랜섬웨어 대비 로컬 백업 솔루션

팀원: 김윤재, 지현도
학과: 컴퓨터학부
지도교수: 김범현 교수님

1 연구 목적

랜섬웨어 공격 증대 및 피해 급증

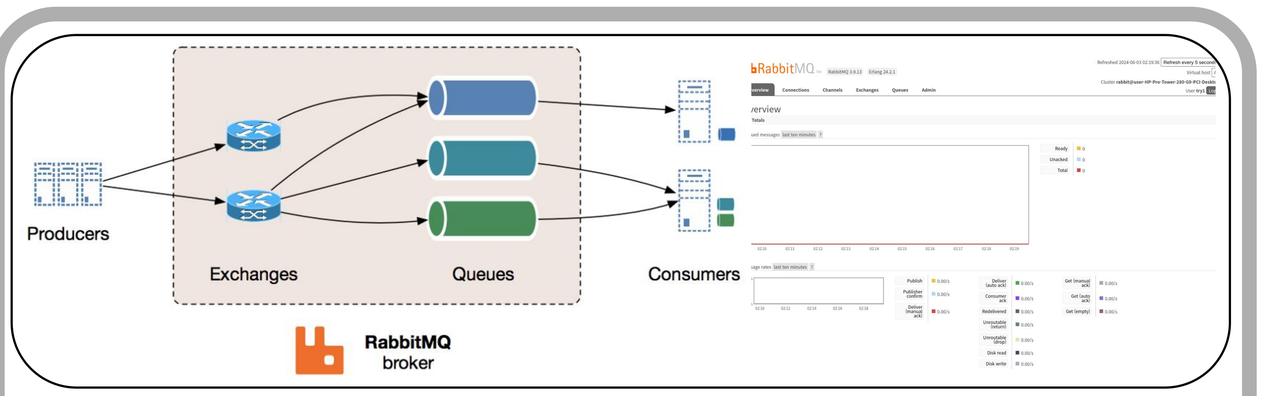
최근 랜섬웨어를 통한 데이터 암호화 및 탈취 사례가 급증하여 중요한 데이터의 백업이 필수적인 과제로 부상하고 있다.
데이터의 백업은 클라우드와 같은 외부 서버에 저장이 가능하지만 외부 서버를 신뢰할수 없음과 외부 서버 서비스를 이용하는 계정이 탈취당할 경우 개인의 데이터가 유출된다는 문제점이 있다. 따라서 외부 서버에 의존하지 않고 데이터를 개인의 로컬에 쉽게 백업 가능하도록 연구를 진행하였다.

데이터 로컬 백업의 필요성

1. 외부 서버 신뢰 불가능: 기업들의 과도한 개인정보 수집으로 인한 사용자의 불안감 급증.

2. 보안 위협: 외부 서버를 노린 거대한 해커 집단의 개인 정보 탈취의 가능성이 존재. 이는 심각한 피해를 초래.

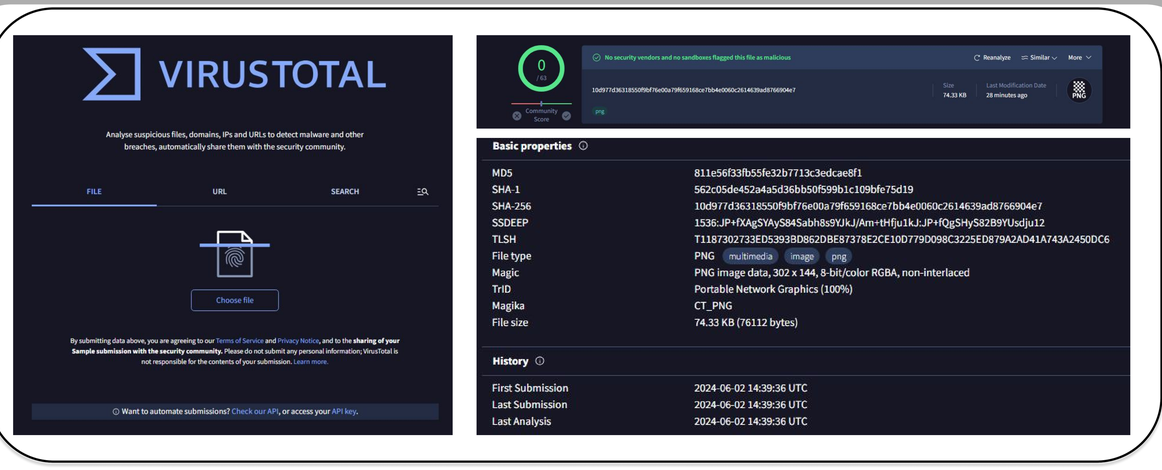
2 Rabbit 메세지큐



1. 안전성: 컴퓨터간의 데이터 전송을 위한 메세지 브로커는 분리되어 있으므로, 한대의 컴퓨터가 랜섬웨어에 감염되더라도, 한번 브로커로 전송한 데이터는 메시지 브로커의 큐를 통해 안전하게 보관된다. 따라서 다른 컴퓨터에서 데이터를 받을 수 있다.
2. 편리성: 브로커는 어떤 컴퓨터에서든 사용할 수 있으며, 브로커 GUI를 통해 메세지큐를 쉽게 관리가능하다.

*GUI: Graphical User Interface

3 VirusTotal OpenAPI

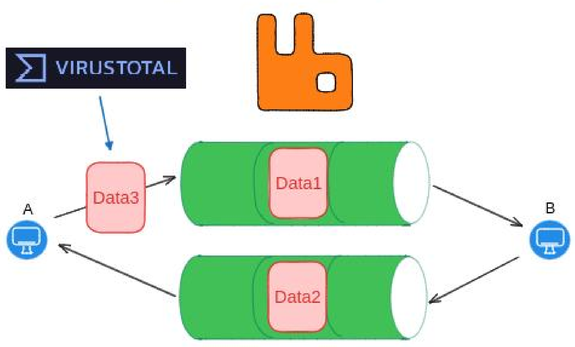


VirusTotal은 파일과 URL을 분석하여 악성 코드와 기타 보안 위협을 감지하는 무료 온라인 서비스이다. 이를 활용하여 브로커가 관리하는 큐로 데이터를 전송하기 전 멀웨어감지를 통해 데이터의 무결성을 확보한다.

*API: Application Programming Interface

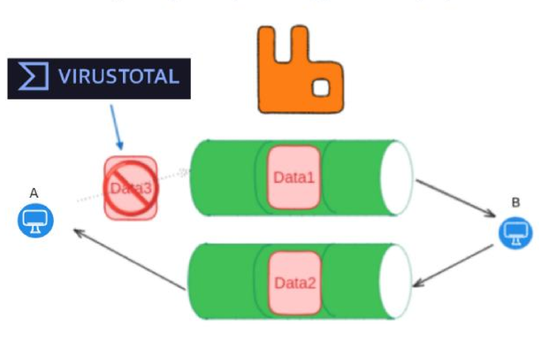
4 작동 과정

1. 컴퓨터 A가 랜섬웨어 공격을 당하기 전



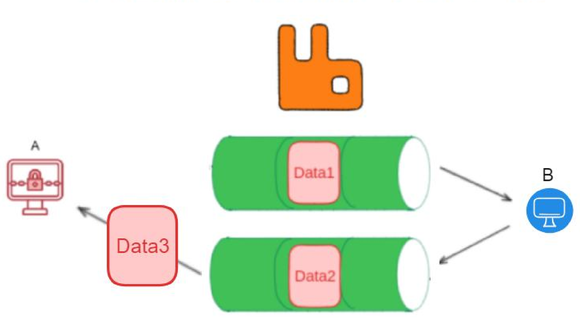
일반적인 상황에는 VirusTotal의 악성코드 검사를 마친 데이터를 메시지 브로커가 관리하는 큐를 통해 파일의 변화를 감지하는 특정 디렉토리로 전송한다.

2. 의심되는 파일은 큐로 보내지않음



만약 전송하려는 데이터가 위험하다고 판별되면 사용자에게 이를 알리고, 전송여부를 묻는다.

3. 랜섬웨어 공격을 받았을 시 데이터 백업



A가 랜섬웨어에 공격받아 A에 있는 중요 데이터가 암호화 되어 사용할 수 없을 시 B에 저장해놓은 데이터를 백업해온다.

```cpp
/*
 MerkelTree 구성 함수
 fileHash값들이 저장된 Vector를 넣어주면, 순서대로 머클트리 Leaf를 구성하고 Root노드의 32바이트 해시값을 리턴
*/
MerkleNode* constructMerkleTree(std::vector<std::string>& fileHashes) {
    std::vector<MerkleNode*> nodes;
    for (const auto& hash : fileHashes) {
        MerkleNode* node = new MerkleNode{hash, nullptr, nullptr};
        nodes.push_back(node);
    }

    while (nodes.size() > 1) {
        std::vector<MerkleNode*> parentLevel; //바로 위 레벨을 나타내는 변수
        for (size_t i = 0; i < nodes.size(); i += 2) {
            MerkleNode* left = nodes[i];
            MerkleNode* right = (i + 1 < nodes.size()) ? nodes[i + 1] : nullptr;
            std::string combinedHash = left->hash + (right ? right->hash : "");

            /* Tree의 left + right를 다시 해시처리 */
            std::array <char, SHA256_DIGEST_LENGTH * 2> hexHash = convertBinaryToHexadecimal(combinedHash);
            std::string hexHashStr(hexHash.data(), hexHash.size());
            MerkleNode* parent = new MerkleNode{hexHashStr, left, right};
            parentLevel.push_back(parent);
        }
        nodes = parentLevel;
    }
    return nodes.front();
}

void printAllFileHashesAndMerkleTreeHash(const std::string& directoryPath, std::map <std::string, std::time_t> currentFilesMap) {
    std::vector <std::string> currentFilesVector;
    for (auto it = currentFilesMap.begin(); it != currentFilesMap.end(); it++) {
        std::string fileHash = calculateFileHash(directoryPath + "/" + it->first);
        std::cout << directoryPath << "/" << it->first << " : " << fileHash << std::endl;
        currentFilesVector.push_back(fileHash);
    }
    MerkleNode* rootnode = constructMerkleTree(currentFilesVector);
    std::cout << "MerkleTree RootNode Hash: " << rootnode->hash << std::endl;
}
```

파일의 해시값을 노드로 사용하는 머클트리 사용하여 특정 디렉토리 파일들을 머클트리로 구성하고 루트노드를 보관하고, 특정디렉토리를 감시하는 동안에는 루트노드를 계속해서 업데이트한다. 이는 감시 프로세스가 꺼져있을때 파일의 변화를 감지하기 위함이다.