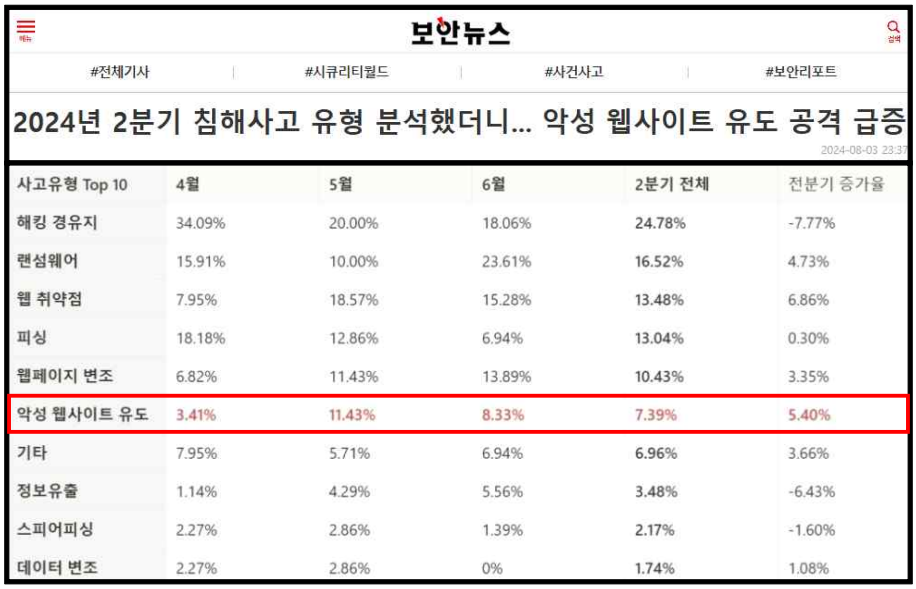
DU-도전학기 결과보고서

과제명	시각장애인용 스크린리더 프로그램 제작		
참여자	성명	소속	학번
		사이버보호전공	
		사이버보호전공	
지도교수 의견	두 참여 학생이 적극적으로 DU-도전학기에서 제안한 주제를 성실히 수행하고 있는 것으로 확인되었으며, 이전에 구현과정에서 발견한 문제점을 잘 해결했고, 사이버 보안 적 측면에서도 피싱 사이트에 대한 탐지가 가능한 것으로 보아 주제에 적합한 장애인용 스크린리더 프로그램이 개발된 것 같습니다. (소속) 사이버보안전공 (성명)		

1. 도전 과제 내용

서울 여의도에 사는 시각장애인 절친 역시 3번이나 대리점을 방문했지만, 유심은 교체하지 못하였다고 한다. 그 역시 토스와 카카오뱅크를 이용하는데 정보가 유출될까 봐 늘 걱정이고 최대한 빨리 유심 교체를 하기 위해 알아봤지만, 방법이 없었다고 한다.

(그림 1) SKT 개인정보 유출사태 , 더욱 힘겨운 시각장애인들 [에이블 뉴스 제공]



보안뉴스

#전체기사 #시큐리티월드 #사건사고 #보안리포트

2024년 2분기 침해사고 유형 분석했더니... 악성 웹사이트 유도 공격 급증

2024-08-03 23:37

사고유형 Top 10	4월	5월	6월	2분기 전체	전분기 증가율
해킹 경유지	34.09%	20.00%	18.06%	24.78%	-7.77%
랜섬웨어	15.91%	10.00%	23.61%	16.52%	4.73%
웹 취약점	7.95%	18.57%	15.28%	13.48%	6.86%
피싱	18.18%	12.86%	6.94%	13.04%	0.30%
웹페이지 변조	6.82%	11.43%	13.89%	10.43%	3.35%
악성 웹사이트 유도	3.41%	11.43%	8.33%	7.39%	5.40%
기타	7.95%	5.71%	6.94%	6.96%	3.66%
정보유출	1.14%	4.29%	5.56%	3.48%	-6.43%
스피어피싱	2.27%	2.86%	1.39%	2.17%	-1.60%
데이터 변조	2.27%	2.86%	0%	1.74%	1.08%

(그림 2) 2024년 2분기 침해사고 유형 [한국인터넷진흥원 제공]

위의 기사 및 도표를 확인해보면 악성 웹사이트 유도 침해사고가 2024년도 2분기에 급증했다는 뉴스를 보고 이런 문제점을 어떤식으로 해결 해 볼 수 있을까 고민을 하게 됐다.

과거에는 악성 파일을 통한 악성 스크립트 삽입으로 불법 사이트로 리다이렉션 하는 공격이 성행 했지만, 현재는 호스팅 관리자 계정을 탈취 후 DNS 설정으로 불법 사이트로 리다이렉션, 악성스크립트 삽입 등의 행위로 공격이 이루어지고 있다.

이 보고서에서는 현재 온라인 광고를 악용하는 멀버타이징 공격을 통해 사용자를 위협한다는 것을 중점으로 다룬다.

여기서 멀버타이징이란 구글 등 인터넷 브라우저에 일정 금액을 제공하면 스폰서의 URL이 제일 위에 노출되는데 이는 일반인뿐 아니라 시각장애인에는 큰 해가 될 것으로 예상되어 해당 공격을 방지 하기 위해 악성 웹사이트로 인한 피해를 줄이기 위해 브라우저의 URL을 탐지해 정상 사이트인지 악성사이트인지 판별 하는 프로그램을 제작하기로 했다.

2. 도전 과제 수행 결과 및 성과

머신러닝 모델을 활용해 악성 URL을 판별하기 위해 모델을 계획 및 실험을

모두 마친 상태에서의 결과이다.

```
RF Accuracy: 0.9246814213679074
              precision    recall  f1-score   support

      benign       0.94      0.97      0.95    103722
   malicious       0.87      0.79      0.83     31331

    accuracy                           0.92    135053
   macro avg       0.91      0.88      0.89    135053
weighted avg       0.92      0.92      0.92    135053

Saved rf_model_tuned.pkl
```

전ㅈ체적인 결과만 보면 정확도는 약 92.4%로 양호해 보인다. 하지만 이것은 해당 데이터에 대한 정확도이다.

이 말은 학습용 데이터 셋에 비슷한 패턴을 가진 URL이 존재해 이것을 학습 데이터로 삼고 출력한 결과로 예측 해 볼 수 있다. 그리고, reacall을 확인해 보면 악성을 79% 확률로 맞춘다고 되어있다. 하지만, 악성을 정상이라고 판단 후 사용자가 이 페이지에 접속하게 되면 치명적인 피해를 끼질 수 있다.

이러한 연구결과를 토대로 본래의 머신러닝 모델은 제거하고 가장 확실한 방법인 화이트/블랙 리스트 기법을 사용해 피싱사이트의 접근을 막기로 했다. 또한, URL을 실시간 검사해주는 API를 추가적으로 기용해, 리스트에 없는 사이트라도 판별이 가능하다는 이점이 있으며, 이를 로컬 저장소에 추가하기 때문에 훨씬 안전하고 확실한 방지법이 될 수 있기 때문이다.

이 프로그램은 Chrome 브라우저를 대상으로 정했다.

확장 프로그램과 유저의 URL 통신 결과를 보여주기 위해 python3 로 만든 간이 스크린 리더 프로그램과 직접제작한 chrome 확장 프로그램에 연결하게 된다.
Chrome 확장 프로그램은 웹 요청 완료 시점을 후킹해 URL을 수집하고,
이것을 Localhost 8080으로 전송한다.

python 측에서는 수신 가능한 REST 서버가 BaseHTTPRequestHandler 기반으로 구동되어 브라우저에 서 보낸 요청(JSON 형태의 URL)를 수신하게 된다.

서버 수신시 json.loads()를 통해 요청을 파싱하고 URL 키를 기준으로 도메인 정보를 추출한 후, 로컬의 화이트/블랙 리스트를 경유하고 VirusTotal API로 교차 확인 후, 최종적으로 피싱으로 판정 될 경우 스크린리더에서 즉시 경고음이 출력된다.

화이트, 블랙리스트에 사용자가 접속 예정인 도메인의 정보가 없다면 Virustotal API를 사용해 약 96개의 백신 엔진에서 10개 이상의 백신 엔진이 피싱으로 결과를 반환하게 된다면, 해당 사이트는 피싱 의심 사이트로 구분됨

Virtustotal에서 정상으로 판단한 도메인은 구글 확장 프로그램에서 실시간으로 정상 리스트에 추가 및 로컬 저장소에 해당 주소를 저장하게 된다.

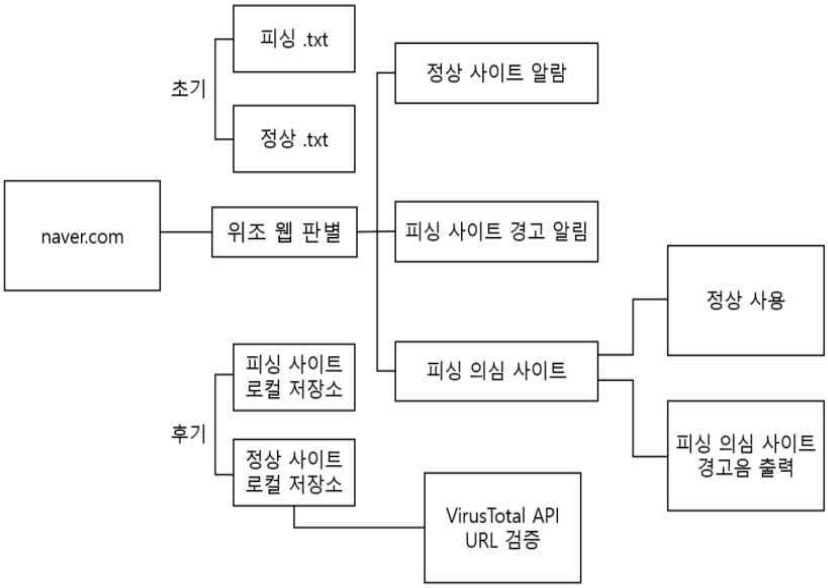
※ 정상 도메인 수집 목록 ※

- o Kaggle > 100만개 가량의 도메인 수집 (방문자 순서)
- o 한국은행 > 국내 사이트 금융 기관 도메인 수집
- o 학교알리미 > 전국 초, 중, 고등학교 도메인 수집
- o LX 한국국토정보공사 > 정부부처 및 공공기관, 지방자치단체 도메인 수집
- o 농림축산식품부 > 정부 기관 관련 도메인 수집
- o 공공데이터포털 > 전국 대학 및 전문 대학 정보 표준 데이터 도메인 수집

※ 악성 피싱 도메인 수집 목록 ※

- o 공공데이터포털 > 한국인터넷진흥원 피싱 사이트 URL 도메인 수집
- o Github > JPCERT CC/phishurl-list 피싱 사이트 도메인 수집
- o PhishTank, URLhaus 피싱 사이트 도메인 수집

해당 도면은 전체적으로 브라우저 환경에서 작동하는 방식을 설명하는 부분이다.



스크린 리더에는 다음과 기능이 탑제되어 있음

1. F2 키 - 현재 입력한 텍스트 읽기

```
def on_f2_press(event):
    if not features["전체 텍스트 읽기 (F2)"].value: return
    if event.event_type == 'down':
        keyboard.press_and_release('ctrl+a')
        time.sleep(0.05)
        keyboard.press_and_release('ctrl+c')
        time.sleep(0.05)
        keyboard.press_and_release('right')
        text = get_clipboard_text()
        if text:
            safe_speak_text(text, get_current_rate())
```

기존 다른 스크린리더에 있는 기능은 입력하고 있는 텍스트를 읽어주는데 이는 현재 치고있는 문장이 무엇인지 알아듣기 힘들기때문에 F2를 누르면 사용자가 작성하고 있는 글을 전체를 검토할수있는 기능을 탑재하였음.

2. Caps Lock 키: 드래그한 문장 읽기

```
def on_capslock_press(event):
    if not features["선택 텍스트 읽기 (Caps Lock)"].value:
        return
    if event.event_type == 'down':
        keyboard.press_and_release('ctrl+c')
        time.sleep(0.1)
        text = get_clipboard_text()
        if text.strip():
            safe_speak_text(text, get_current_rate())
```

특정 문장이나 단어만 듣고싶을때 사용하는 기능으로 사용자가 원하는 부분

만 골라서 들을수 있도록 하였음.

3. Ctrl 키: 낭독 중지

```
def interrupt_current_speech(show_log=True):
    global current_process, space_process
    for proc in [current_process, space_process]:
        if proc and proc.is_alive():
            proc.terminate()
            proc.join()
```

```
def start_keyboard_hooks():
    keyboard.hook_key('ctrl', lambda e: interrupt_current_speech() if e.event_type == 'down' and features["⎋ 음성 낭독 중단 (Ctrl)"].value else None)
```

위 F2나 Caps Lock키를 직접 사용해보니 실수로 사용하여 긴 문장을 읽는경우가 있어 낭독을 중단하는 기능이 필요하다고 판단하여 추가하였음.

4. 키패드 + / - : 음성 속도 조절

```
def on_plus(event):
    if not features["|▸▸| 낭독 속도 조절 (+ / -)"].value: return
    if event.event_type == 'down' and current_speed_index.value < len(speed_levels) - 1:
        current_speed_index.value += 1
        new_speed = speed_levels[current_speed_index.value]
        safe_speak_text(f"{new_speed:.1f} 배속", rate=180)

def on_minus(event):
    if not features["|▸▸| 낭독 속도 조절 (+ / -)"].value: return
    if event.event_type == 'down' and current_speed_index.value > 0:
        current_speed_index.value -= 1
        new_speed = speed_levels[current_speed_index.value]
        safe_speak_text(f"{new_speed:.1f} 배속", rate=180)
```

사용자마다 문장을 이해하는 능력이 다르기때문에 음성 속도가 빠르고 느릴수도 있다 생각하여 속도 조절을 할 수 있는 기능을 추가하였음.

6. 스페이바 읽기

```
def on_space_press(event):
    if not features["□ 스페이스 키 읽기"].value: return
    if event.event_type == 'down':
        safe_speak_text("스페이스", 300)
```

스페이스바는 타이핑중에 가장 많이 눌리고 마우스를 가져다대거나 F2, Caps Lock 으로 텍스트를 읽는 기능은 띄어쓰기를 알려주지않기때문에 스페

이스바를 눌렀을때 사용자에게 알려야한다고 판단하였음. 또한 텍스트를 입력할때 가장 많이 사용되기때문에 음성으로 출력된느 속도가 빨라야한다고 판단하여서 빠르게 설정하였음.

7. 전체 UI 개요

```
def start_ui():
    root = tk.Tk()
    root.title("🔊 스크린리더 기능 제어")
    root.geometry("600x520")
    root.configure(bg="#f0f0f0")

    menubar = tk.Menu(root)
    menubar.add_command(label="파일")
    menubar.add_command(label="도움말")
    root.config(menu=menubar)

    feature_frame = tk.LabelFrame(root, text="사용 가능한 기능들", padx=10, pady=10, bg="#f0f0f0")
    feature_frame.pack(padx=10, pady=10, fill="x")

    for idx, (text, val) in enumerate(features.items()):
        var = tk.BooleanVar(value=val.value)
        cb = tk.Checkbutton(feature_frame, text=text, font=("맑은 고딕", 10), variable=var,
                            bg="#f0f0f0", anchor="w", command=toggle_feature_wrapper(text, var))
        cb.grid(row=idx, column=0, sticky="w", pady=2)

    entry = tk.Entry(root, font=("맑은 고딕", 10), width=50)
    entry.pack(pady=(5, 5))

    output_box = tk.Text(root, height=10, font=("맑은 고딕", 10))
    output_box.insert(tk.END, "[스크린리더 시작됨]")
    output_box.pack(padx=10, pady=(5, 5), fill="x")

    root.mainloop()
```

기본적으로 UI크기는 600x520으로 크지도않고 작지도않은 적당한 크기로 형성.
추후에 메뉴나 도움말의 기능을 추가할 여지를 두는 의미에서 상단에 메뉴바를 만들었음.
전체 텍스트 읽기 (F2)
선택 텍스트 읽기 (Caps Lock)
음성 낭독 중단 (Ctrl)
낭독 속도 조절 (+ / -)
스페이스 키 읽기
마우스 위치 텍스트 자동 읽기
다음과 은같 기능을 체크할수있는 체크박스를 만들어 on/off를 할 수 있게 하였음.
아래에 텍스트를 입력할수있는 공간을 만들어 스크린리더의 기능을 테스트 할 수 있음.

3. 자기 평가

: 도전학기를 시작하기 전에 베타 버전으로 만들어본 스크린리더는 구현하기에 생각보다 어렵지 않아 완벽한 프로그램 하나를 3개월이라는 시간안에 제작 할 수 있을것이라고 생각했지만, 막상 Windows api 에 대해 공부를 하고 코딩을 했을때는 생각만큼 쉽지않아 스크린리더 프로그램을 완벽하게 구현은 못했지만, 구색은 갖춰진거 같아 만족한다. 보안 프로그램에 대한 부분은 머신러닝을 사용해 판별하는 것이 사실상 불가능에 가까워 더욱 더 합리적인 방안을 찾고 원하는 방향대로 개발이 됐다.

: 도전학기 초기때 계획한 스크린리더 기능을 여러가지 개발하기로 하였는데 현재 개발한 기능은 F2 키를 누르면 현재 입력 중이거나 선택된 텍스트를 실시간으로 읽어주는 기능을 구현하였고
이는 내부적으로 Ctrl+A, Ctrl+C를 자동으로 조합하여 텍스트를 복사하고 읽어주는 방식으로 설계하였습니다. 또한 사용 중 갑작스럽게 읽기를 중단해야 하는 경우를 대비하여 Ctrl 키를 누르면 즉시 음성 출력을 멈추는 기능을 구현하였습니다.
그리고 사용자별로 음성 속도가 늦은사람도 있을것이며 빠른사람도 있을것이라 판단하여 음성 출력 속도를 조절할 수 있는 기능도 추가하였습니다. 숫자 키패드의 '+' 또는 '-' 키를 통해 실시간으로 속도를 높이거나 낮출 수 있으며 현재 속도 상태는 음성으로 피드백됩니다.
또 특정 텍스트만 읽게하고싶은 경우를 대비하여 Caps Lock 키를 누르면 마우스로 드래그한 특정 문장을 읽는 기능을 개발하였으며 텍스트 입력을 할때 스페이스바는 가장 많이쓰이지만 다른 기능으로는 띄어쓰기를 읽어주지 않음으로
스페이스바를 누르면 알려주는 기능도 함께 개발하였습니다. 이러한 기능들은 시각장애인들이 문서 작업, 웹 탐색, 학습 등 다양한 환경에서 사용할 수 있도록 하였습니다.
또한 프로젝트 후반에는 전체 시스템을 직관적으로 조작할 수 있는 사용자 인터페이스를 제작하였습니다. 단순한 체크박스와 토글 버튼을 통해 주요 기능을 켜고 끌 수 있도록 하여 사용자가 원하는 기능만 키고 끌수 있게 하였습니다.
이와 같은 기능을 하나하나 구현하고 통합하는 과정에서 다양한 문제를 마주했지만 끊임없는 시행착오를 걸치며 해결하며 기술적 성장을 경험할 수 있었습니다. 결과적으로 본 프로젝트는 아직 시각장애인이 사용하기에 불편한점이 있을수도 있지만
시각장애인들에게 최대한 배포하려고 노력하여 피드백을 받아 불편한점이나 추가할점을 개선해나갈것입니다. 도전학기 초반에 세운 목표를 성공적으로 달성하

였으며 이 과정은 저에게 큰 성취감을 안겨주었으며 소프트웨어 개발의 전 과정을 경험할 수 있는 소중한 시간이었습니다.

4. 최종 결과물

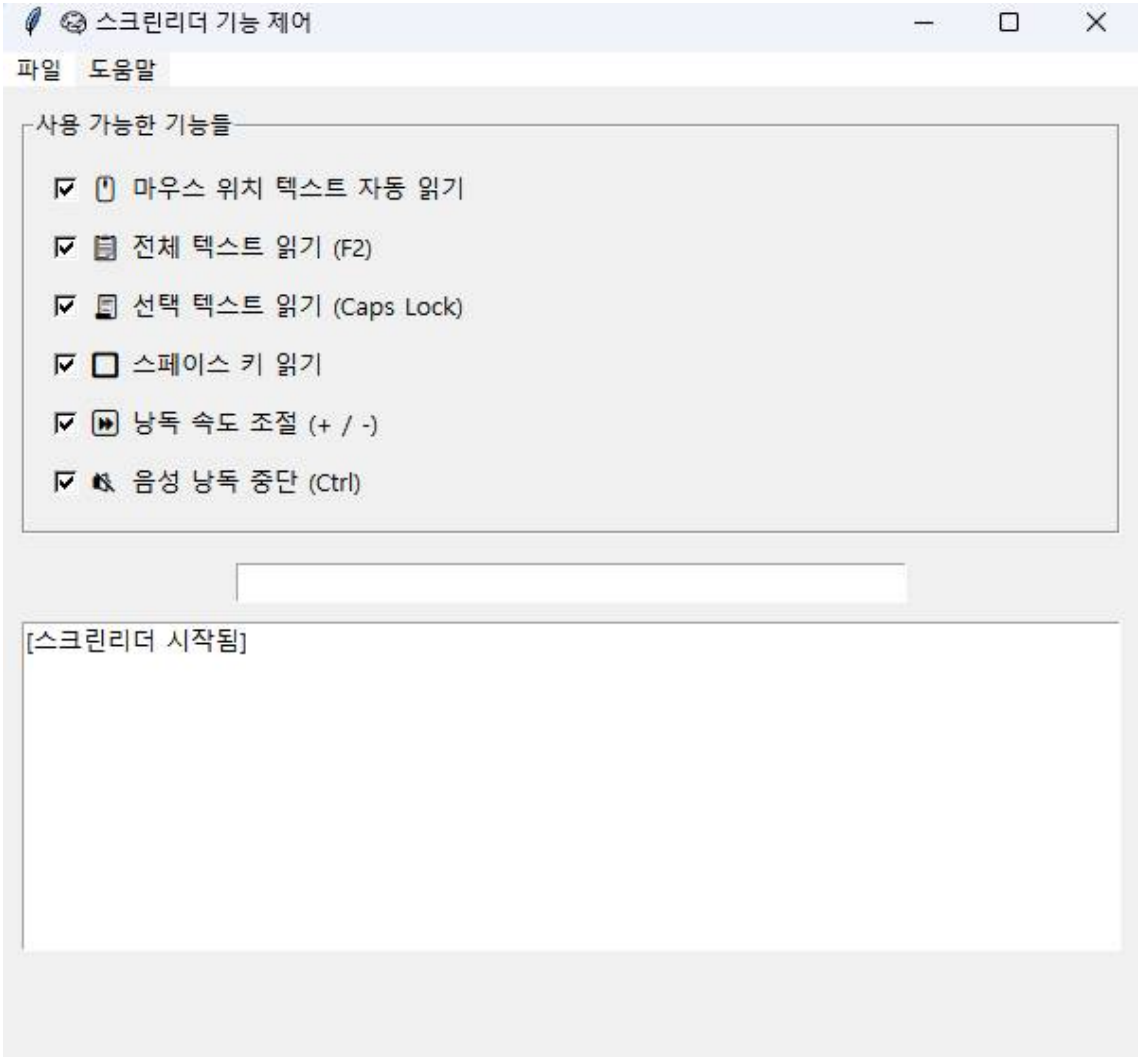


그림1 https://www.ablenews.co.kr/news/articleView.html?idxno=221005
그림2 1https://www.boannews.com/media/view.asp?idx=131817