의 전술과 기술에 관련된 공격정보 및 대응 방안을 확인할 수 있다.

3. 본 론

3.1 시스템 구성

시스템의 구성은 아래와 같이 이루어진다.

1. GRR을 구축할 수 있는 서버를 선정하여 GRR 구축을 완료한다.
2. 대상 OS에 GRR 클라이언트를 설치한다.
3. 자동화 스크립트를 통하여 기록을 원하는 원본 데이터를 수집한다.
4. 스크립트를 통해 설정된 시간마다 추가적으로 데이터를 수집한다.
5. 추가로 수집된 데이터와 원본 데이터를 비교한다.
6. 비교된 데이터에서 의심되는 파일을 VT에서 확인하고 Slack으로 전송한다.



[그림 1. 시스템 구성도]

3.2 프로그램 구성

3.2.1 Windows

1. 연동이 완료된 클라이언트의 PC에서 PowerGRR 모듈과 인증키를 입력한다.

현재 컴퓨터 이름을 출력 받고 출력 받은 컴퓨터을 라벨로 등록하여 해당 PC에서만 제어가 가능하게 설정한다. 진단이 필요한 경로를 입력하여 연동된 Slack으로 보낼 준비를 마친다.

```
Import-Module C:\PowerGRR-0.12.0\PowerGRR-0.12.0\PowerGRR.psd1 -force
$GRRCredential = Microsoft.PowerShell.Security\Get-credential -UserName admin -Message "GRR 인증 password 입력중 기다리세요"

    #컴퓨터이름 출력
$Client_ComputerName = hostname

    #연동된 client를 모두 보는법(clientid만 출력하게 만듦.)
$ALL_Clientid = (Get-GRRClientIdFromComputerName $Client_ComputerName).Clientid

    #클라이언트ID에 라벨 붙이는법(client host name붙)
Set-GRRLabel -ComputerName $Client_ComputerName -Label Park

    #클라이언트 ID를 라벨로 찾는법
$clients = Find-GRRClientByLabel -SearchString Park

    #HUNTING 준비 및 시작
    #프로세스 현황 준비 + huntid만 출력 변수에 저장
    #시스템 폴더 현황
$Huntid_systeml = New-GRRHunt -HuntDescription "file system path hash save1" -Flow FileFinder -RuleType Label -Label Park -ActionType Hash -Mode ALL_HITS -Path 'C:\WINDOWS\*++','C:\Users\*++','C:\ProgramData\*++', 'C:\$Recycle.Bin

$Huntid_systeml.hunt_id > huntid.txt

    #GRR헌트 시작
Start-GRRHunt -HuntId $Huntid_systeml.hunt_id
```

[그림 2. 윈도우 GRR 원격 명령어 입력을 위한 PowerGRR 스크립트 일부]

2-1. GRR에서 수집된 데이터와 Slack의 원활한 연동을 위하여 봇의 설정을 체크한다. 해당 설정을 통하여 필요한 경우 명령어를 변경할 수 있다.
기본 대화형 설정은 [@] 호출을 통해 설정된 봇을 입력한 상태에서 [/] 입력이 필요하다.

```
# 사용자 정의 Slack 슬래시 명령 처리
@app.route('/slash/', methods=['POST'])
def hello_slash():
    command = request.form['command']
    text = request.form['text']
    user_id = request.form['user_id']
    channel = request.form['channel_id']  # 채널 ID 가져오기

    if command == '/windows_start':
        # 추가: PowerShell 스크립트 실행 및 인증 메시지 대화 상자 처리 함수 호출
        execute_ps_script()
        answer = create_start_block()  # 봇의 명령어를 호출합니다.
    elif command == '/windows_progress':
        answer = create_progress_block()  # 봇의 명령어를 호출합니다.
        answer = "@grr_bot 진행중"
    elif command == '/windows_result':
        answer = "@GRR BOT /windows_result 를 입력하여 주세요"
    else:
        answer = "지원하지 않는 명령입니다."

    return make_response(answer, 200, {"content_type": "application/json"})
```

[그림 3. 윈도우 슬랙봇 연동 소스코드 일부]

2-2. [/]를 명령어를 제외한 [@] 호출 명령어의 답변을 확인할 수 있다.

```python
# Slack 명령어 처리 및 응답 함수
def get_answer(text):
    user_text = text.replace(" ", "")

    answer_dict = {
        '/windows_start': create_start_block(),
        '/windows_progress': create_progress_block(),
        '/windows_result': scan_file_and_send_result(),
        'help': '"봇으로 진행하고 싶은 사항을 입력해주세요 \n질문 옵션을 알고싶으시면 <@grr_bot 옵션> 을 검색해주세요"',
        '옵션': '/windows_start : GRR 진단 시작\n/windows_progress : PC 진단 진행사항\n/windows_result : PC 진단 결과',
        '진행중': grr_art()
    }

    if user_text == '' or None:
        return "알 수 없는 질의입니다. 답변을 드릴 수 없습니다."
    elif user_text in answer_dict.keys():
        return answer_dict[user_text]
    elif user_text == '진행사항':
        return create_progress_block()
    else:
        for key in answer_dict.keys():
            if key.find(user_text) != -1:
                return "연관 단어 [" + key + "]에 대한 답변입니다.\n" + answer_dict[key]

        for key in answer_dict.keys():
            if answer_dict[key].find(text[1:]) != -1:
                return "질문과 가장 유사한 질문 [" + key + "]에 대한 답변이에요.\n" + answer_dict[key]

    return text + "은(는) 없는 질문입니다."
```

[그림 4. 윈도우 슬랙봇 입력 명령어 소스코드 일부]

3. 기본 값으로 설정한 최초 실행 명령어 [/windows_start] 명령어를 입력하게 되면 연동된 스크립트에 의해서 데이터 수집이 시작된다. 스크립트 실행 시 필요한 인증은 pyautogui를 이용하여 사용자가 직접 입력할 필요는 없다.

```python
# PowerShell 스크립트 실행 및 인증 처리 함수
def execute_ps_script():
    ps_script_path = r"C:\\PowerGRR-0.12.0\\PowerGRR-0.12.0\\grr_auto_powershell_sim.ps1"
    username = "admin"
    password = "park1004"

    try:
        # PowerShell 스크립트 실행
        process = subprocess.Popen(
            ['powershell', '-ExecutionPolicy', 'Bypass', '-File', ps_script_path],
            stdin=subprocess.PIPE,
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
            text=True
        )

        # 인증 대화 상자가 나타날 때까지 대기
        time.sleep(1)  # 필요에 따라 대기 시간을 조정합니다.
        # ID 입력
        pyautogui.write(username)  # ID 입력
        pyautogui.press('tab')  # 탭 키로 비밀번호 입력 필드로 이동
        pyautogui.write(password)  # 비밀번호 입력
        pyautogui.press('enter')  # Enter 키로 로그인 또는 확인
        time.sleep(0.5)
        pyautogui.write(password)  # 비밀번호 입력
        pyautogui.press('enter')
        # process.wait(20)

    except Exception as e:
        print(f"An error occurred: {str(e)}")
```

[그림 5. 윈도우 데이터 수집 명령어 실행 과정 소스코드 일부]

```
# GRR 시작 블록 메시지 생성 함수
def create_start_block():
    start_block = {
        "blocks": [
            {
                "type": "section",
                "text": {
                    "type": "plain_text",
                    "text": "*Windows PC 진단 GRR이 실행되었습니다.*",
                    "emoji": True
                }
            },
            {
                "type": "divider"
            },
            {
                "type": "section",
                "text": {
                    "type": "mrkdwn",
                    "text": "*:star::star::star:*Runing GRR*:star::star::star:\n*구동시간:* {}_windows PC\n*예상시간 30 ~ 1:30 소요 예정*\n진행사항을 알고싶으면\n /windows_progress을 입력해 주세요".format(get_dateti
                },
                "accessory": {
                    "type": "image",
                    "image_url": "https://blog.kakaocdn.net/dn/vw5kI/btrX2V1UhXb/nwe1Y7uw07dzkLy0bSiu0k/img.jpg",
                    "alt_text": "alt text for image"
                }
            },
            {
                "type": "divider"
            }
        ]
    }
    return start_block
```

[그림 6. 윈도우 /windows_start 명령어 실행 화면 소스코드 일부]

4. 명령어가 정상적으로 실행되었다면 반환받는 메시지로 진행 사항을 확인 가능하며 txt 로 저장되는 진행 사항을 탬플릿에 맞게 확인 가능하다.

```
# 진행 상황 블록 메시지 생성 함수
def create_progress_block():
    current_datetime = get_datetime()  # 현재 날짜와 시간 가져오기

    # grr_hunt.txt 파일 읽기
    try:
        with open("C:\PowerGRR-0.12.0\PowerGRR-0.12.0\process.txt", "r", encoding="utf-16") as file:
            show_delita = file.read()
    except UnicodeDecodeError:
        # If the file is not UTF-8 encoded, try another encoding (e.g., ISO-8859-1)
        with open("C:\PowerGRR-0.12.0\PowerGRR-0.12.0\process.txt", "r", encoding="ISO-8859-1") as file:
            show_delita = file.read()
```

[그림 7. 윈도우 진행 사항 확인 소스코드 일부]

```
    progress_block = {
        "blocks": [
            {
                "type": "section",
                "text": {
                    "type": "plain_text",
                    "emoji": True,
                    "text": "GRR 시스템 실행중입니다."
                }
            },
            {
                "type": "divider"
            },
            {
                "type": "section",
                "text": {
                    "type": "mrkdwn",
                    "text": "*GRR START TIME:* {}\nWINDOWS 내 파일 시스템 내부 및 경로로 점검 진행중".format(current_datet
                },
                "accessory": {
                    "type": "image",
                    "image_url": "https://api.slack.com/img/blocks/bkb_template_images/notifications.png",
                    "alt_text": "calendar thumbnail"
                }
            },
            {
                "type": "rich_text",
                "elements": [
                    {
                        "type": "rich_text_section",
                        "elements": [
                            {
                                "type": "text",
                                "text": "진단 세부 내역",
                                "style": {
```

[그림 8. 윈도우 진행 사항 실행 화면 소스코드 일부]

5. 데이터 수집이 진행되는 중 Slack에 “진행 중”이 표시된 경우 스크립트에 의해 스케줄러 등록을 위한 소스로 넘어간다.

```python
def grr_art():
    ps_script_path = r"C:\\PowerGRR-0.12.0\\PowerGRR-0.12.0\\ret_auto.ps1"

    try:
        # PowerShell 스크립트 실행
        process = subprocess.Popen(
            ['powershell', '-ExecutionPolicy', 'Bypass', '-File', ps_script_path],
            stdin=subprocess.PIPE,
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
            text=True
        )
    except Exception as e:
        print(f"An error occurred: {str(e)}")
```

[그림 9. 윈도우 스케줄러 등록 소스코드 일부]

6. 선행 과정의 자동화를 위하여 스케줄러를 등록하고 실행 횟수, 실행 시간을 스크립트가 실행되는 시간 +30분으로 설정하고 30분 간격으로 반복 실행이 되는 스케줄러를 등록하고 기본 설정된 4시간이 지나면 스크립트가 종료되며 등록된 스케줄러가 삭제된다.

```powershell
# 현재 날짜와 시간을 얻습니다.
$currentTime = Get-Date

# 작업 액션을 생성합니다. 실행할 Python 스크립트 파일을 지정합니다.
$action = New-ScheduledTaskAction -Execute 'python.exe' -Argument 'C:₩PowerGRR-0.12.0₩PowerGRR-0.12.0₩py_slackbot_test₩ret_grr.py'

# 작업 트리거를 생성합니다. 현재 실행 시간에서 30분 후에 시작하도록 설정합니다.
$trigger = New-ScheduledTaskTrigger -Once -At ($currentTime.AddMinutes(30))

# 작업을 등록합니다. 작업 이름과 트리거를 지정합니다.
Register-ScheduledTask -TaskName 'grr_ret' -Trigger $trigger -Action $action

# 작업에 설명을 추가합니다.
$task = Get-ScheduledTask -TaskName 'grr_ret'
$task.Description = "grr 결과값 받아오는 작업 실행중"

# 작업 트리거를 수정하여 30분 간격으로 반복하도록 설정합니다.
$trigger.RepetitionInterval = "PT30M"  # 30분 간격
$trigger.RepetitionDuration = "P1D"  # 1일 동안 반복

# 수정된 트리거를 업데이트합니다.
Set-ScheduledTask -TaskName 'grr_ret' -Trigger $trigger

# 4시간 후에 작업을 삭제
Start-Sleep -Seconds (4 * 60 * 60)  # 4시간 대기
Unregister-ScheduledTask -TaskName 'grr_ret' -Confirm:$false  # 작업 삭제
```

[그림 10. 윈도우 자동 수집을 위한 스케줄러 스크립트 일부]

7. 데이터 수집의 결과 GRR 자체의 이슈로 설정한 시간과 정확히 일치하지 않는 부분을 고려하여 30분마다 반복으로 결과를 받을 수 있게 설정하였다. 마찬가지로 인증에 필요한 내용은 pyautogui에 의해 자동적으로 넘어간다.

```python
import subprocess
import pyautogui
import time

with open("C:\\PowerGRR-0.12.0\\PowerGRR-0.12.0\\huntid.txt", "r",encoding="UTF-16") as i:
    huntid = i.read()

# PowerShell 스크립트를 생성하고 파일로 저장
ps_script = f"""
Import-Module C:\\PowerGRR-0.12.0\\PowerGRR-0.12.0\\PowerGRR.psd1 -force;
$GRRCredential = Microsoft.PowerShell.Security\\get-credential -UserName admin -Message 'GRR 인증 password 입력중 기다리세요';
Get-GRRHuntResult -HuntId {huntid}_system1 -ShowJSON > C:\PowerGRR-0.12.0\PowerGRR-0.12.0\py_slackbot_test\json_ret\\test2.json
"""

# 스크립트를 파일로 저장
with open("C:\\PowerGRR-0.12.0\\PowerGRR-0.12.0\\ret_grr.ps1", "w") as script_file:
    script_file.write(ps_script)

# PowerShell 스크립트 실행
ps_script_path = "C:\\PowerGRR-0.12.0\\PowerGRR-0.12.0\\ret_grr.ps1"
username = "admin"
password = "park1004"

process = subprocess.Popen(
    ['powershell', '-ExecutionPolicy', 'Bypass', '-File', ps_script_path],
    stdin=subprocess.PIPE,
    stdout=subprocess.PIPE,
    stderr=subprocess.PIPE,
    text=True
)

# 인증 대화 상자가 나타날 때까지 대기
time.sleep(1)  # 필요에 따라 대기 시간을 조정합니다.
# ID 입력
pyautogui.write(username)  # ID 입력
pyautogui.press('tab')  # 탭 키로 비밀번호 입력 필드로 이동
pyautogui.write(password)  # 비밀번호 입력
pyautogui.press('enter')  # Enter 키로 로그인 또는 확인
time.sleep(0.5)
pyautogui.write(password)  # 비밀번호 입력
pyautogui.press('enter')
```

[그림 11. 윈도우 스케줄러 실행 소스코드 일부]

8. 수집된 데이터는 코드를 최초 실행했을 때 저장된 원본 데이터에 기록된 SHA256의 값과 비교하여 추가, 변경된 내용을 변수에 입력하여 json 파일을 별도로 저장한다.

```python
def scan_file_and_send_result():
    # JSON 파일 두 개를 읽어옵니다.
    with open(r"C:\PowerGRR-0.12.0\PowerGRR-0.12.0\py_slackbot_test\json_ret\test1.json", "r") as file1, open(r"C:\PowerGRR-0.12.0
        data1 = json.load(file1)
        data2 = json.load(file2)

    # 각 파일에서 hash 값을 추출합니다.
    hash_set1 = set(entry["payload"]["hash_entry"]["sha256"] for entry in data1["items"])
    hash_set2 = set(entry["payload"]["hash_entry"]["sha256"] for entry in data2["items"])

    # 두 세트를 비교하여 중복되지 않는 항목을 찾습니다.
    unique_hashes = hash_set1.symmetric_difference(hash_set2)

    # 중복되지 않는 hash 값을 가진 항목을 리스트로 저장합니다.
    unique_entries = []

    # index를 1부터 시작하도록 초기화합니다.
    index = 1

    for entry in data1["items"] + data2["items"]:
        if entry["payload"]["hash_entry"]["sha256"] in unique_hashes:
            sha256 = entry["payload"]["hash_entry"]["sha256"]
            sha1 = entry["payload"]["hash_entry"]["sha1"]
            md5 = entry["payload"]["hash_entry"]["md5"]
            path = entry["payload"]["stat_entry"]["pathspec"]["path"]
            timestamp = entry["timestamp"]
            unique_entry = {
                "index": index,
                "sha256": sha256,
                "sha1": sha1,
                "md5": md5,
                "path": path,
                "timestamp": timestamp
            }
            unique_entries.append(unique_entry)
            # 다음 항목을 위해 index를 증가시킵니다.
            index += 1

    # 중복되지 않는 hash 값을 가진 항목을 JSON 파일로 저장합니다.
    with open(r"C:\PowerGRR-0.12.0\PowerGRR-0.12.0\py_slackbot_test\json_ret\result_grr_data.json", "w", encoding="UTF-16") as out
        json.dump(unique_entries, output_file, indent=4)
```

[그림 12. 윈도우 결과 값 정제 과정 소스코드 일부]

9. 정제된 내용에서 추가, 변경된 내용에 대해서 정제 마지막에 저장한 json 파일을 대상으로 바이러스 토탈과 연동하여 악성 유무를 판단한다.

```
# VT 스캔 및 결과 저장
api_key = "1222899cb078a1baedc9db0755c86f5ff283d1511ddd5fc33e12fb084724f246"
result_file_path = r"C:\PowerGRR-0.12.0\PowerGRR-0.12.0\py_slackbot_test\json_ret\result_grr_data.json"

response_list = []

with open(result_file_path, "r", encoding="utf-16") as result_file:
    unique_entries = json.load(result_file)

if unique_entries:
    entry = unique_entries[0]
    sha256_hash = entry["sha256"]
    print(sha256_hash)
    file_path = entry["path"]

    scan_url = f"https://www.virustotal.com/api/v3/files/{sha256_hash}"
    headers = {
        "x-apikey": api_key
    }

    response = requests.get(scan_url, headers=headers)
```

[그림 13. 윈도우 VT 연동 소스코드 일부]

10. 자동화로 바이러스 토탈에 등록된 백신들에 수집한 HASH값을 입력하여 결과를 Slack으로 출력한다.

```
if response.status_code == 200:
    scan_result = response.json()

    # VT 결과 처리
    file_risk, status_message, formatted_date, formatted_time, vt_link = "", "", "", "", ""
    if scan_result:
        VT_TIME = scan_result["data"]["attributes"]["last_modification_date"]
        korea_timezone = pytz.timezone("Asia/Seoul")
        utc_time = datetime.fromtimestamp(VT_TIME, tz=pytz.UTC)
        korea_time = utc_time.astimezone(korea_timezone)

        analysis_results = scan_result["data"]["attributes"]["last_analysis_results"]

        total_engines = len(analysis_results)
        malicious_engines = sum(1 for result in analysis_results.values() if result.get("category") == "malicious")

        if total_engines > 0:
            file_risk = f"파일 위험도: {malicious_engines}/{total_engines}"
        else:
            file_risk = "파일 위험도: 알 수 없음"

        formatted_date = korea_time.strftime("%Y-%m-d %H:%M:%S")

        current_time = datetime.now()
        formatted_time = current_time.strftime("%Y-%m-d %H:%M:%S")

        file_status = scan_result["data"]["attributes"]["last_analysis_stats"]["harmless"]
        if file_status > 0:
            status_message = "스캔 결과: 악성 파일일 수 있습니다. 관리자에게 문의하세요."
        else:
            status_message = "스캔 결과: 안전한 파일입니다."

        vt_link = f"VirusTotal 파일 정보 링크: https://www.virustotal.com/gui/file/{sha256_hash}/details"
```

[그림 14. 윈도우 최종 결과 Slack 연동 소스코드 일부]

3.2.2 Liunx

가상의 시나리오로 데이터를 매월 1일마다 수집 대상 클라이언트의 정보를 별도로 저장하여 데이터를 수집하는 과정으로 진행하였다.

1. APT공격 사례와 OSINT자료를 활용하여 C2 Server, 기본 구성 경로, 네트워크 정보, 파일 시스템을 대상으로 수집하였다. C2 Server의 경우 각 경로 중 100000바이트 이상의 크기 가진 파일을 대상으로 수집한다.

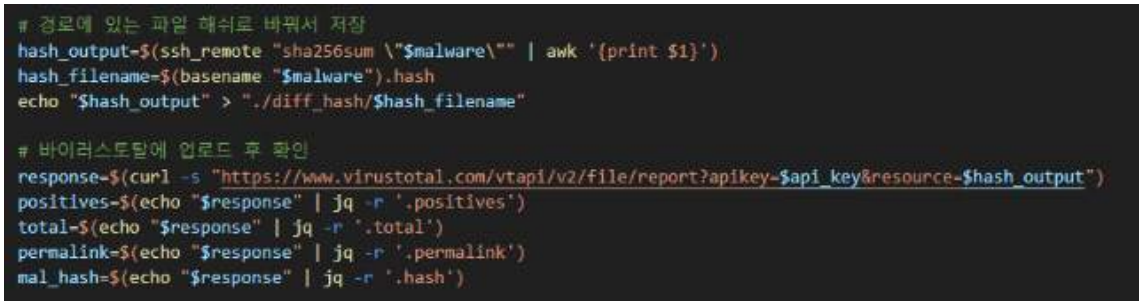
[그림 15. 리눅스 GRR 데이터 수집 소스코드 일부]

2. 기존 저장한 원본 데이터와 추가 수집한 정보를 비교한다. 변경된 내용이 있다면 별도로 저장을 진한다.

```
# 파일 이름 시간 저장
current_datetime=$(date +'%Y%m%d_%H%M')

# 파일 내용 비교
send_slack_notification() {
  message="$1"
  filename="./date/$current_datetime.txt"
  diff_directory="./diff_data/$current_datetime"

  if [ -n "$c2_files" ]; then
    malware=$(echo "$c2_files" | head -n 1)
  fi

  if [ ! -d "$diff_directory" ]; then
    mkdir -p "$diff_directory"
  fi

  if [ -f "$filename" ]; then
    diff_output=$(diff -u "$filename" "./date/2023MM_main.txt")
    if [ -n "$diff_output" ]; then
      echo "$diff_output" >> "$diff_directory/$current_datetime.txt"
    fi
  fi
```

[그림 16. 리눅스 데이터 비교 소스코드 일부]

3. 비교된 값중 용량에 변화가 발생한 파일을 대상으로 위 코드에서 지정한 변수에 의해 별도로 저장하고 해당 값을 바이러스 토탈과 연동하여 악성 유무를 판단한다.



```
# 경로에 있는 파일 해쉬로 바꿔서 저장
hash_output=$(ssh_remote "sha256sum \"$malware\"" | awk '{print $1}')
hash_filename=$(basename "$malware").hash
echo "$hash_output" > "./diff_hash/$hash_filename"

# 바이러스토탈에 업로드 후 확인
response=$(curl -s "https://www.virustotal.com/vtapi/v2/file/report?apikey=$api_key&resource=$hash_output")
positives=$(echo "$response" | jq -r '.positives')
total=$(echo "$response" | jq -r '.total')
permalink=$(echo "$response" | jq -r '.permalink')
mal_hash=$(echo "$response" | jq -r '.hash')
```

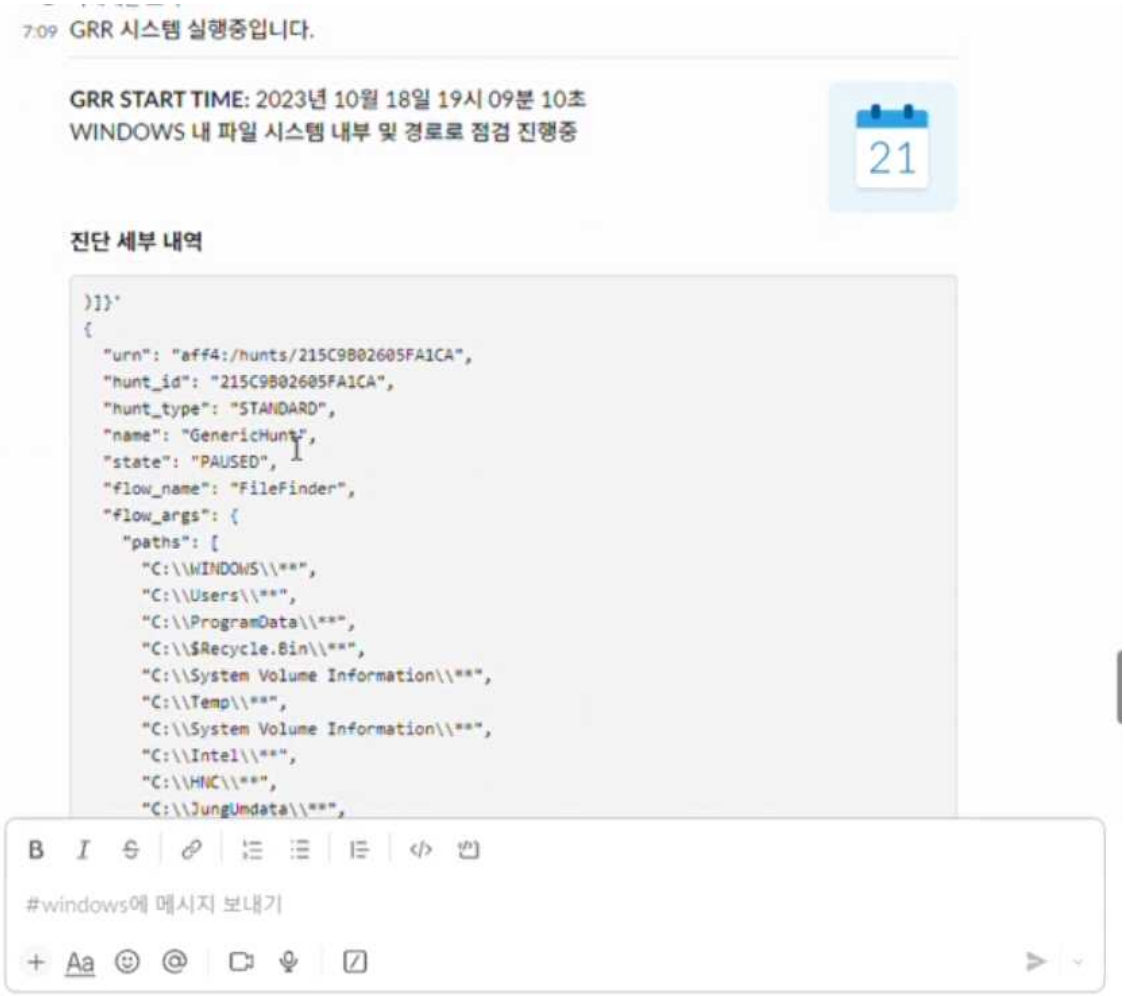
[그림 17. 리눅스 악성 파일 탐지 소스코드 일부]

4. 악성 유무의 판단이 끝난 결과를 $filename으로 저장하고 변수가 비어있는 경우에는 N/A로 초기화 한다.

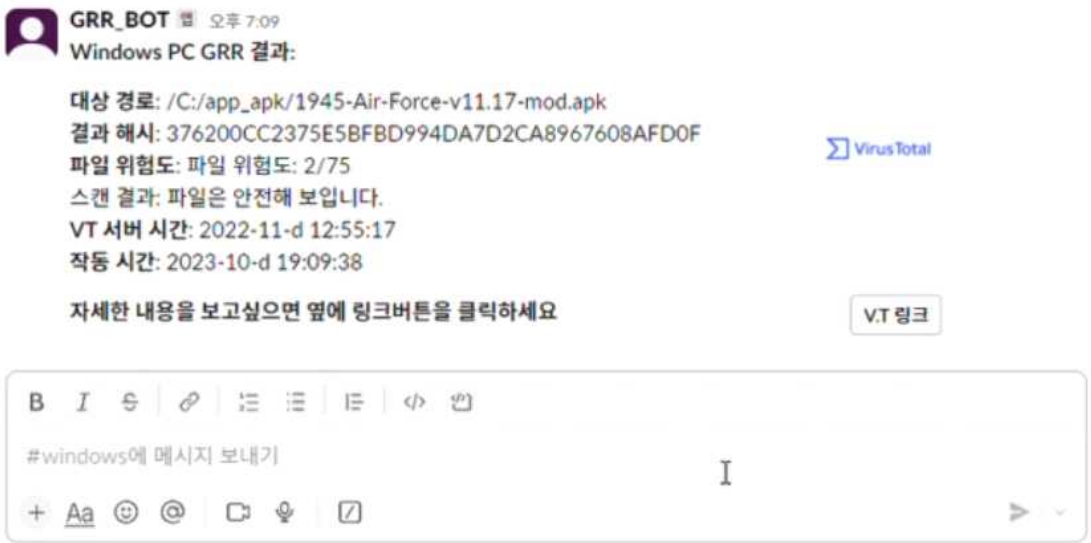
```
if [[ -z "$positives" ]]; then
  positives="N/A"
fi

if [[ -z "$mal_hash" ]]; then
  positives="N/A"
fi

if [[ -z "$total" ]]; then
  total="N/A"r
fi

if [[ -z "$permalink" ]]; then
  permalink="N/A"
fi

echo "Ubuntu 18.04 LTS 결과:" >> "$filename"
echo "대상 경로: $local_path" >> "$filename"
echo "결과 해시: $mal_hash" >> "$filename"
echo "파일 위험도: $positives / $total" >> "$filename"
echo "VT 링크: $permalink" >> "$filename"
```

[그림 18. 리눅스 변수 초기화 및 결과 출력 소스코드 일부]

[그림 19. 슬랙 윈도우 GRR 데이터 수집 시작]



[그림 20. 슬랙 윈도우 GRR 데이터 수집 과정]

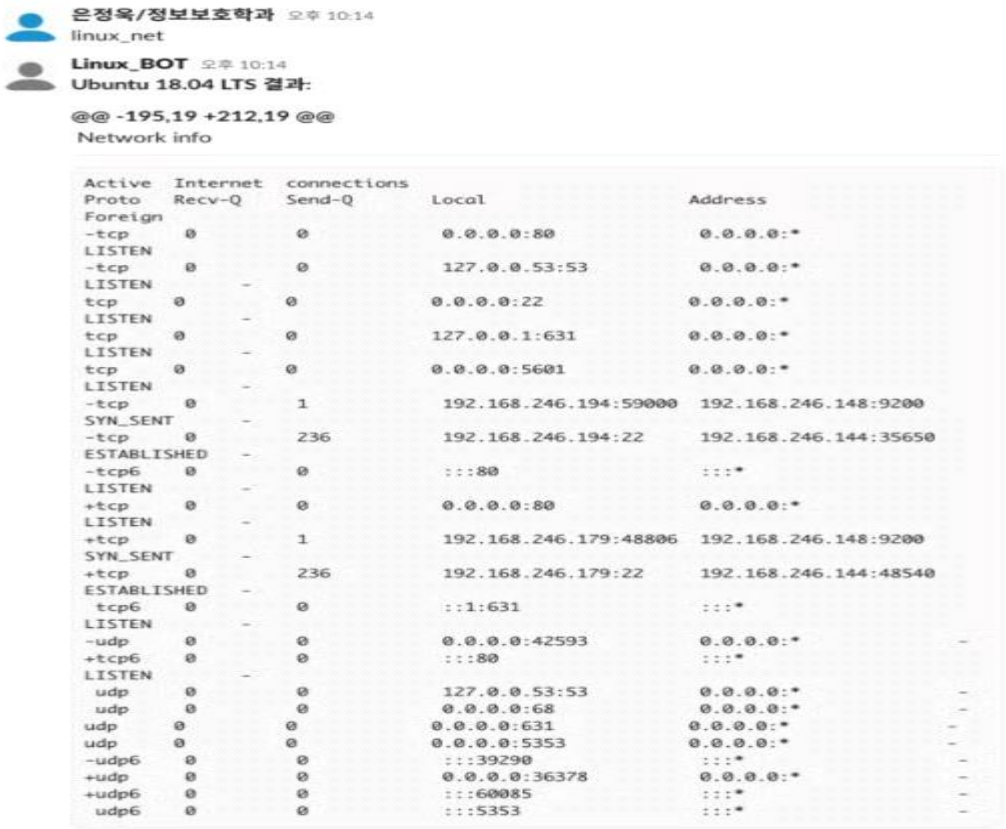


[그림 21. 슬랙 윈도우 GRR 데이터 수집 결과]

[그림 22. 슬랙 리눅스 GRR 데이터 수집 시작]

[그림 23. 슬랙 리눅스 GRR file 데이터 수집 과정]

[그림 24. 슬랙 리눅스 GRR C2 데이터 수집 과정]



[그림 25. 슬랙 리눅스 GRR 네트워크 데이터 수집 과정]

[그림 26. 슬랙 리눅스 GRR 데이터 수집 결과]

4. 분 석

4.1 활용 결과 및 성능

기업에서 높은 사용률을 보이는 슬랙과 많은 이용자와 안정성이 높은 백신들의 리소스를 이용할 수 있는 바이러스 토탈 등의 연동으로 최소한의 리소스로 높은 효율을 원했던 초기 목표와 부합한 결과를 보인다고 판단한다. GRR 서버 자체에서 제한이 걸려있는 수준의 크기, 바이러스 토탈 API에서 허용 가능한 크기를 넘지 않았을 경우 문제없이 작동하는 것을 확인할 수 있다.

4.2 추후 보완사항

현재 슬랙에 한하여 알림 서비스를 제공하는 형식이지만 사용자가 많은 서비스를 추가로 선정하여 서비스 확장 가능하다. 대화형 방식을 채택하여 사용하고 있으나 버튼 형식 사용이 가능한 경우 알림 서비스 내 버튼과 연동하여 사용자가 버튼 클릭으로 더 간편하게 악성 파일에 대한 처리가 가능하게 추가 예정이며, 현재 테스트된 운영체제 외 추가 운영체제에 대한 서비스 확장 고려 중에 있다.

5. 결 론

5.1 결 론

GRR과 Slack을 효과적으로 연동한 서비스를 개발함으로써 보안 감시 및 위협 대응 능력을 크게 향상시켰다. 이를 통해 실시간으로 보안 이벤트를 모니터링하고 신속하게 대응할 수 있게 되었으며, 한정된 자원을 활용하여 보안성을 높이는 데 있어서 새로운 방향과 가능성을 제시하였다. 보안 감시와 위협 대응 능력을 향상시키는 서비스의 개발과 소규모, 저자본 환경에서의 보안성 강화가 가능하였다.

5.2 기대효과

GRR-Slack을 활용한 연동은 대응 프로세스를 단순화하여 더 효과적이고 효율적인 업무 수행이 가능하다. 보안 이벤트에 관한 모니터링으로 빠른 대응이 가능하며 기존 데이터의 무결성에 대한 검증이 가능하다. 정제된 데이터의 결과에 대한 알림으로 사용자의 직관적인 보안 대처가 가능하다. 알림 서비스에 대한 확장성이 다양하다. 가장 큰 기대효과는 목표로 했던 적은 리소스를 활용하여 큰 효율을 얻는다는 점에 있다.

6. 별 첨

6.1 소스 코드

https://github.com/JeungWookEun/Graduation-Project

6.2 발표 자료

침해사고 대응 알림 서비스
(Infringement Incident Response Notification Service)

목표75KG

팀원 소개

은정욱

팀장

총괄
윈도우 서비스 개발

박형준

팀원

윈도우 서버 구축
윈도우 서비스 개발

김두형

팀원

리눅스 서버 구축
리눅스 서비스 개발

1

프로젝트 배경

< 한국인터넷진흥원(KISA) 침해사고 신고 건수 통계(건) >

구분 \ 연도	2021년		2022년		2023년
	상반기	하반기	상반기	하반기	상반기
건수	298	342	473	669	664
합계	640		1,142		664

(단위:건)

구분	2019년	2020년	2021년	2022년	2023년 8월
DDoS 공격	155	213	123	122	159
악성코드 감염·유포	59	140	234	347	218
시스템 해킹	204	250	283	673	513
합계	**418**	**603**	**640**	**1,142**	**890**

(자료: 한국인터넷진흥원, 2023.8월말 기준)

침해사고 신고 건수는 매년 증가
보안을 위협하는 공격은 계속해서 진화하고 발전함

보안 감시 및 위협 대응 능력 향상
소규모, 저자본 보안성 강화

2

프로젝트 개요

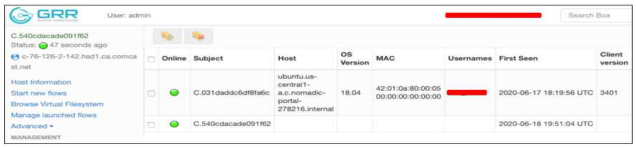
slack
+
GRR RAPID RESPONSE
+
VirusTotal

3

프로젝트 개요 - GRR

GRR(Google Rapid Response) GRR RAPID RESPONSE

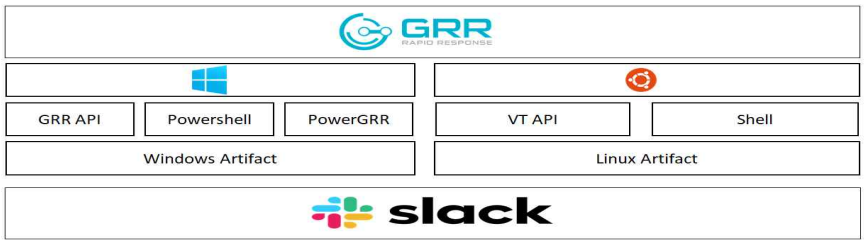
- 원격 라이브 포렌식에 중점을 둔 침해사고 대응 프레임워크이다.
- Linux, Mac OS 및 클라이언트에 대한 크로스 플랫폼 지원한다.



4

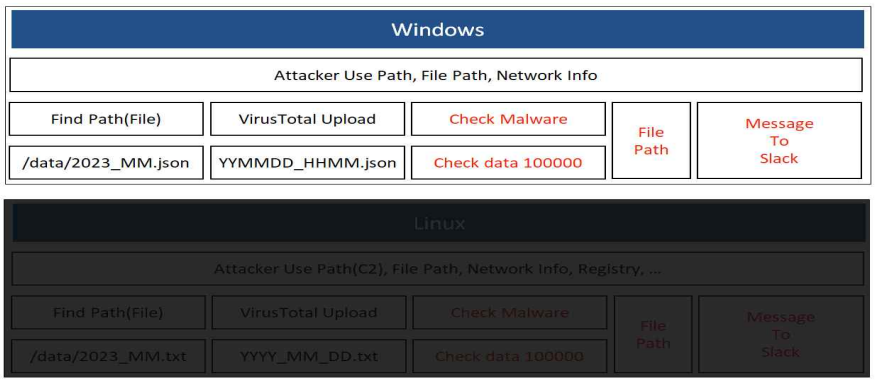
서비스 요약

침해사고 대응 알림 서비스 서비스 요약



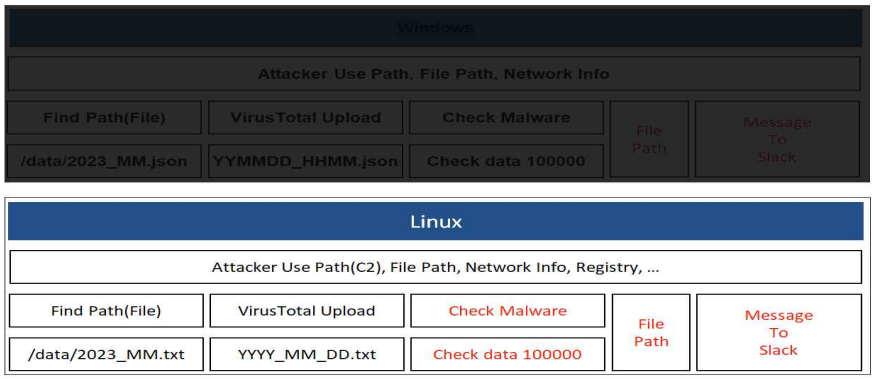
5

서비스 요약



6

서비스 요약

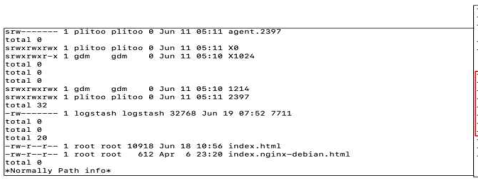


7

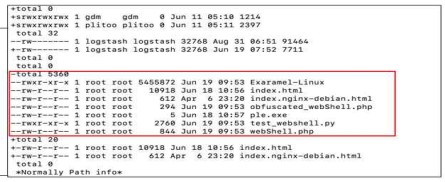
위협 탐지 시나리오 - 리눅스

1.

- 매월 1일 정보들을 수집하여 YYYYMM_main.txt로 저장
- MIRTE ATT&CK와 보안 회사 리포트를 기반으로 리눅스 정보 수집
- C2 서버로 사용될 경우, 기본 경로, 네트워크 정보, 파일 시스템 등
- YYYYMM_main과 YYYYMMDD_HHMM 파일을 비교하여 /diff_date/YYYYMM_HHMM 저장



date/2023MM_main.txt



data/2023MMDD_HHMM.txt

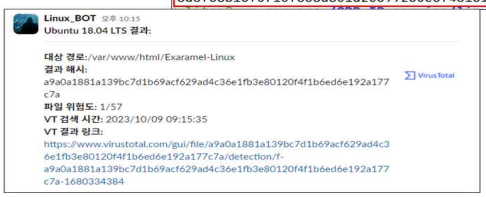
8

위협 탐지 시나리오 - 리눅스

2.

- /diff_date/YYYYMM_HHMM 에서 1000000이상의 파일이 있다면 Hash로 변환한다.
- 변환한 정보는 /diff_hash/filename.hash 로 저장
- 해당 해시를 VirusTotal API를 이용하여 업로드 후 검사

```
plitoo@grr_server:~/GRR_IR_service/linux/diff_hash$ ls -a
.  ..  Exaramel-Linux.hash  hermez.hash
plitoo@grr_server:~/GRR_IR_service/linux/diff_hash$ cat Exaramel-Linux.hash hermez.hash
a9a0a1881a139bc7d1b69acf629ad4c36e1fb3e80120f4f1b6ed6e192a177c7a
8d3f68b16f0710f858d8c1d2c699260e6f43161a5510abb0e7ba567bd72c965b
```



9

위협 탐지 시나리오 - 윈도우

10

위협 탐지 시나리오 - 윈도우

Slack Start 메시지

GRR 진단 powershell 스크립트
진단 시작 후 중간 내용 process.txt에 저장

```
1  Import-Module C:\PowerGRR-0.12.0\PowerGRR-0.12.0\PowerGRR.psd1 -force
2  $GRRCredential = Microsoft.PowerShell.Security\get-credential -UserName admin -Message "GRR 인증 password 입력중 기다리세요"
3      #컴퓨터이름 출력
4  $Client_ComputerName = hostname
5      #연동된 client를 모두 보는법(clientid만 출력하게 만듬.)
6  $ALL_Clientid = (Get-GRRClientIdFromComputerName $Client_ComputerName).Clientid
7      #클라이언트ID에 라벨 붙이는법(client host name별)
8  Set-GRRLabel -ComputerName $Client_ComputerName -Label Park
9      #클라이언트 ID를 라벨로 찾는법
10 $clients = Find-GRRClientByLabel -SearchString Park
11     #프로세스 헌팅 준비 + huntid만 출력 변수에 저장
12 $Huntid_system1 = New-GRRHunt -HuntDescription "file system path hash save1" -Flow FileFinder -RuleType Label -Label Park -ActionType Hash -Mode ALL_HITS -Path "C:\
13 $Huntid_system1.hunt_id > huntid.txt
14     #GRR헌트 시작
15 Start-GRRHunt -HuntId $Huntid_system1.hunt_id
```

GRR Client 배포

11

위협 탐지 시나리오 - 윈도우

```powershell
# 작업 액션을 생성합니다. 실행할 Python 스크립트 파일을 지정합니다.
$action = New-ScheduledTaskAction -Execute 'python.exe' -Argument 'C:\PowerGRR-0.12.0\PowerGRR-0.12.0\py_slackbot_test\ret_grr.py'

# 작업 트리거를 생성합니다. 현재 실행 시간에서 30분 후에 시작하도록 설정합니다.
$trigger = New-ScheduledTaskTrigger -Once -At ($currentTime.AddMinutes(30))

# 작업을 등록합니다. 작업 이름과 트리거를 지정합니다.
Register-ScheduledTask -TaskName 'grr_ret' -Trigger $trigger -Action $action

# 작업에 설명을 추가합니다.
$task = Get-ScheduledTask -TaskName 'grr_ret'
$task.Description = "grr 결과값 받아오는 작업 실행중"

# 작업 트리거를 수정하여 30분 간격으로 반복하도록 설정합니다.
$trigger.RepetitionInterval = "PT30M"  # 30분 간격
```

GRR 진단 결과 자동업로드를 위해
스케줄러에 등록

```python
# PowerShell 스크립트를 생성하고 파일로 저장
ps_script = f"""
Import-Module C:\\PowerGRR-0.12.0\\PowerGRR-0.12.0\\PowerGRR.psd1 -force;
$GRRCredential = Microsoft.PowerShell.Security\\get-credential -UserName admin -Message 'GRR 인증 password 입력창 기다리세
Get-GRRHuntResult -HuntId {huntid}_system1 -ShowJSON > C:\PowerGRR-0.12.0\PowerGRR-0.12.0\py_slackbot_test\json_ret\tes
"""

# 스크립트를 파일로 저장
with open("C:\\PowerGRR-0.12.0\\PowerGRR-0.12.0\\ret_grr.ps1", "w") as script_file:
    script_file.write(ps_script)

# PowerShell 스크립트 실행
ps_script_path = "C:\\PowerGRR-0.12.0\\PowerGRR-0.12.0\\ret_grr.ps1"
username = "admin"
password = "park1004"

process = subprocess.Popen(
    ['powershell', '-ExecutionPolicy', 'Bypass', '-File', ps_script_path],
    stdin=subprocess.PIPE,
    stdout=subprocess.PIPE,
    stderr=subprocess.PIPE,
    text=True
)
```

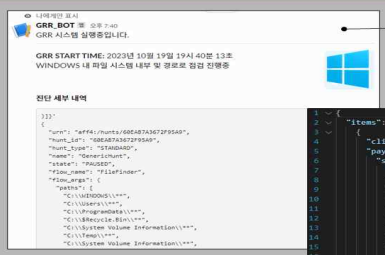
스케줄러 등록된 실행 .py 파일
결과 값을 json형태로 가져온다.

```python
# 각 파일에서 hash 값을 추출합니다.
hash_set1 = set(entry["payload"]["hash_entry"]["sha256"] for entry in data1["items"])
hash_set2 = set(entry["payload"]["hash_entry"]["sha256"] for entry in data2["items"])

# 두 세트를 비교하여 중복되지 않는 항목을 찾습니다.
unique_hashes = hash_set1.symmetric_difference(hash_set2)

# 중복되지 않는 hash 값을 가진 항목을 리스트로 저장합니다.
unique_entries = []

# index를 1부터 시작하도록 초기화합니다.
index = 1

for entry in data1["items"] + data2["items"]:
    if entry["payload"]["hash_entry"]["sha256"] in unique_hashes:
        sha256 = entry["payload"]["hash_entry"]["sha256"]
        sha1 = entry["payload"]["hash_entry"]["sha1"]
        md5 = entry["payload"]["hash_entry"]["md5"]
        path = entry["payload"]["stat_entry"]["pathspec"]["path"]
        timestamp = entry["timestamp"]
        unique_entry = {
            "index": index,
            "sha256": sha256,
            "sha1": sha1,
            "md5": md5,
            "path": path,
            "timestamp": timestamp
        }
        unique_entries.append(unique_entry)
        # 다음 항목을 위해 index를 증가시킵니다.
        index += 1
```

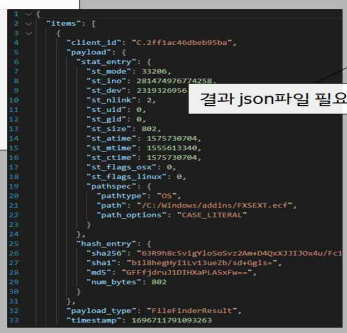
결과값 json 파일
초기 원본파일과 대조 추가된 값 추출
해쉬값 추출 및 정재

GRR 진단 원격 실행

위협 탐지 시나리오 - 윈도우



Slack상 진행사항 과정



```
{
  "items": [
    {
      "client_id": "C.2ff1ac46dbeb95ba",
      "payload": {
        "stat_entry": {
          "st_mode": 33206,
          "st_ino": 2814749767742258,
          "st_dev": 2319326056,
          "st_nlink": 2,
          "st_uid": 0,
          "st_gid": 0,
          "st_size": 802,
          "st_atime": 1575730704,
          "st_mtime": 1555613340,
          "st_ctime": 1575730704,
          "st_flags_osx": 0,
          "st_flags_linux": 0,
          "pathspec": {
            "pathtype": "OS",
            "path": "/C:/Windows/addins/FXSEXT.ecf",
            "path_options": "CASE_LITERAL"
          }
        },
        "hash_entry": {
          "sha256": "63R9h8c5vigVloSoSvz2Am+D4QxXJJIJOx4u/Fc1
          "sha1": "b118hegHyI1Lvi3ue2b/sd+Ggls=",
          "md5": "GfFjdru3IDIHXaPLASxFw==",
          "num_bytes": 802
        }
      },
      "payload_type": "FileFinderResult",
      "timestamp": 1696711791093263
```

결과 json파일 필요값만 추출

```
{
    {
        "index": 1,
        "sha1": "gk5sg8B5BzE5IRzI531As3IVbSc=",
        "sha256": "V6Annjtr FYQEz79QcOM7QZY+KIp7sjA4Kx0/YRYpnHU=",
        "md5": "ggzXQp58U6dDc1s3aAfuaQ==",
        "path": "/C:/Windows/bfsvc.exe",
        "timestamp": 1696732898053408
    },
    {
        "index": 2,
        "sha1": "A/eB5V2PTW3YirIIhR8OKcziQdk=",
        "sha256": "7iVE42tmq0ZaQxFthd6SeX2tduFVOtN9dBGT6yQL6x8=",
        "md5": "4zpSVsdvaXqkYZid3FwQHg==",
        "path": "/C:/Windows/bootstat.dat",
        "timestamp": 1696732898053485
    },
```

JSON

진단 내역 JSON 파일로 정제 후 저장

위협 탐지 시나리오 - 윈도우

```python
entry = unique_entries[0]
sha256_hash = entry["sha256"]
print(sha256_hash)
file_path = entry["path"]

scan_url = f"https://www.virustotal.com/api/v3/files/{sha256_hash}"
headers = {
    "x-apikey": api_key
}

response = requests.get(scan_url, headers=headers)
```

해쉬값을 이용 V.T 스캔

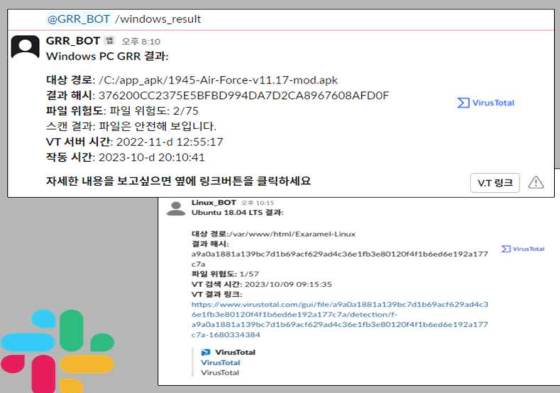
```python
if total_engines > 0:
    file_risk = f"파일 위험도: {malicious_engines}/{total_engines}"
else:
    file_risk = "파일 위험도: 알 수 없음"

formatted_date = korea_time.strftime("%Y-%m-d %H:%M:%S")

current_time = datetime.now()
formatted_time = current_time.strftime("%Y-%m-d %H:%M:%S")

file_status = scan_result["data"]["attributes"]["last_analysis_stats"]["harmless"]
if file_status > 0:
    status_message = "스캔 결과: 악성 파일일 수 있습니다. 관리자에게 문의하세요."
else:
    status_message = "스캔 결과: 안전한 파일입니다."

vt_link = f"VirusTotal 파일 정보 링크: https://www.virustotal.com/gui/file/{sha256_h
```

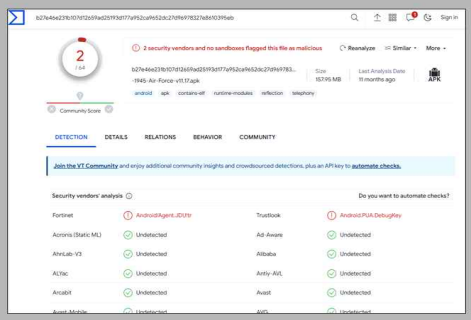
VirusTotal

위험도, 스캔 시간, 자세한 링크로
스캔결과를 가져옴

바이러스 토탈 연동 및 스캔

위협 탐지 시나리오 - 윈도우

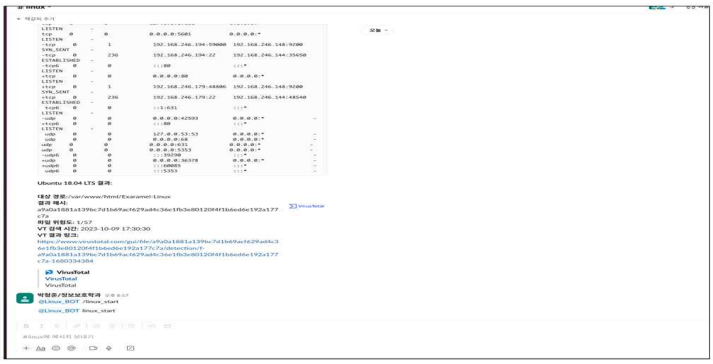




진단 결과 Slack 전송

15

시연 영상 - 리눅스



16

시연 영상 - 윈도우

17

결론 및 기대 효과

결론

- 유료 도구들을 사용하지 않고 서버들의 위협을 탐지하는 서비스를 개발하였다.
- 불필요한 정보들은 제외하며 관리자가 원하는 정보만 수집하였다.
- 모든 사항을 Slack으로 확인함으로 사용자 친화적으로 확인할 수 있다.

기대 효과

- 값비싼 솔루션을 사용하지 못 하는 기업들의 비용을 아낄 수 있다.
- APT 그룹들의 목표를 가기 위한 C2 서버와 Host에 대한 위험 부담을 줄일 수 있다.

18

감사합니다.

목표75KG

6.3 소개 자료

2023년 중부대학교 정보보호학과 졸업작품 전시회

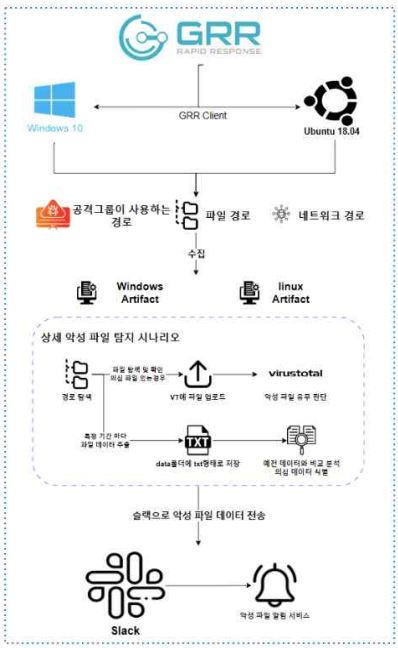
침해사고 대응 알림 서비스

Infringement Incident Response Notification Service

✓개요

대부분의 기업은 운영체제의 최신버전을 쓰지않는다.
이는 보안 위협에 쉽게 노출될 수 있으며, 해커들의 공격은 진화하고 발전하고 있다.
GRR과 **Slack**을 효과적으로 연동하여 실시간 의사소통 및 협업을 강화하고
실시간으로 시스템 및 네트워크 상태를 감시하고 대응이 가능한 서비스를 기획하였다.

✓서비스 구상도



✓서비스 개발내용

박형준

은정욱

김두형