

CTI STIX Visualization



TEAM: ICEWALL

MEMBER: 문서진, 문경태, 이수민, 임한섭, 최서연, 최성민, 한정현

목차

1. 프로젝트 소개	3
(1) 프로젝트 배경	3
(2) 프로젝트 목적	3
(3) 기대 효과	3
2. 프로젝트 사전 조사	4
(1) STIX 란 무엇인가?	4
(2) STIX 의 구조	4
3. 프로젝트 내용	6
(1) 기존 웹사이트 분석	6
1) 기능 설명	7
2) 변경할 점	8
(2) 웹사이트 변경 사항	13
4. 결론	28
(1) 활용 가능성	28
(2) 시사점 및 향후 프로젝트	28

1. 프로젝트 소개

(1) 프로젝트 배경

2023년 3월부터 시작된 멀웨어 캠페인이 현재까지 진행되고 있으며 북미, 남미, 유럽, 일본에서 40개 은행, 5만 명 사용자가 피해를 보았다. 지속해서 보안에 관심을 갖고 보안 기술을 업데이트 하였는데도 침해사고가 일어나는 이유는 공격 방식이 다양화되고 해킹 방지 우회 기술 등이 빠르게 생겨나는 것 때문이라고 볼 수 있다. 그렇지만 해당 사고에 대한 정보가 빠르게 공유되었다면 같은 공격에 이렇게 많이, 오랫동안 피해가 발생하진 않았을 것이다. 따라서 침해사고가 일어났을 때 해당 사고에 대한 정보를 신속히 공유할 수 있는 구조화된 사이버 위협 정보 표현인 STIX에 관심을 갖게 되었다.

(2) 프로젝트 목적

모든 사람이 이해하기 쉽게 하기 위해 STIX를 가시화하는 프로그램을 만드는 것이 초기 목적이었다. 하지만 초기 목적에 맞는 플랫폼과 웹 사이트가 이미 존재하기에 STIX 가시화 웹사이트인 STIX Visualizer를 사용자들이 더 편리하게 사용할 수 있도록 디자인을 개선하고 기능을 추가하는 것으로 목적을 변경하였다.

(3) 기대 효과

- 1) 기존 웹페이지에 선별 기능을 추가해서 원하는 데이터만 뽑아서 볼 수 있다.
- 2) 디자인을 수정함으로써 가독성을 높이고 편리하게 사용할 수 있다.
- 3) STIX가 생겨난 배경을 이해하고 STIX의 SDO, SRO의 종류와 의미 등을 정리함으로써 STIX에 대한 이해도를 높인다.
- 4) 마이데이터나 피싱 범죄로 인해 정보 공유에 대한 중요성이 커지고 있다. STIX는 정보 공유 체계 중 하나로서 STIX 프로젝트 경험은 다른 정보 공유 체계를 사용하거나 개발할 때 도움이 될 것이다.

2. 프로젝트 사전 조사

(1) STIX란 무엇인가?

STIX(Structured Threat Information eXpression)는 사이버 보안 생태계에서의 표준화된 위협 정보 교환을 지원하기 위한 국제 표준이다. 사이버 보안 커뮤니티 간의 통일된 언어와 데이터 교환 형식을 제공하여 보안 전문가들이 효과적으로 협업하고, 사이버 위협에 대한 통찰력을 공유할 수 있도록 한다.

STIX는 XML 및 JSON을 기반으로 하여 사이버 위협 정보를 표현한다. 이것은 다양한 위협을 분석하고 이해하기 위해 고안된 고급 분석 도구 및 자동화 시스템에서 쉽게 활용된다. STIX의 핵심 구성 요소로는 Threat Actor(위협 가해자), Attack Pattern(공격 패턴), Tactics & Techniques & Procedures(TTPs, 전략/기술/절차), 보안 취약점 등이다. 이러한 구조는 보다 체계적이고 포괄적인 위협 모델을 구현할 수 있다.

(2) STIX의 구조

STIX의 전체적인 구조는 다음과 같다. 크게 STIX Core Objects와 STIX Meta Objects(SMO)로 나뉘고, 각각의 객체 내엔 세부적인 객체가 존재한다.

STIX Objects						STIX Bundle Object
STIX Core Objects			STIX Meta Objects (SMO)			
STIX Domain Objects (SDO)	STIX Cyber-observable Objects (SCO)	STIX Relationship Objects (SRO)	Extension Definition Objects	Language Content Objects	Marking Definition Objects	

Table 1 STIX 구조

각 객체에 대한 간단한 설명은 다음과 같다.

1) STIX Core Objects (SDO, SCO, or SRO)

- STIX Core Objects는 SDO(STIX Domain Object), SCO(STIX Cyber-observable Object), 또는 SRO(STIX Relationship Object)를 포함한다.

2) STIX Domain Objects

- 고수준 지능 객체로, 위협 분석가가 위협 환경을 이해하는 동안 일반적으로 생성하거나

작업할 것으로 예상되는 동작 및 구성을 나타낸다.

- Attack Pattern, Campaign, Course of Action, Grouping, Identity, Indicator, Infrastructure, Intrusion Set, Location, Malware, Malware Analysis, Note, Observed Data, Opinion, Report, Threat Actor, Tool, 및 Vulnerability 등의 객체가 존재한다. 각 객체는 CTI(Cyber Threat Intelligence)에서 일반적으로 사용되는 개념이다.

3) STIX Cyber-observable Objects

- 네트워크 또는 호스트의 관찰된 정보를 나타내는 객체로, 이러한 객체는 높은 수준의 지능과 함께 사용되어 위협 환경을 보다 완전히 이해하는 데 사용될 수 있다.
- STIX 사이버-관측 가능 객체(SCOs)는 네트워크 또는 호스트에서 발생한 사실을 문서화하며, 누가, 언제, 왜 했는지에 대한 정보는 포함하지 않는다. SCOs를 STIX 도메인 객체(SDO)와 연결하면 위협 환경에 대한 높은 수준의 이해를 전달할 수 있고, 특정한 누군가와 특정한 정보가 어떤 기관과 관련되었는지에 대한 여부도 파악할 수 있다.

4) STIX Relationship Objects

- SDO, SCO간의 객체를 연결하여 위협 환경을 완전히 이해하기 위한 객체이다.
- 일반적인 STIX 관계 객체(SRO)는 두 가지 SRO 중 하나로, STIX에서 대부분의 관계에 사용되고 있다. 이 일반적인 SRO에는 관계를 더 구체적으로 설명하는 relationship_type이라는 속성이 포함되어 있다.
- 현재 (일반적인 관계 외에) 다른 SRO는 Sighting SRO뿐이다. Sighting 객체는 SDO를 "보았을 때"의 경우에 사용되는데, 예를 들어, Indicator를 관찰한 경우에 해당된다. Sighting SRO는 Sighting 관계에만 해당되는 count와 같은 추가 속성을 포함하기 때문에 별도의 SRO로 구분된다.

5) STIX Meta Objects (SMO)

- SMO는 STIX Core Objects를 풍부하게 만들거나, 혹은 확장하여 사용자 및 시스템 작업 흐름을 지원하기 위한 필수적인 관련 메타데이터를 제공하는 STIX 객체이다.
- Extension Definition Objects, Language Content Objects, Marking Definition Objects가 포함되어 있으며, SMO는 공통 속성 중 일부만 사용하지만 모든 속성을 사용하지는 않는다.

6) STIX Bundle Object

- 임의의 STIX 콘텐츠를 함께 패키징하는 Wrapper 메커니즘을 제공하는 객체이다.

3. 프로젝트 내용

(1) 기존 웹사이트 분석

먼저 [기존 STIX Visualizer](#)에 대해 알아보고자 한다. 다음 사진은 기존 페이지의 사진이며, 밑의 문단에 기존 웹페이지에 대한 설명을 다루고 있다.

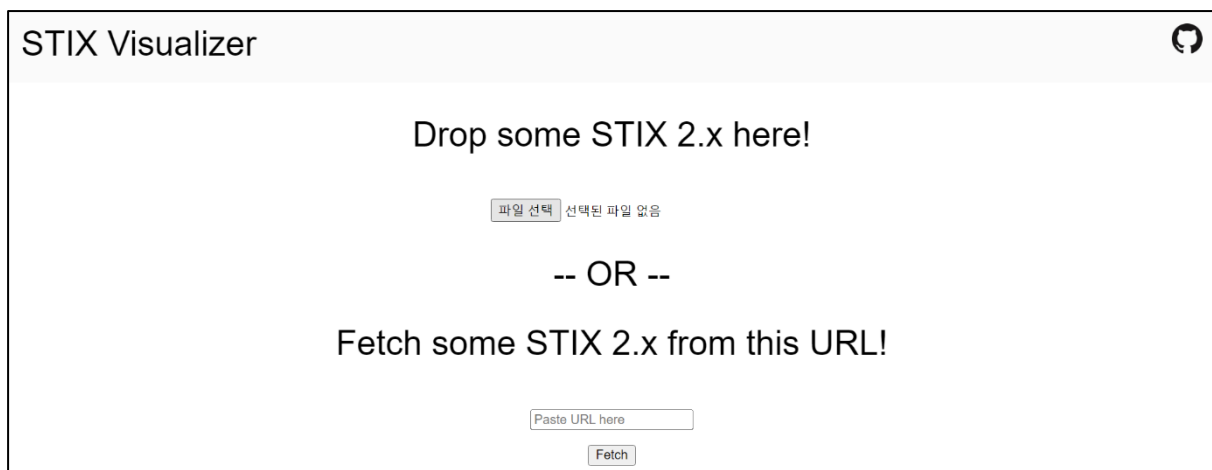


Figure 1 시작 화면

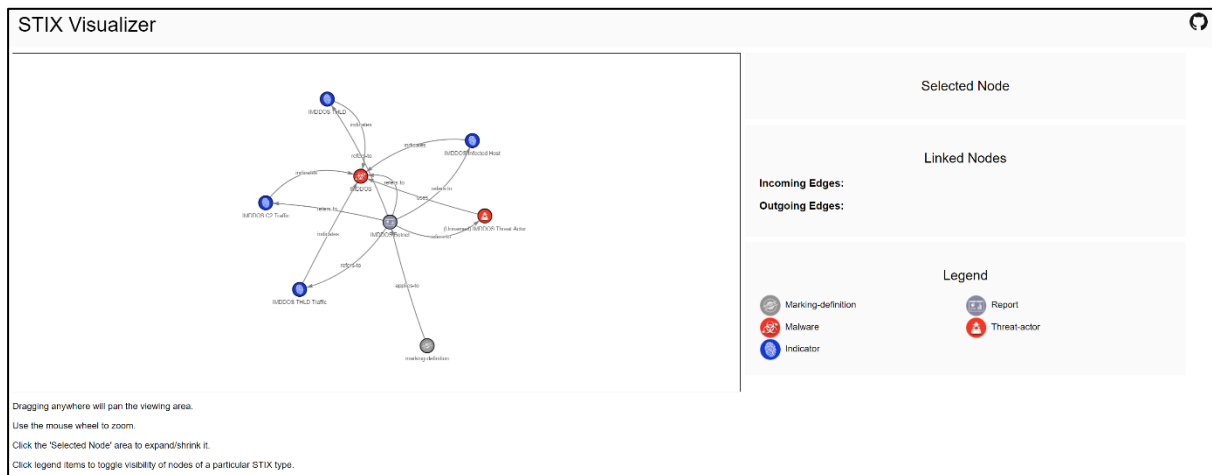


Figure 2 데이터 업로드 후 Visualizer 메인 화면

1) 기능 설명

기존 웹페이지에서 사용자는 다음과 같이 STIX 데이터에 포함된 정보를 가시화할 수 있다.

1. 시작 화면에서 파일/URL/JSON 텍스트 입력을 통해 STIX 데이터를 Visualizer에 올린다.
2. STIX 데이터의 객체 이름, 객체 간 관계 등이 그래프로 나타나며, 그래프 하단에 Visualizer 사용 방법이 표시된다.
 - a. 그래프에 포함되는 정보로는 객체 이름, 아이콘, 연결 관계가 있다.
3. 노드 선택 시
 - a. 측면의 Selected Node에 선택한 노드의 STIX 정보(type, id, created, modified, name, labels)가 표시된다.
 - b. 측면의 Linked Nodes에 간선으로 연결된 노드들을 확인할 수 있다.
 - i. 번호 옆 하이퍼링크된 객체 이름과 연결 관계가 표시된다.
 - ii. 객체 이름을 클릭하면 Selected Node, Linked Nodes가 해당 객체의 정보로 바뀐다.
 - iii. object_refs와 object_marking_refs의 경우 클릭하면 Selected Node, Linked Nodes가 해당 객체의 정보로 바뀐다.
 - iv. Incoming Edges의 경우 선택한 노드로 들어오는 방향의 간선과 연결된 노드의 정보를 확인할 수 있다.
 - v. Outgoing Edges의 경우 선택한 노드에서 나가는 방향의 간선과 연결된 노드의 정보를 확인할 수 있다.
4. 노드 미선택 시
 - a. 측면의 Selected Node, Linked Node에 아무 정보가 나타나지 않는다.
 - b. Legend의 각 항목을 클릭하면 해당 객체가 그래프에서 사라져, 확인하고 싶은 객체만 볼 수 있다.

2) 변경할 점

팀 내에서 논의된 추가 기능들은 다음과 같다.

1. 두 개 이상의 파일 업로드 기능

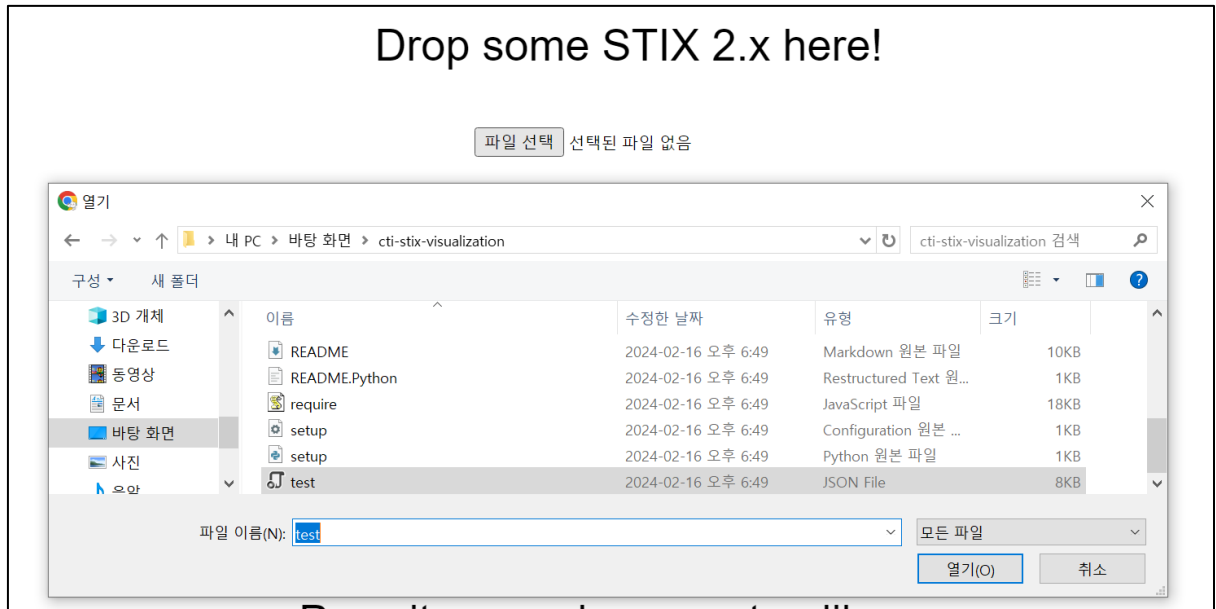


Figure 3 시작 화면의 '파일 선택' 클릭 후 업로드

시작 화면에서 하나의 STIX 데이터 파일 (.json)만을 올릴 수 있어, 한 번에 한 파일의 데이터만 시각화하여 확인할 수 있다. 연관성이 있는 STIX 데이터가 여러 개의 서로 다른 파일에 분산되어 저장되어 있는 경우에도 모두 한 번에 엮어서 시각화할 수 있도록, 두 개 이상의 파일 업로드 기능을 제안하였다.

2. SDO 검색 기능

STIX 2.0은 SDO(STIX Domain Object), SRO(STIX Relationship Object)로 구성되어 있으며, 12개 SDO와 2개 SRO로 구조화되어 있다.

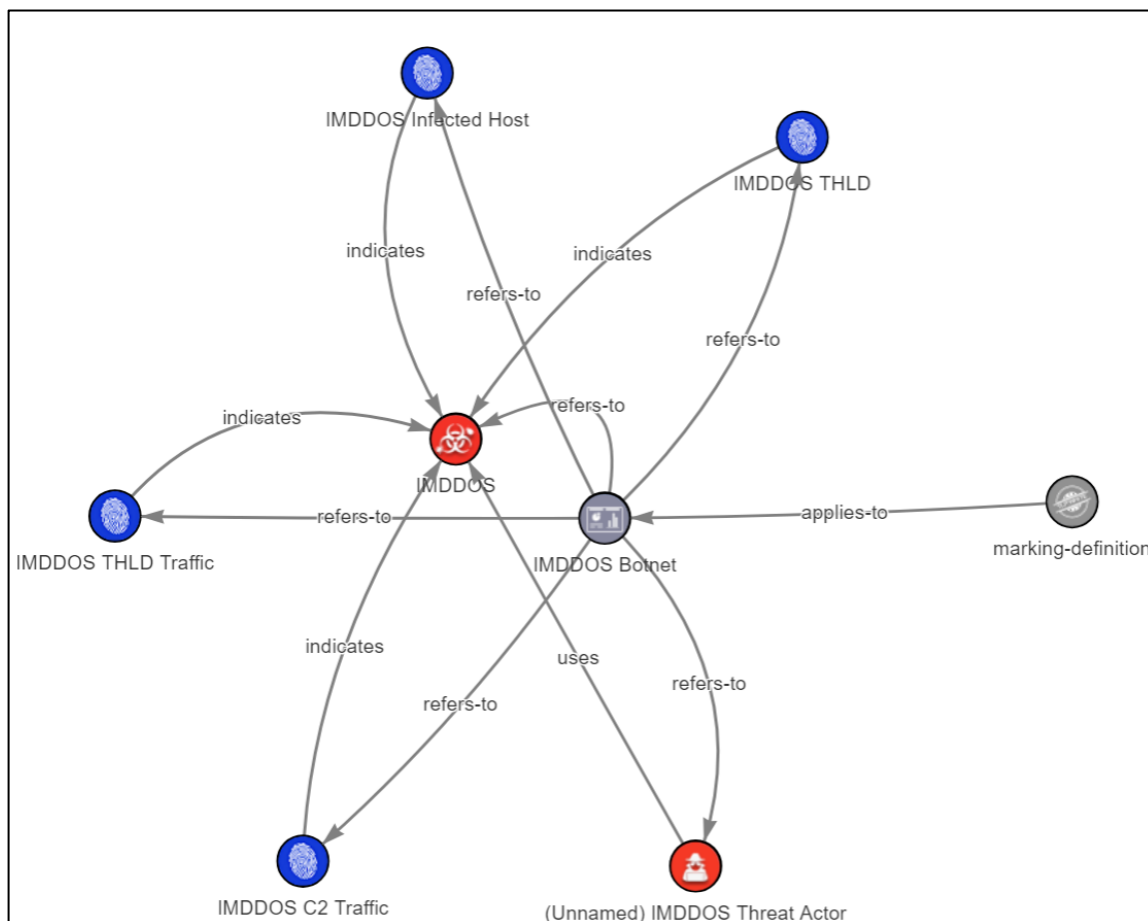


Figure 4 STIX Visualizer 메인 화면의 그래프

기존 웹페이지에서는 STIX 데이터 파일에 포함된 모든 Object들이 하나의 캔버스 안에 자리 잡고 있어, 파일 내 데이터가 많을 경우 구조화된 해석을 하기에 어려움이 있다. 따라서 SDO 종류를 기준으로 검색했을 때 해당하는 Object만 하이라이트되는 방식을 제안하였다.

3. 기간 검색 기능

특정 기간에 보고된 공격 패턴, 멀웨어 등을 효율적으로 모아보기 위해 기간 검색 기능을 제안하였다. 사용자 목적에 따라, 과거에 발생한 보안 사건의 추이를 파악하거나 향후 발생할 수 있는 위협을 예측할 수도 있고, 실시간 보안 사고 대응에 필요한 정보만을 신속하게 찾아내는 데도 도움이 될 것으로 예상된다.

4. 범례(Legend) 내 토글 기능

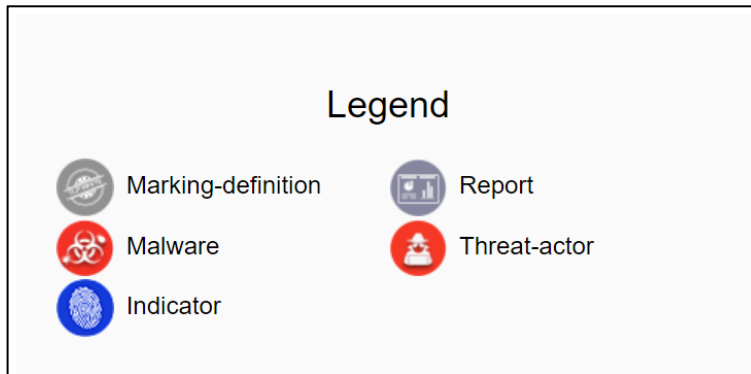


Figure 5 STIX Visualizer 메인 화면의 범례(Legend) 부분

범례의 각 항목을 클릭하면 항목에 해당하는 Object들이 그래프에서 제외되거나 다시 추가되는 모습을 보였다. 항목별로 선택해 보며 그래프에서 확인하기 이전에, 각 항목에 해당하는 Object 목록을 Legend에서 찾아볼 수 있도록 하는 기능을 제안하였다. 이때, 토글 방식을 사용하여 처음부터 많은 공간을 차지하지 않게 한다.

5. 디자인 보완

a. 텍스트 폰트 변경

기존 설정된 폰트는 그래프 내 요소들(간선, 아이콘)과 겹쳐 표시될 때 알아보기 힘들다는 의견이 있었다. 가독성 향상을 위해 새로운 폰트를 알아보기로 논의하였다.

b. 선택된 노드 강조

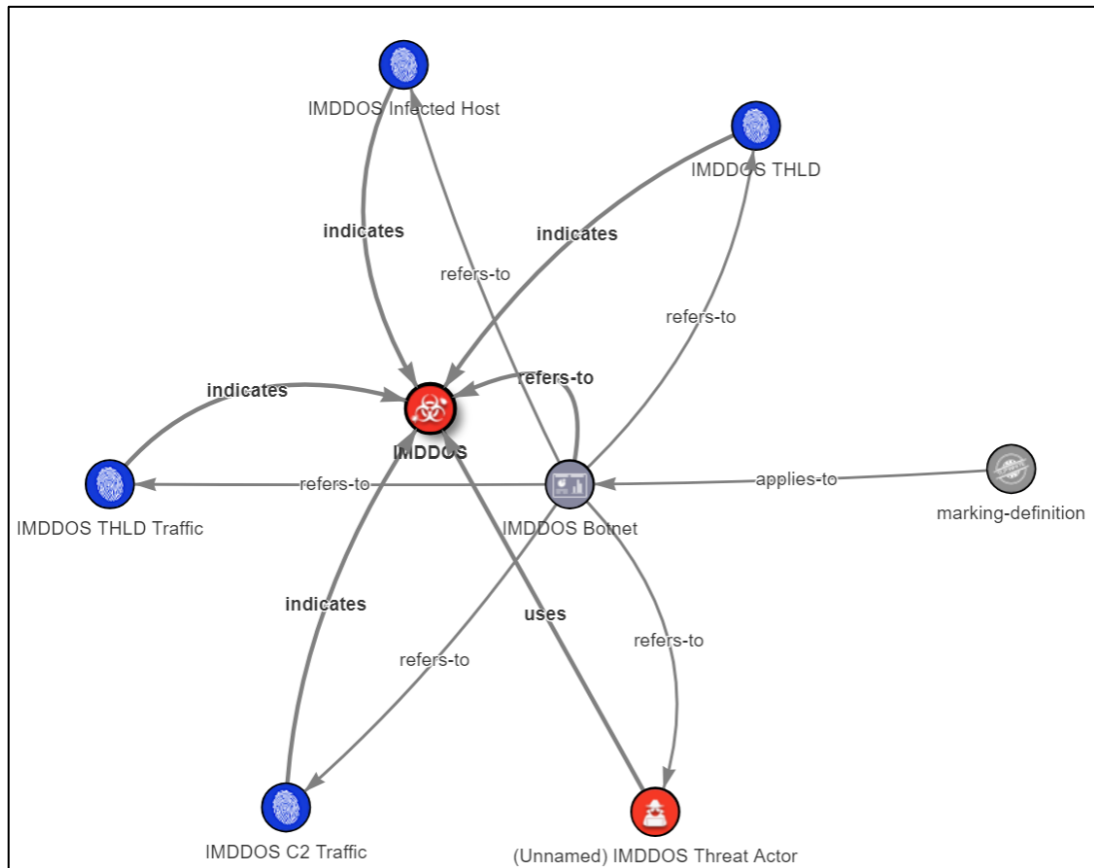
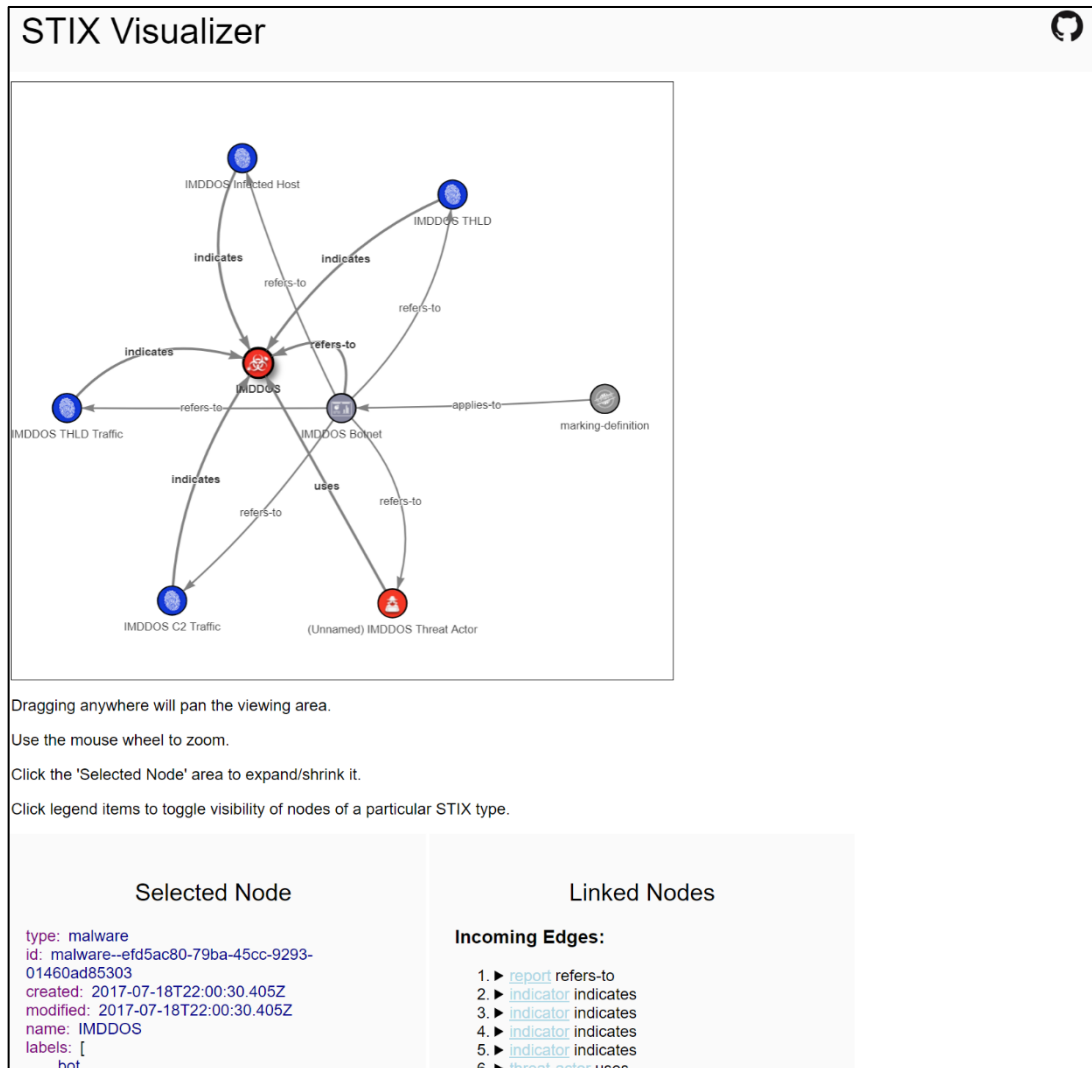


Figure 6 STIX Visualizer 메인 화면의 그래프에서 노드 선택 시 화면

그래프 내 Object 클릭 시 해당 Object와, 연결된 간선들이 굵게 표시되긴 하지만, 데이터가 방대한 경우 강조 표시가 눈에 띄지 않아 불편함이 있었다. 선택된 노드와 연결된 부분들에 대해서는 다른 색상을 사용하거나 투명도를 조절하여 더 강하게 표현하는 방식이 논의되었다.

c. 웹페이지 사용법 선택 표시



반응형 웹페이지로 되어있어 화면 너비가 줄어들면 Selected Node, Linked Nodes 설명 부분이 아래로 내려오게 되는데, 사용법 4문장에 의해 스크롤을 아래로 많이 내려야만 확인 가능하다. 웹페이지 사용법은 자주 확인하는 항목이 아니므로 자바스크립트의 Tooltip 또는 Collapse 기능을 사용하여 숨기기로 하였다.

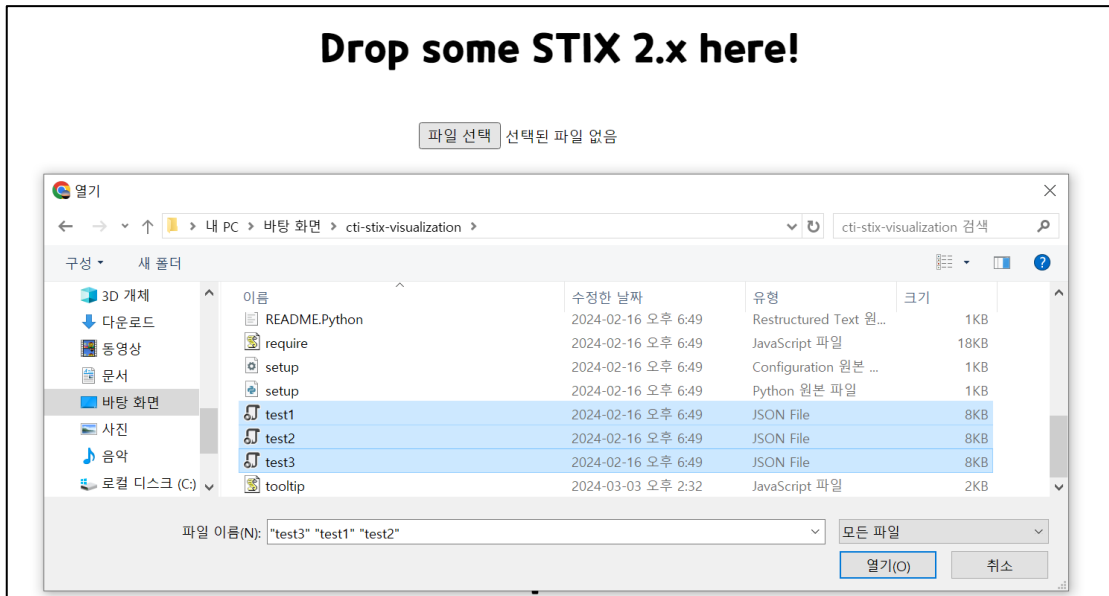
d. 필요 없는 부분, 기능 삭제

Selected Node 설명 부분을 클릭하였을 때 가로로 확대되며 다른 요소들(Linked Node 설명 부분, 그래프 일부)을 가리는 것을 확인하였다. 필요 없는 기능이라 생각하여 해당 부분을 코드상에서 제외할 것을 제안하였다.

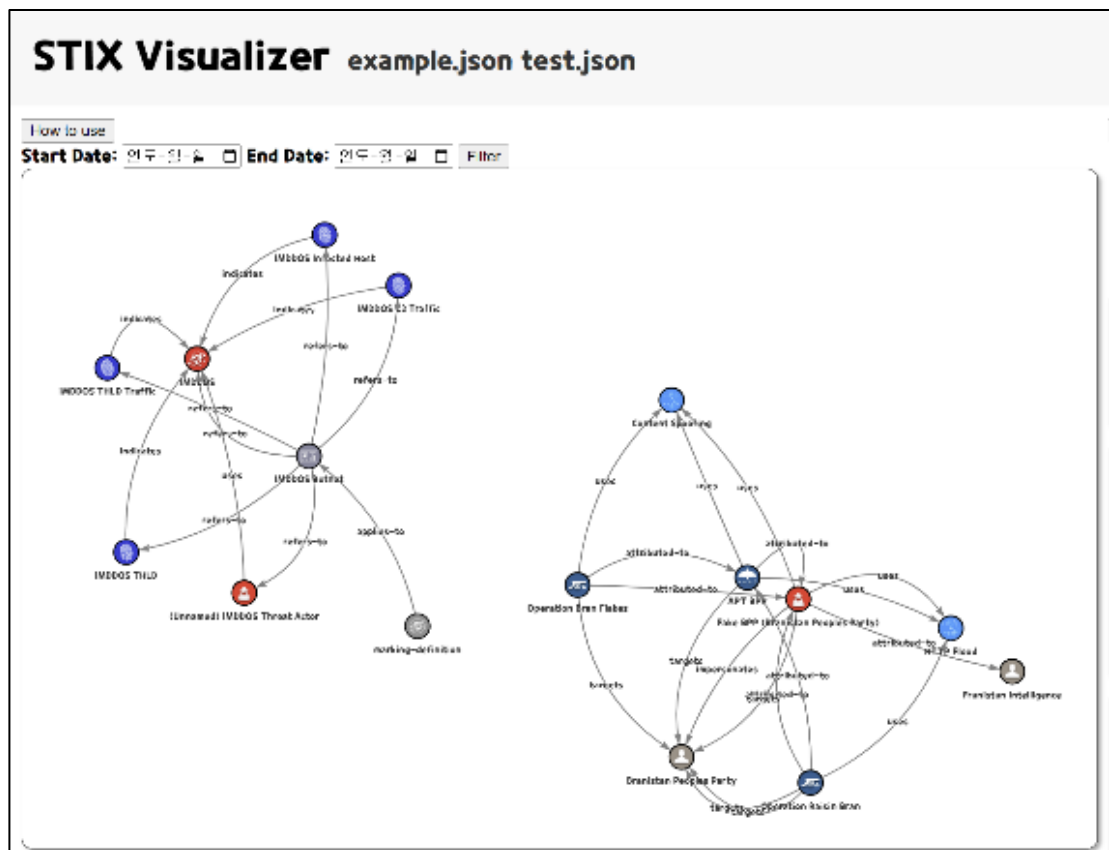
(2) 웹사이트 변경 사항

1. 두 개 이상의 파일 업로드 기능

- 두 개 이상의 JSON 파일을 선택하여 올릴 수 있도록 하였다.



- 각각의 JSON의 내용을 결합하여 그 결과를 그래프로 표현한다.



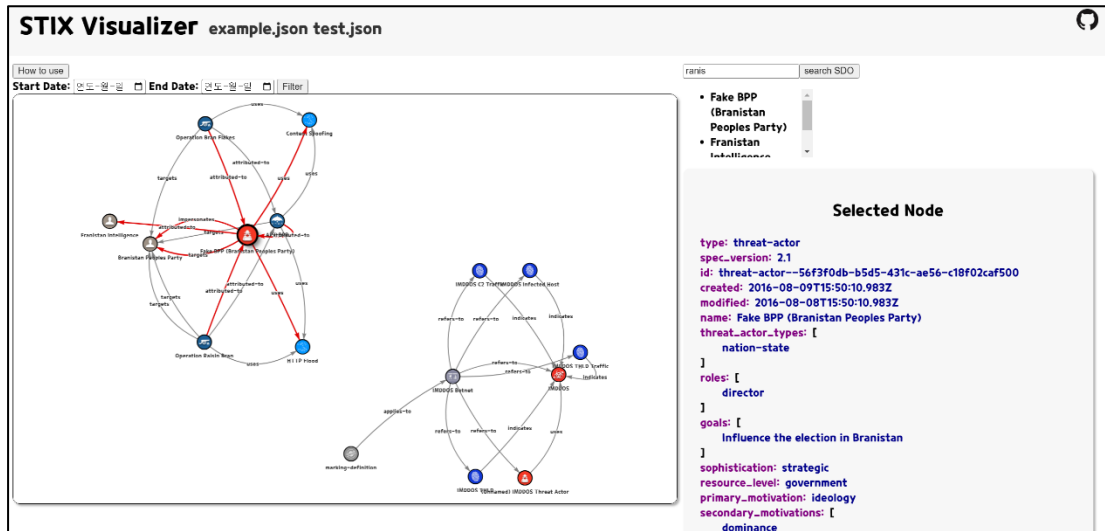
c. 코드는 다음과 같다.

```
function handleFiles(files) {  
  // files is a FileList of File objects (in our case, just one)  
  for (var i = 0, f; f = files[i]; i++) {  
    let customConfig = '';  
    let jsonData = {};  
  
    // 단일 파일 텍스트로 읽어오는 함수  
    function readFile(file) {  
      return new Promise((resolve, reject) => {  
        const reader = new FileReader();  
  
        reader.onload = function (e) {  
          const data = JSON.parse(e.target.result);  
          resolve(data);  
        };  
  
        reader.onerror = function (e) {  
          reject(e);  
        };  
  
        reader.readAsText(file);  
      });  
    }  
  }  
}
```

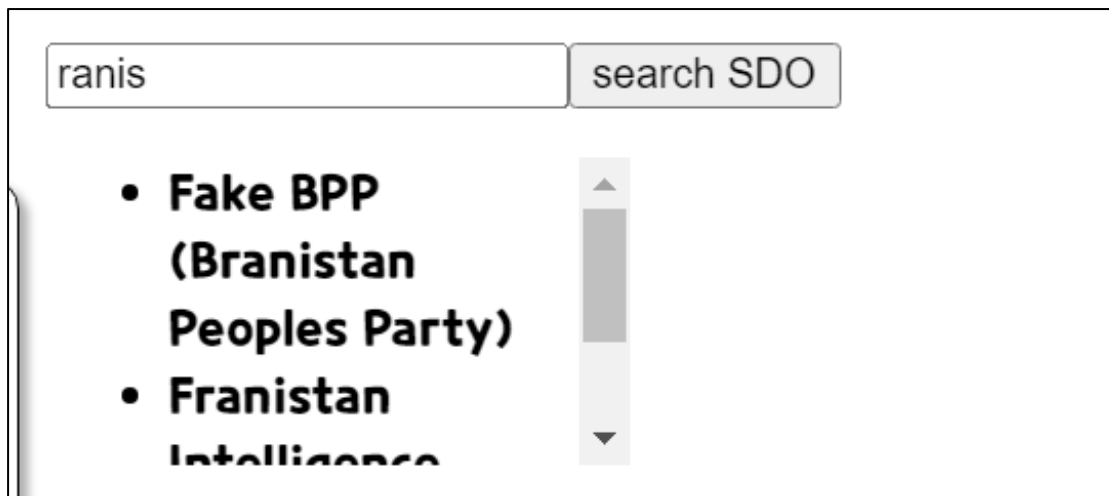
```
// FileList 배열로 변환  
const filesArray = Array.from(files);  
  
// 모든 파일 읽고 데이터 결합하는 Promise  
Promise.all(filesArray.map(readFile))  
  .then((fileDataArray) => {  
    // 여러 파일에서 데이터 결합  
    jsonData = fileDataArray.reduce((acc, curr) => {  
      if (curr.type === 'bundle') {  
        // type이 bundle이면 objects 값만 추출  
        return acc.concat(curr.objects || []);  
      } else {  
        // bundle이 아니면 그대로 결합  
        return acc.concat(curr);  
      }  
    }, []);  
  
    jsonData = JSON.stringify(jsonData);  
  
    vizStixWrapper(jsonData, customConfig);  
  
    document.getElementById('chosen-files').innerText = filesArray.map(file => file.name).join(' ');  
  })  
  .catch((error) => {  
    console.error('Error reading file:', error);  
  });  
  
linkifyHeader();  
}
```

2. SDO 검색 기능

- a. SDO의 name을 검색하면 해당 SDO의 노드가 강조표시되고, 연결된 엣지는 빨간색으로 하이라이트된다.



- b. "search SDO" 버튼 옆에 입력 중인 텍스트를 기준으로 SDO 이름에 대한 자동완성 리스트가 실시간으로 출력되어, 원하는 검색 결과를 얻기 쉽게 구성했다.



c. 코드는 다음과 같다.

i. 검색창, 검색 결과 자동완성 요소 생성

```
let searchInput = document.createElement("input");
searchInput.setAttribute("type", "text");
searchInput.setAttribute("id", "searchInput");
searchInput.classList.add("search-input");

let searchButton = document.createElement("button");
searchButton.setAttribute("id", "searchButton");
searchButton.textContent = "search SDO";

let suggestionList = document.createElement("ul");
suggestionList.setAttribute("id", "suggestions");
suggestionList.classList.add("suggestions")

let searchDiv = document.querySelector("#canvas-container");
searchDiv.insertBefore(searchInput, searchDiv.children[1]);
searchDiv.insertBefore(searchButton, searchDiv.children[2]);
searchDiv.insertBefore(suggestionList, searchDiv.children[3]);
```

```
for (let key of stixNameToObj.keys()){
  if (key){
    if(key.includes(keyword) && !view.isHidden(key)){
      matchedResults.push(key);
    }
  }
}

suggestionList.innerHTML = '';

for (let result of matchedResults){
  const li = document.createElement('li');
  li.textContent = result;
  li.addEventListener(
    "click", e => {
      e.stopPropagation(),
      suggestionListClickHandler(edgeDataSet, stixIdToObj, stixNameToObj.get(li.textContent))
    }
  );
  suggestionList.appendChild(li);
}
});
searchButton.addEventListener(
  "click", e => {
    e.stopPropagation(),
    searchNameHandler(edgeDataSet, stixIdToObj, stixNameToObj)
  }
);
searchInput.addEventListener("keydown", function(event){
  if(event.keyCode === 13) {
    event.stopPropagation(),
    searchNameHandler(edgeDataSet, stixIdToObj, stixNameToObj)
  }
});
```

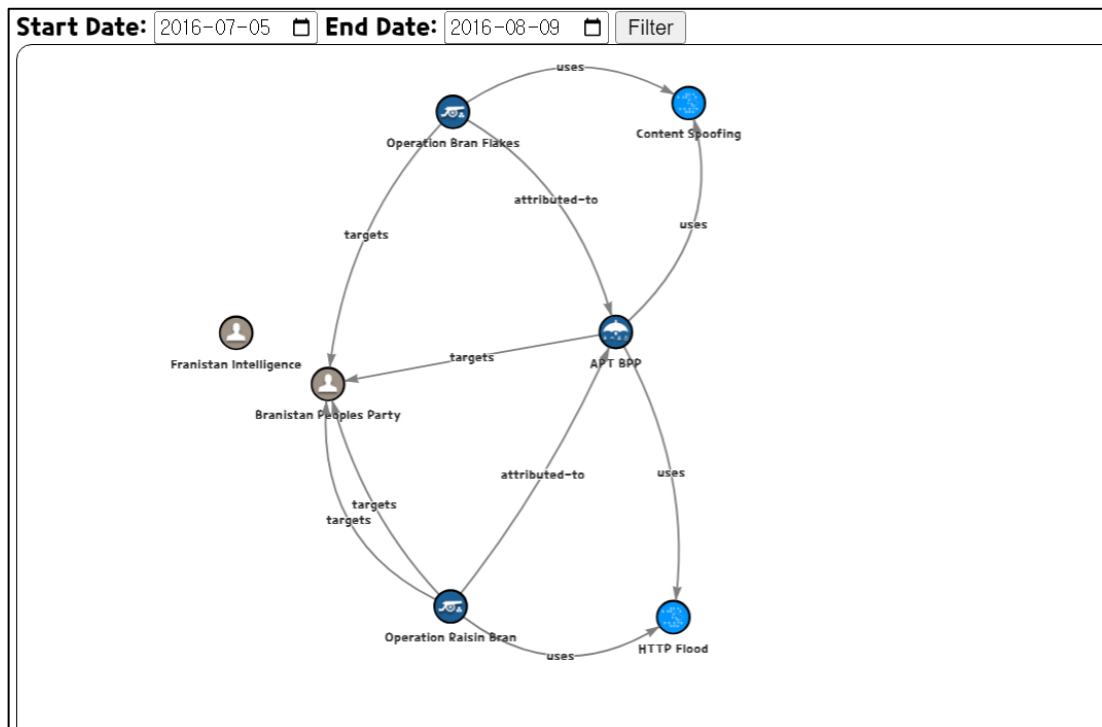

ii. 각 요소에 대한 eventHandler

```
function searchNameHandler(edgeDataSet, stixIdToObject, stixNameToObject){
  var searchText = document.getElementById("searchInput").value;
  // 여기서 searchText로 검색 진행 (일치하는 키 있으면 객체가, 없으면 undefined가 된다)
  let stixObject = stixNameToObject.get(searchText);
  if(stixObject)
  {
    if(!view.isHidden(searchText))
    {
      view.selectNode(stixObject.get('id'));
      populateSelected(stixObject, edgeDataSet, stixIdToObject);
    }
    else
    {
      view.selectNode(undefined);
      populateSelected(new Map([[ "", "(Hidden)"] ]), edgeDataSet, stixIdToObject);
    }
  }
  else
  {
    view.selectNode(undefined);
    populateSelected(new Map([[ "", "(No Result)"] ]), edgeDataSet, stixIdToObject);
  }
}
```

```
function suggestionListClickHandler(edgeDataSet, stixIdToObject, stixObject){
  if(stixObject)
  {
    view.selectNode(stixObject.get('id'));
    populateSelected(stixObject, edgeDataSet, stixIdToObject);
  }
  else
  {
    view.selectNode(undefined);
    populateSelected(new Map([[ "", "(No Result)"] ]), edgeDataSet, stixIdToObject);
  }
}
```

3. 날짜 검색 기능

- a. 날짜 범위를 선택하여 해당 범위의 created를 가진 노드와 엣지만을 표현한다.



- b. 코드는 다음과 같다.

- i. Constructor에서 created 속성 추가

```
this.#nodeDataSet = new visjs.DataSet();

nodeDataSet.forEach((item, id) => {
  let tmp = stixIdToObject.get(id).get("created");

  // tmp를 Date 객체로 변환
  let reform = new Date(tmp);
  this.#nodeDataSet.add({
    ...item,
    group: stixIdToObject.get(id).get("type"),
    createdAt: reform.toISOString()
  });
});
```

- ii. 날짜 범위에 따라 노드를 toggle하는 함수 추가

```
function handleFilterDate() {
    if(!view)
        return;

    let startDate = document.getElementById("startDate").value;
    let endDate = document.getElementById("endDate").value;
    //alert(startDate + " " + endDate);
    view.toggleStixDate(startDate, endDate);
}
```

- iii. Date 관련 eventHandler

```
// start와 end 사이의 created object property를 가진 노드만 보이게 하는 함수
toggleStixDate(start, end)
{
    let startDate = new Date(start);
    let endDate = new Date(end);

    let nodes = this.nodeDataSet.get({
        filter: item => item.createdDate >= startDate.toISOString() && item.createdDate <= endDate.toISOString(),
        fields: ["id", "hidden"]
    });

    if (nodes.length === 0){
        alert("There is no node between the dates");
        return;
    }

    this.enablePhysics();

    let hiding = false;
    let toggledNodes = [];
    let toggledEdges = [];

    let otherNodes = this.nodeDataSet.get({
        filter: item => item.createdDate < startDate.toISOString() || item.createdDate > endDate.toISOString(),
        fields: ["id", "hidden"]
    });

    let hiddenNodes = [];
    let hiddenEdges = [];
    let hiddenEdgeIds = new Set();

    for (let node of otherNodes)
    {
        hiddenNodes.push({
            id: node.id, hidden: true, physics: false
        });

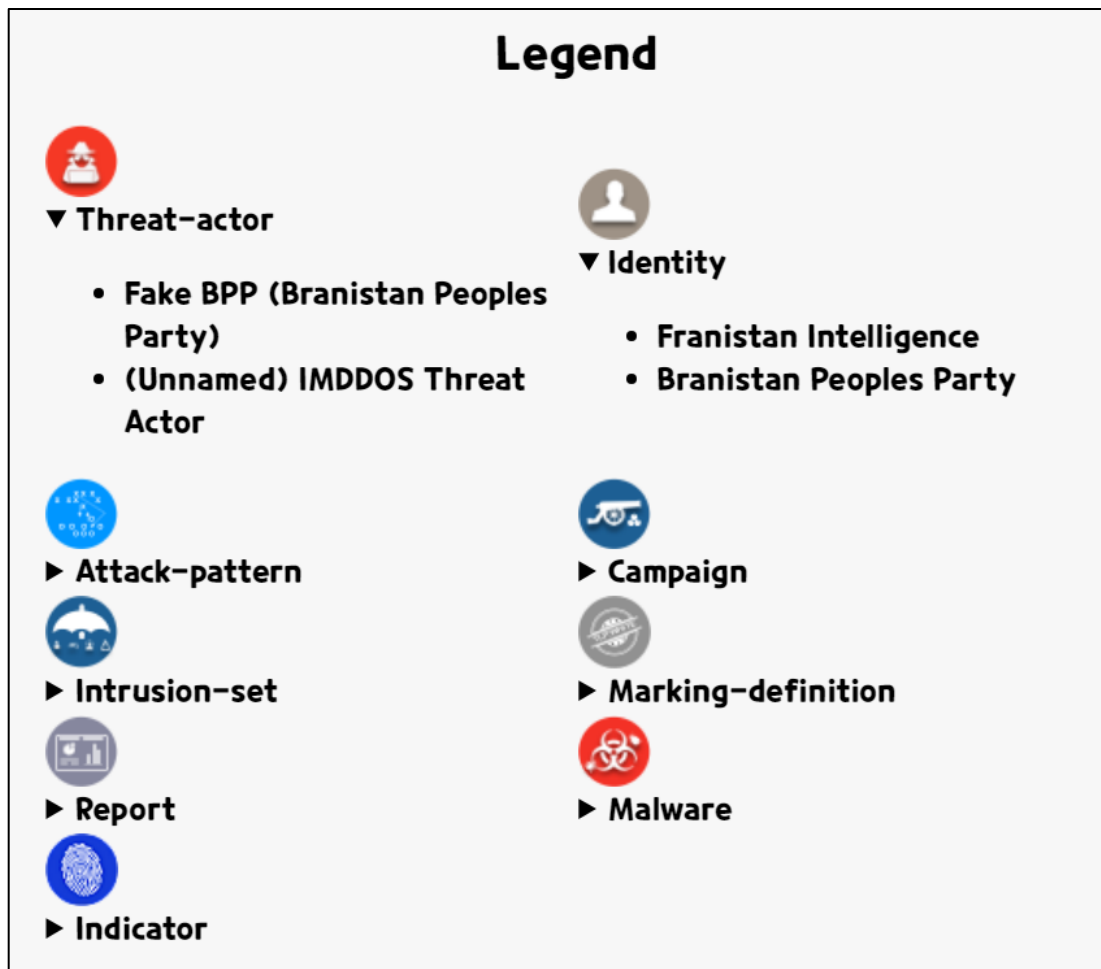
        let edgesForNode = this.edgeDataSet.get({
            filter: item => (item.from === node.id || item.to === node.id)
                && !item.hidden === true && !hiddenEdgeIds.has(item.id),
            fields: ["id", "from", "to"]
        }); // undefined같은 경우도 있어서 !item.hidden으로 해야함.

        for (let edge of edgesForNode)
        {
            hiddenEdges.push({
                id: edge.id, hidden: true, physics: false
            });
            hiddenEdgeIds.add(edge.id);
        }
    }

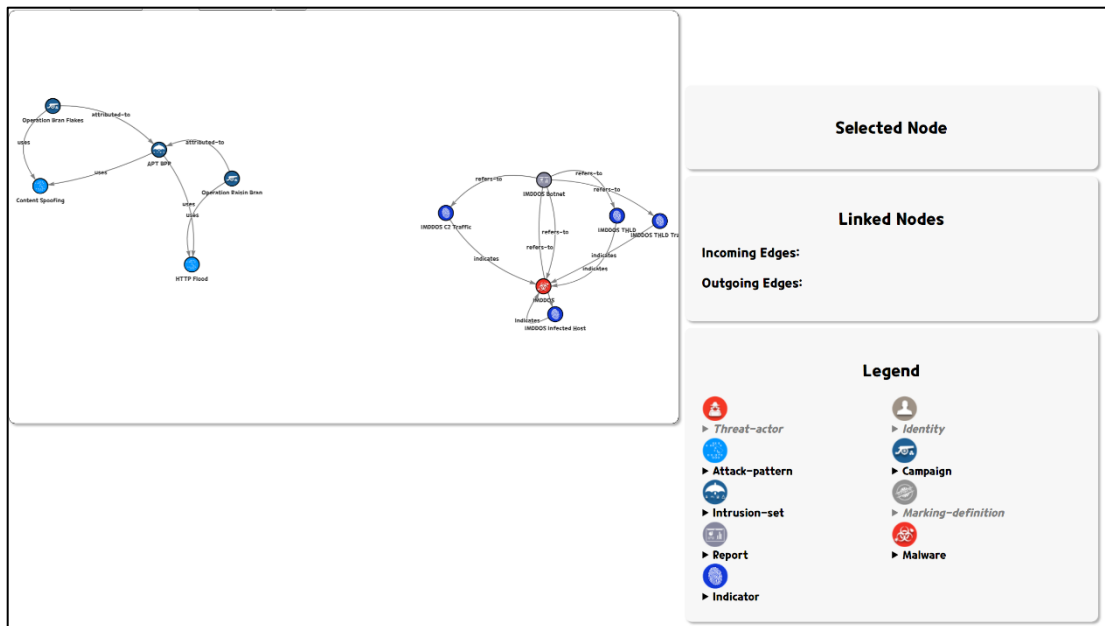
    this.nodeDataSet.updateOnly(hiddenNodes);
    this.edgeDataSet.updateOnly(hiddenEdges);
}
```

4. 범례(legend) 내 토글 기능

- a. 범례의 토글을 클릭하면 해당 범례에 속한 SDO 목록을 볼 수 있다.



b. 토크 내 SDO를 클릭하면 그래프 내에서 on/off 할 수 있다.



c. 코드는 다음과 같다.

i. toggleStixName (개별 노드 처리 함수)

```
toggleStixName(stixName)
{
  let node = this.nodeDataSet.get({
    filter: item => item.label === stixName,
    fields: ["id", "hidden"]
  });

  if (node.length === 0)
    return;
  node = node[0];
  this.enablePhysics();

  // Whether we are hiding or showing nodes of the selected type.
  // If first node is currently hidden, we must be showing, and vice
  // versa.
  let hiding = !node.hidden;

  let toggledNodes = [];
  let toggledEdges = [];

  // An edge could connect two nodes of the same type. Ensure we don't
  // toggle an edge more than once!
  let toggledEdgeIds = new Set();

  // Toggling the node is simple
  toggledNodes.push({
    id: node.id, hidden: hiding, physics: !hiding
  });

  // Toggling the edges is more complex...
  let edgesForNode = this.edgeDataSet.get({
    // find (a) edges connecting to 'node'; (b) edges with the
    // right visibility; (c) edges we have not already seen.
    filter: item => (item.from === node.id || item.to === node.id)
      && !item.hidden === hiding && !toggledEdgeIds.has(item.id),
    fields: ["id", "from", "to"]
  });
}
```

```

if (hiding)
{
  // simple case: unconditionally hide everything
  for (let edge of edgesForNode)
  {
    toggledEdges.push({
      id: edge.id, hidden: true, physics: false
    });
    toggledEdgeIds.add(edge.id);
  }
}
else
{
  // showing is a more complex case: gotta check the other ends
  // of the edges. Only show if the other end is also visible
  // or of the selected type (meaning it will become visible).
  for (let edge of edgesForNode)
  {
    let otherEndId;
    if (edge.from === node.id)
      otherEndId = edge.to;
    else
      otherEndId = edge.from;

    let otherEndNode = this.nodeDataSet.get(
      otherEndId,
      {fields: ["group", "hidden"]}
    );

    if (!otherEndNode.hidden)
    {
      toggledEdges.push({
        id: edge.id, hidden: false, physics: true
      });
      toggledEdgeIds.add(edge.id);
    }
  }
}

this.nodeDataSet.updateOnly(toggledNodes);
this.edgeDataSet.updateOnly(toggledEdges);
}

```

ii. 토글 클릭 처리

```

function legendClickHandler(event)
{
  if (!view)
    return;

  let td, toggledStixObject, toggledStixType;
  let clickedTagName = event.target.tagName.toLowerCase();

  if (clickedTagName === "li")
  {
    // ... if the stix item text was clicked
    toggledStixObject = event.target.textContent;
    view.toggleStixName(toggledStixObject);
    event.target.classList.toggle("typeHidden");

    if (event.target.classList.value === "" && event.target.parentElement.parentElement.querySelector("summary").classList.value === "typeHidden")
      event.target.parentElement.parentElement.querySelector("summary").classList.toggle("typeHidden")
    else if (event.target.classList.value === "typeHidden")
    {
      let li = event.target.parentElement.querySelectorAll("li")

      let hiding = true
      for (let i of li)
        if (i.classList.value === "")
          hiding = false

      if(hiding)
        event.target.parentElement.parentElement.querySelector("summary").classList.toggle("typeHidden")
    }
  }
}

```

```

else
{
    let summary;

    if (clickedTagName === "td")
        // ... if the legend item text was clicked
        td = event.target;
    else if (clickedTagName === "img")
        // ... if the legend item icon was clicked
        td = event.target.parentElement;
    else
        return;

    // The STIX type the user clicked on
    summary = td.querySelector("details").querySelector("summary");
    toggledStixType = summary.textContent;
    toggledStixType = toggledStixType.trim().toLowerCase();

    view.toggleStixType(toggledStixType);

    // style change to remind users what they've hidden.

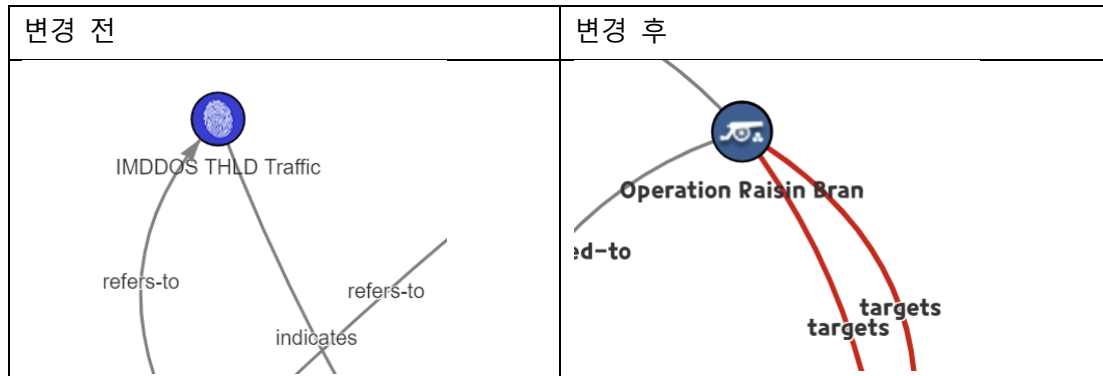
    summary.classList.toggle("typeHidden");
    let li = td.querySelector("details").querySelectorAll("ul li");

    li.forEach(function(item) {
        if (item.classList.value !== summary.classList.value)
            item.classList.toggle("typeHidden")
    });
}
}

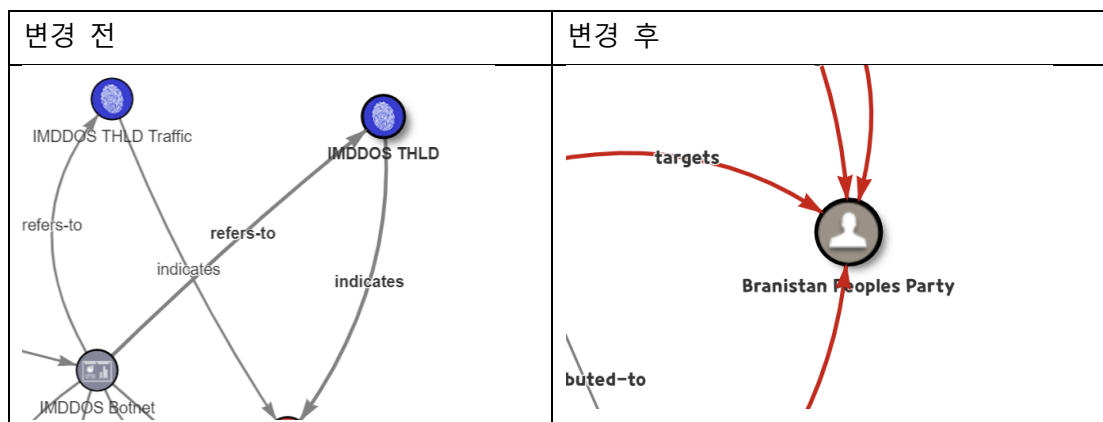
```

5. 디자인 보완

a. 그래프 내 텍스트를 두껍게 하여 가독성을 높였다.



b. 노드 선택 시 연결된 엣지를 빨간색으로 표시하여 가독성을 높였다.



```

if (this.selected || this.hover) {
  /*
   * @leesoomin
   * Analysis: can assign custom design to the clicked node & connected edge here.
   */
  if (this.chooser === true) {
    if (this.selected) {
      var selectedWidth = this.options.selectionWidth;

      if (typeof selectedWidth === "function") {
        values.width = selectedWidth(values.width);
      } else if (typeof selectedWidth === "number") {
        values.width += selectedWidth;
      }

      values.width = Math.max(values.width, 0.3 / this.body.view.scale);
      values.color = "#DF0101";
      values.shadow = this.options.shadow.enabled;
    }
  }
}

```


c. How to use를 호버링을 통해 볼 수 있도록 하여 필요시에만 확인할 수 있도록 하였다.

변경 전	<div>(Unnamed) IMDD</div> <p>Dragging anywhere will pan the viewing area.</p> <p>Use the mouse wheel to zoom.</p> <p>Click the 'Selected Node' area to expand/shrink it.</p> <p>Click legend items to toggle visibility of nodes of a particular STIX type.</p>
변경 후	<div>How to use</div> <p>Dragging anywhere will pan the viewing area.</p> <p>Use the mouse wheel to zoom.</p> <p>Click the 'Selected Node' area to expand/shrink it.</p> <p>Click legend items to toggle visibility of nodes of a particular STIX type.</p>

```

<!-- toggle tooltip for "How to use" -->
<script src="tooltip.js"></script>
<style>
.tooltip {
  position: fixed;
  padding: 10px 20px;
  border: 1px solid #b3c9ce;
  border-radius: 4px;
  text-align: left;
  color: #333;
  background: #fff;
  box-shadow: 3px 3px 3px rgba(0, 0, 0, .3);
}
</style>
</head>

```

```

let tooltipElem;

document.onmouseover = function(event) {
    let target = event.target;

    // data-tooltip 속성이 있는 요소
    let tooltipHtml = target.dataset.tooltip;
    if (!tooltipHtml) return;

    // 툴팁 요소를 만듭니다.

    tooltipElem = document.createElement('div');
    tooltipElem.className = 'tooltip';
    tooltipElem.innerHTML = tooltipHtml;
    document.body.append(tooltipElem);

    // 툴팁 요소를 data-tooltip 속성이 있는 요소 위, 가운데에 위치시킵니다.
    let coords = target.getBoundingClientRect();

    let left = coords.left + (target.offsetWidth - tooltipElem.offsetWidth) / 2;
    if (left < 0) left = 0; // 툴팁이 창 왼쪽 가장자리를 넘지 않도록 합니다.

    let top = coords.top - tooltipElem.offsetHeight - 5;
    if (top < 0) { // 툴팁이 창 위로 넘치면 요소 아래에 보여줍니다.
        top = coords.top + target.offsetHeight + 5;
    }

    tooltipElem.style.left = left + 'px';
    tooltipElem.style.top = top + 'px';
};

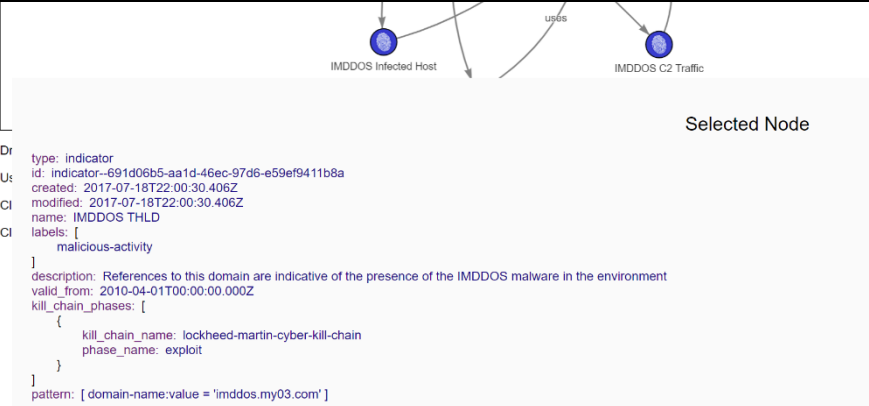
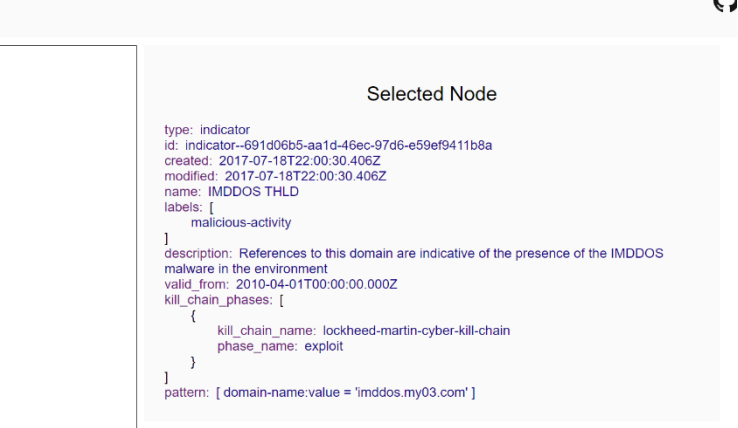
document.onmouseout = function(e) {

    if (tooltipElem) {
        tooltipElem.remove();
        tooltipElem = null;
    }

};

```

d. Selected Node 칸을 클릭했을 때 해당 부분을 확대하는 기능을 삭제하였다.

<p>변경 전</p>	
<p>변경 후</p>	

4. 결론

(1) 활용 가능성

첫 번째, 여러 STIX 데이터 파일을 한 번에 한 화면에 시각화하고 싶을 때 활용할 수 있다. 기존 웹페이지는 1개의 파일만 올릴 수 있었다. 하지만 STIX 데이터들이 여러 개로 나뉘어 있지만 하나의 화면에서 보고 싶을 때를 고려하여 여러 파일을 올릴 수 있도록 변경하였다.

두 번째, 여러 SDO 중에서 날짜별로, SDO의 instance 별로 원하는 것을 선별해서 시각화하고 싶을 때 활용할 수 있다. 어떤 STIX 파일은 10,000줄 이상의 STIX 데이터들로 구성되어 있다. 이런 상황에서 보고 싶은 부분만을 빼내서 파일을 만들거나 STIX 데이터를 재구성하는 것은 비효율적이라고 생각했다. 따라서, 필터링 기능을 추가하여 원하는 정보가 포함된 URL이나 파일 등을 넣었을 때 원하는 기간에 해당하는 SDO들만 보여주고 SDO 중에서 보고 싶은 instance로만 선별해서 화면에 나타낼 수 있도록 변경하였다.

(2) 시사점 및 향후 프로젝트

여러 파일을 한 번에 올릴 수 있도록 기능을 변경하여 올려진 파일들의 데이터를 한 화면에 나타내었다. SDO 검색 기능을 추가하여 원하는 SDO 정보를 확인하기 위해 하나씩 눌러보는 수고로움을 없앴다. 날짜 필터링 기능을 통해 원하는 기간의 STIX 데이터를 확인할 수 있다. SDO 종류별 토글 기능을 추가하여 각각 SDO에는 어떤 instance들이 있는지 확인할 수 있고 instance 이름을 클릭함으로써 화면에 해당 instance를 지울 수도, 추가할 수도 있다. 또한, 텍스트 두께, edge 색 등 디자인을 보완하여 가독성을 높였다. 이러한 기능의 변경과 추가로 기존 웹사이트보다 사용자의 편리성을 추구하였다.

해당 프로젝트를 위해 STIX 공식 문서를 읽고 정리함으로써 STIX 이해도를 높일 수 있었다. 또한, 기존 웹사이트의 기능을 파악하고 변경하기 위한 코드 분석 경험과 웹페이지 개발 경험 등을 쌓을 수 있었다. 마지막으로, 정보 공유의 중요성이 커지는 지금, 정보 공유 체계 중 하나인 STIX 프로젝트를 통해 정보 공유 체계 표준화의 중요성을 깨닫고 정보 공유 체계의 효율적인 시각화 방법 등에 대해 깊이 고민해 보는 시간을 가질 수 있었다.

향후, SDO 토글 부분에 검색 기능을 추가하여 원하는 instance를 일일이 찾는 게 아니라 검색해서 쉽게 찾을 수 있도록 하고 SDO 부분을 사이드바로 만드는 등 디자인 부분을 좀 더 수정한다면 가독성이 더욱 높아질 것으로 예상된다. 또한, STIX 데이터를 활용하는 플랫폼인 OpenCTI에서는 STIX 데이터를 웹사이트처럼 시각화하는 기능이 없기 때문에 OpenCTI 코드에 맞게 재구성해서 해당 기능을 추가해 보는 프로젝트나 침해사고 보고서에서 STIX 형식으로 변환하는 프로젝트도 향후 프로젝트로 적합해 보인다.