

# 악성코드 분석 웹서비스

팀 명 : 시큐리티  
지도 교수 : 이병천 교수님  
팀 장 : 김근오

2020. 11.  
중부대학교 정보보호학과

# 목 차

## 1. 서론

1.1 연구 배경 및 주제 선정 .....	3
-------------------------	---

## 2. 관련 연구

2.1 Golang .....	3
2.2 Docker .....	3
2.3 Minio .....	3
2.4 Yara .....	3
2.5 VirtualBox .....	3

## 3. 본론

3.1 시스템 구성 .....	4
3.2 프로그램 구성 및 설명 .....	4

## 4. 결론

4.1 결론 .....	9
--------------	---

## 5. 별첨

5.1 데모 스크린샷 .....	9
5.2 소스 코드 .....	12
5.3 발표 자료 .....	13

# 1. 서론

## 1.1 연구 배경 및 주제 선정

멀웨어를 좀 더 쉽게 분석하자! 에서 시작하게 된 프로젝트 이다.  
평소 사용하던 Virustotal, Malwares.com 들은 어떠한 방식으로 구성되어있고, 어떤 방식으로 분석하는지 궁금하여 직접 서비스를 구축해보고 싶어서 시작하였다.

## 2. 관련 연구

### 2.1 Golang

Go 는 C언어 기반으로 c++ 와 Java, Python 의 장점을 가진 언어이다.  
객체지향 프로그래밍 언어이며, 크로스컴파일을 지원하고, 동시성 프로그램을 쉽게 만들 수 있다.

### 2.2 Docker

Docker 는 애플리케이션을 신속하게 구축, 테스트 및 배포를 할 수 있는 소프트웨어 플랫폼 이다. Docker는 소프트웨어를 컨테이너라는 표준화된 유닛으로 패키징하며, 이 컨테이너에는 라이브러리, 시스템 도구, 코드, 런타임 등 소프트웨어를 실행하는 데 필요한 모든 것이 포함되어 있다.. Docker를 사용하면 환경에 구애받지 않고 애플리케이션을 신속하게 배포 및 확장할 수 있다.

### 2.3 MiniO

Minio 는 AWS 의 S3 SDK 와 호환되는 오픈소스 오브젝트 스토리지 서버 이다.  
Golang 으로 제작되어있으며, 현재 프로젝트에서 스토리지 서버로 활용 중 이다.

### 2.4 Yara

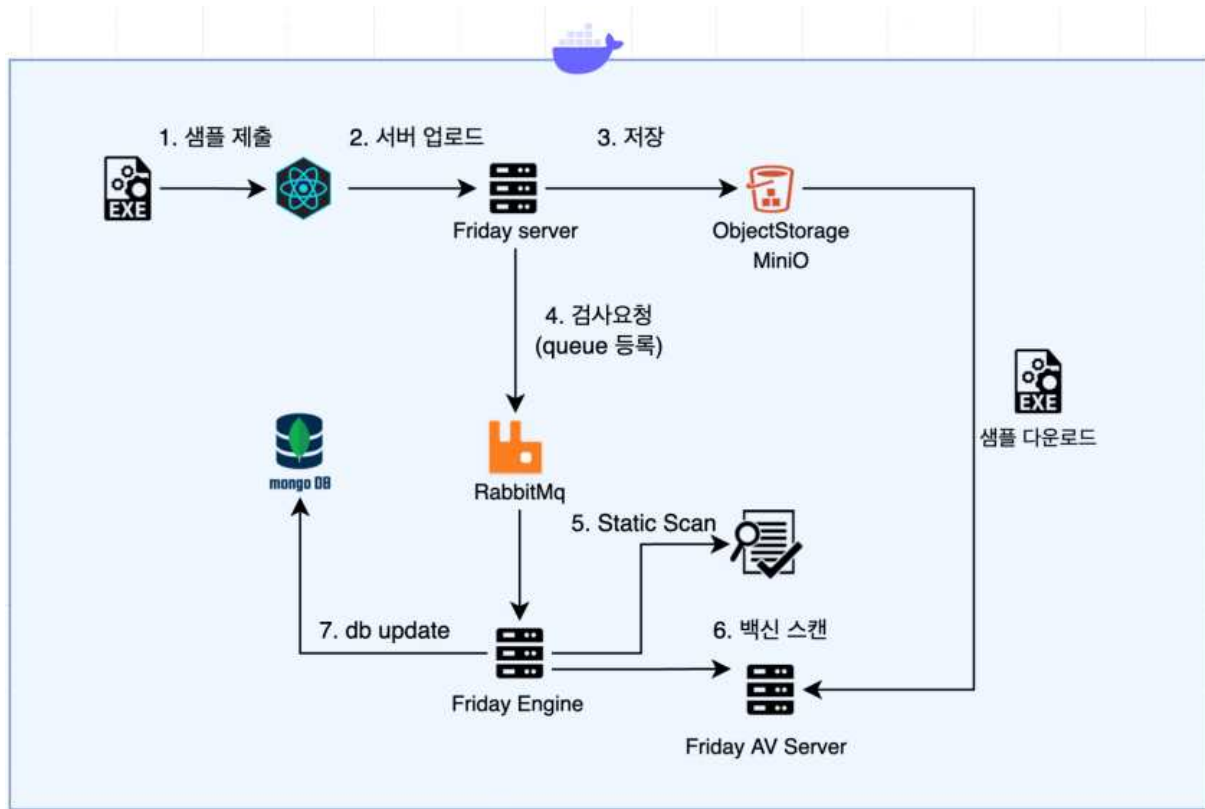
Yara 는 멀웨어를 시그니처 기반으로 판별 및 분류 할 수 있게 하는 도구 이다.  
악성코드의 시그니처는 파일, 프로세스에 포함되어 있는 텍스트 문자열 또는 바이너리 패턴이다. Yara 를 사용하면 텍스트 문자열 혹은 바이너리 패턴으로 Rule을 만들어 그에 매칭되는 파일들을 검색하고 분류 가능 하다.  
단순히 텍스트, 바이너리 패턴 뿐만 아니라 파일이나 프로세스의 오프셋, 가상 메모리 주소 활용 및 정규표현식을 이용해 다양한 rule 을 만들어 효과적으로 패턴 매칭을 실행하여 악성코드를 검색 및 분류한다.

### 2.5 VirtualBox

리눅스, macOS, 솔라리스, 윈도우를 게스트 운영 체제로 가상화하는 x86 가상화 소프트웨어이다. 현재 프로젝트에서 샌드박스 사용하고 있다.

### 3. 본론

#### 3.1 시스템 구성



[그림 1. 시스템 구성도]

#### 3.2 프로그램 구성 및 설명

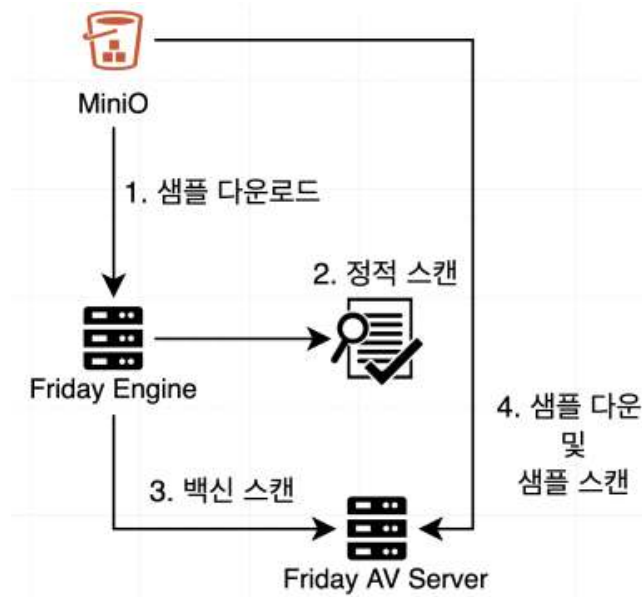
해당 프로그램 (Friday Malware Analysis) 은 Golang 으로 제작되었으며 웹은 React.js 로 개발 되었다.

큰 구성은 WEB (React.js), Friday Server (Rest Api), Friday Engine (StaticScan & gRPC Client), Friday AV Server (gRPC Server) 로 되어있으며, Docker 위에 각각 컨테이너로 구성하여 배포되어있으며, Docker network 로 서로 통신한다.

설명하기 편하게 서버, 엔진, 백신 서버로 바꿔 부르겠다.

1. 사용자가 웹에 접속하여 샘플을 제출하게되면 서버 에서 해당파일을 MiniO 에 업로드를 하게된다. MiniO는 각 서버에 공유 폴더라고 보면 편하다.
2. 업로드가 끝나게되면 서버에서 RabbitMq 에 작업 메세지 큐를 보낸다.
3. 엔진에서 메세지 큐를 소비하게되면 작업이 기본작업이 시작되게된다.

엔진에서는 기본적으로 악성코드 파일을 해시화를 시키며, 파일의 메타정보와 Strings 를 파싱하게된다.



[그림 2. 분석 과정]

[그림 2] 의 정적스캔은 파일을 해시화 (sha-1, sha-256, sha-512, crc32, ssdeep... 등) 및 다음과 같은 정보를 파싱한다.

Exif (linux file meta parser) 파일의 정보를 파싱 하며, Packer (DiE engine) PE의 패커를 파싱 한다. Strings 의 경우 현재 다음과 같은 방식으로 추출 중이다.

```
(s *Server) defaultScan(path string, n int) ([]string, error) {
    b, _ := ioutil.ReadFile(path)
    return GetASCIIStrings(b, n)
}
```

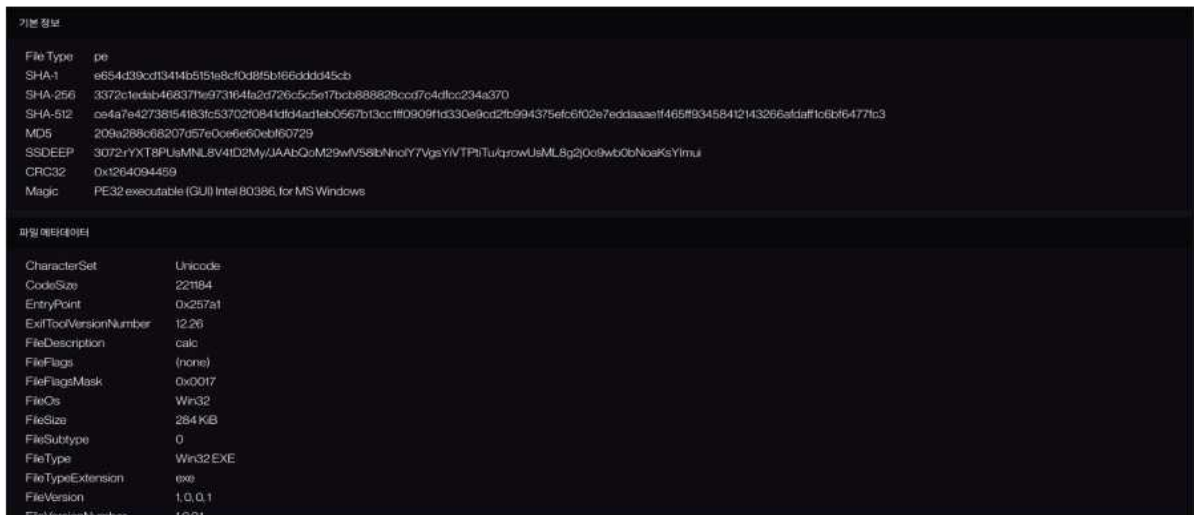
```
func GetUnicodeStrings(data []byte, n int) []string {
    StringRegex := fmt.Sprintf("(?:[ ~][\x00]){%d,}", n)
    re := regexp.MustCompile(StringRegex)
    unicodeStrings := re.FindAllString(string(data), -1)
}
```

```
func GetASCIIStrings(data []byte, n int) []string {
    StringRegex := fmt.Sprintf("[ ~]{%d,}", n)
    re := regexp.MustCompile(StringRegex)
    asciiStrings := re.FindAllString(string(data), -1)
    return asciiStrings
}

func GetUnicodeStrings {
    _ := decodeUTF16([]byte(str))
    d(s, decoded)
}
```

[그림 3. strings 추출 과정]

[그림 3] 을 간략하게 설명하면 파일을 바이트 형식으로 읽은 후 (ioutil.ReadFile) 각각의 Regex 로 ASCII 와 Unicode string 들을 읽어온다. 마지막으로 Yara rule 을 이용하여 파일의 바이너리패턴, 텍스트 문자열등으로 악성코드 인지 판별하게 되면 정적 스캔이 종료된다. 이후 [그림 2]의 백신 서버로 백신 스캔 요청을한 뒤 결과를 취합해 DB에 저장한다.



[그림 4. 정적 스캔으로 추출 된 정보들 ]

백신서버 의 경우 직접 백신들을 구축해준 Docker Container 이다. 현재 구축된 백신 서버들은 다음과 같다.

[comodo, window defender, drweb, clamAV]



[그림 5. 백신 서버 응답 값]

[그림 5] 는 각 Av Engine 들의 Response 이다.

comodo	Malware	drweb	Trojan.AVKill.36635
windefender	Ransom:Win32/Tescrypt.B	clamav	발견 되지 않음

[그림 6. 탐지 된 악성코드].

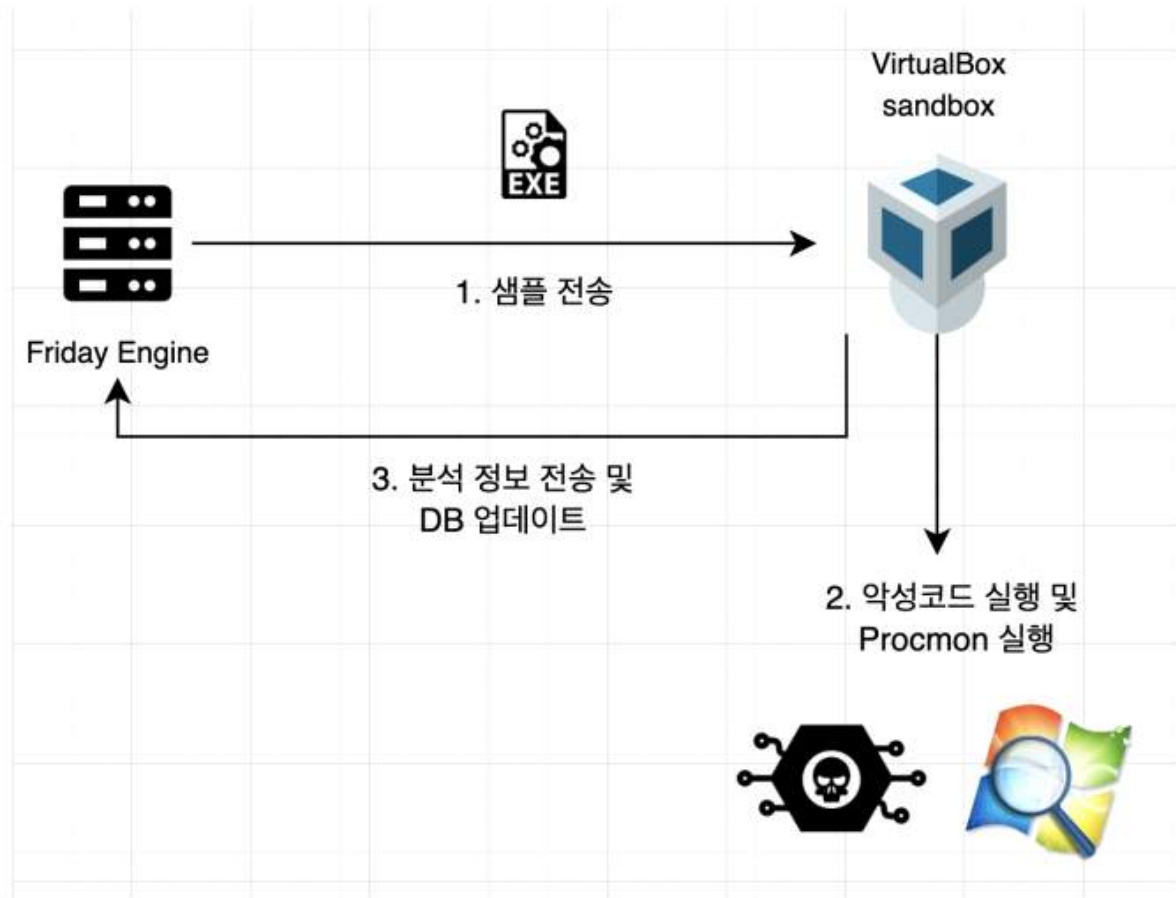
```

_id: ObjectId("61546cef2419b172d19e63ad")
filekey: "51B4EF5DC9D26B7A26E214CEE90598631E2EAA67"
md5: "6e080aa085293bb9fbdcc9015337d309"
sha1: "51b4ef5dc9d26b7a26e214cee90598631e2eaa67"
sha256: "9b462800f1bef019d7ec00098682d3ea7fc60e6721555f616399228e4e3ad122"
sha512: "4e173fb5287c7ea8ff116099ec1a0599b37f743f8b798368319b5960af38e742124223..."
ssdeep: "6144:xy+als+0nIycigV5cbEo6dZbB0DPisjQ/UFsYW:xy+aCFnIycigVSb0bB0DTMud"
crc32: "0x2890568138"
magic: "PE32 executable (GUI) Intel 80386 (stripped to external PDB), for MS W..."
size: 263680
exif: Object
  CodeSize: "39424"
  MachineType: "Intel 386 or later, and compatibles"
  PeType: "PE32"
  SubsystemVersion: "4.0"
  FileType: "Win32 EXE"
  ImageFileCharacteristics: "No relocs, Executable, No line numbers, No symbols, 32-bit, No debug"
  Subsystem: "Windows GUI"
  LinkerVersion: "2.24"
  UninitializedDataSize: "3072"
  FileSize: "258 KiB"
  OsVersion: "4.0"
  FileTypeExtension: ".exe"
  InitializedDataSize: "262656"
  EntryPoint: "0x12c0"
  ImageVersion: "1.0"
  ExifToolVersionNumber: "12.26"
  MimeType: "application/octet-stream"
tags: Object
packer: Array
firstsubmission: 2021-09-29T13:41:03.692+00:00
lastsubmission: 2021-09-30T02:52:38.040+00:00
lastscanned: 2021-09-29T13:41:08.197+00:00
submissions: Array
submissionscount: 4
strings: Array
multiav: Object
status: 2
pe: Object
histogram: null
byteentropy: null
type: "pe"
yara: Array
  0: "Ransom_TeslaCrypt_2"

```

[그림 7. MongoDB document]

[그림 7] 은 MongoDB 의 기본적인 정적 스캔이 완료되었을 때 모습이다. Yara rule 의 경우 현재 내가 보유하고있는 rule 에 대해서만 검사하기 때문에 가지고 있지 않은 패턴 이라면 탐지명(카테고리)를 추출해 줄 수 없다. 기본적인 정적 스캔이 끝나게 되면 VirtualBox 로 샘플을 넘기게 된다.



[그림8. 샌드박스 전송 및 분석]

[그림 8] VirtualBox에서 샘플을 받게되면 악성코드를 실행하게되며, 실행과 동시에 Process Monitor 를 실행해 해당 악성코드의 행동 패턴을 분석한다. 분석한 결과를 DB 형식에 맞게 변환 후 DB 에 업데이트하게된다. 프로세스 캡처과정에서 파일 생성/수정/삭제 레지스트리 읽기/쓰기 네트워크 input/output 이 캡처된다. 해당 캡처된 데이터는 일부 노이즈가 발생하게되는데 이 노이즈는 정상적인 프로세스 행동 과 악성코드의 행동이 섞여서 발생하게되는데 현재로써는 악성코드의 Pid 와 Child Pid 기반으로 백엔드 에서 1차로 분류하고 프론트엔드에서 2차로 데이터 변환 과정을 거친다.



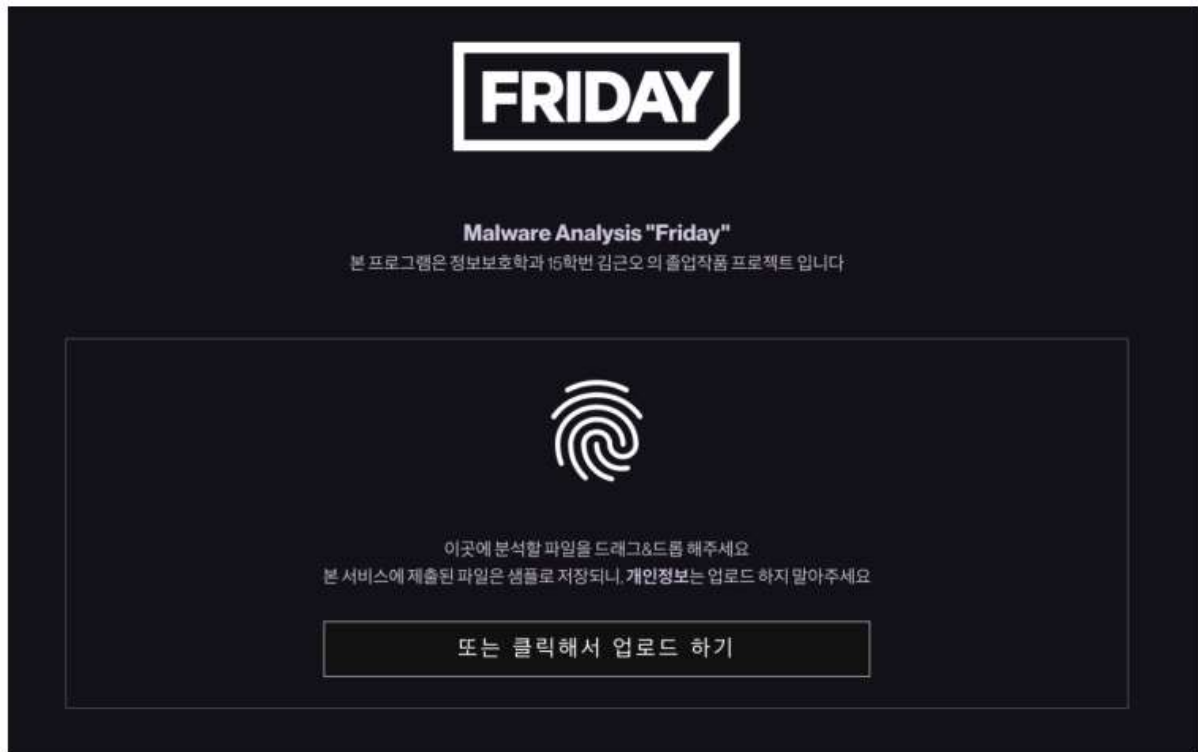
## 4. 결론

### 4.1 결론

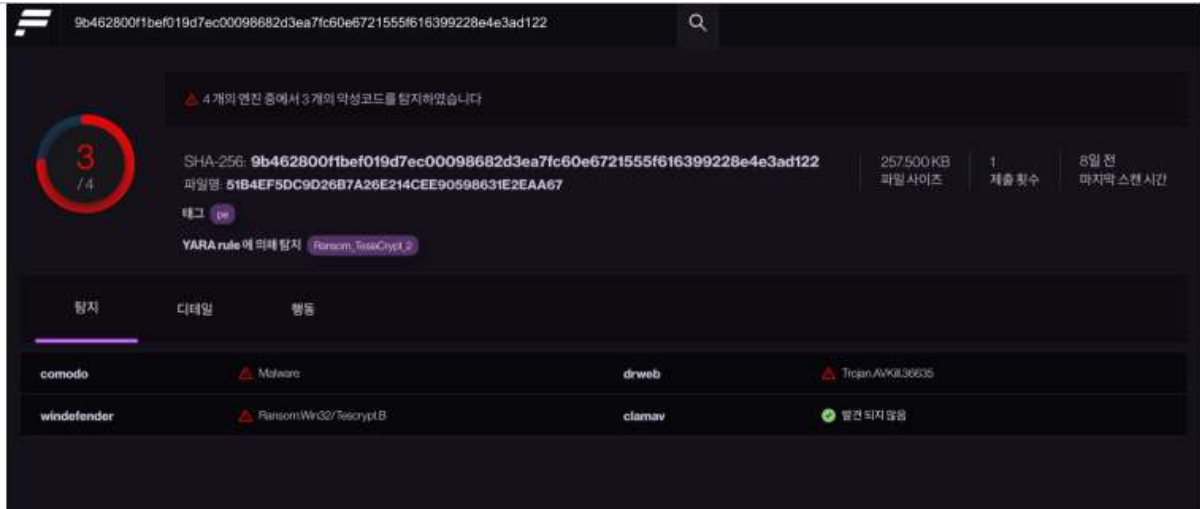
Malware Analysis Tool 을 제작하면서 악성코드를 어떤식으로 분석하는지 알수있는 계기가 되었으며, 대규모 서비스를 구성하면서 마이크로서비스가 왜 중요한지에 대해서 알게되었다.

## 5. 별첨

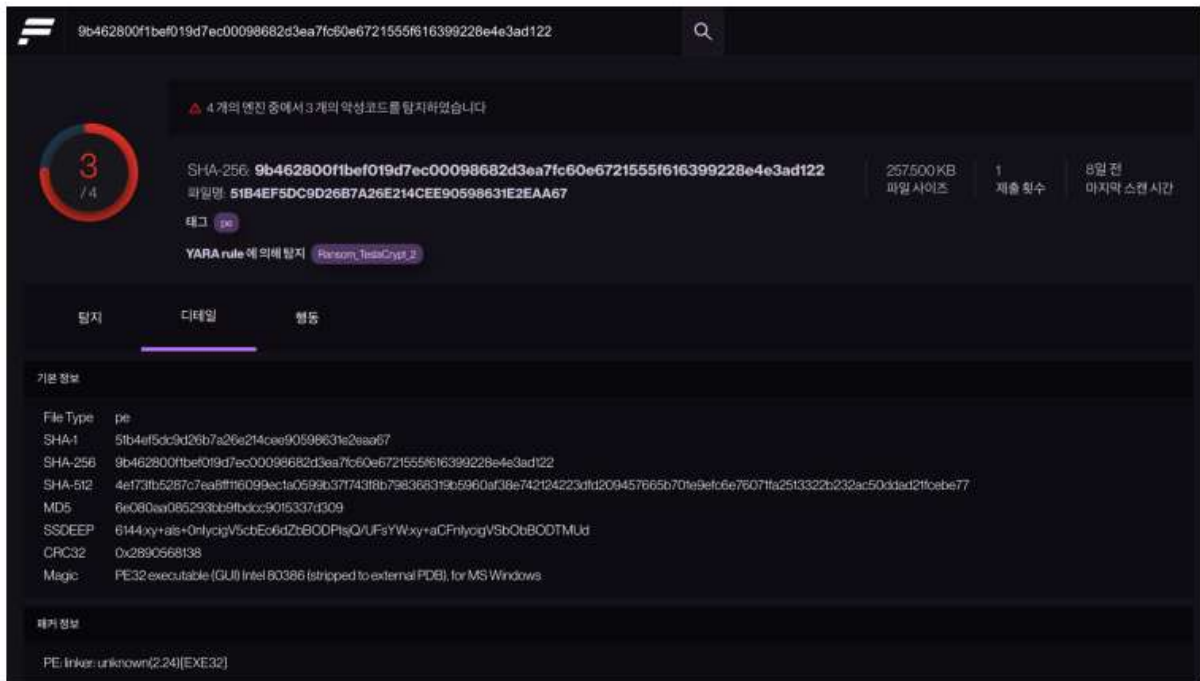
### 5.1 데모 스크린샷



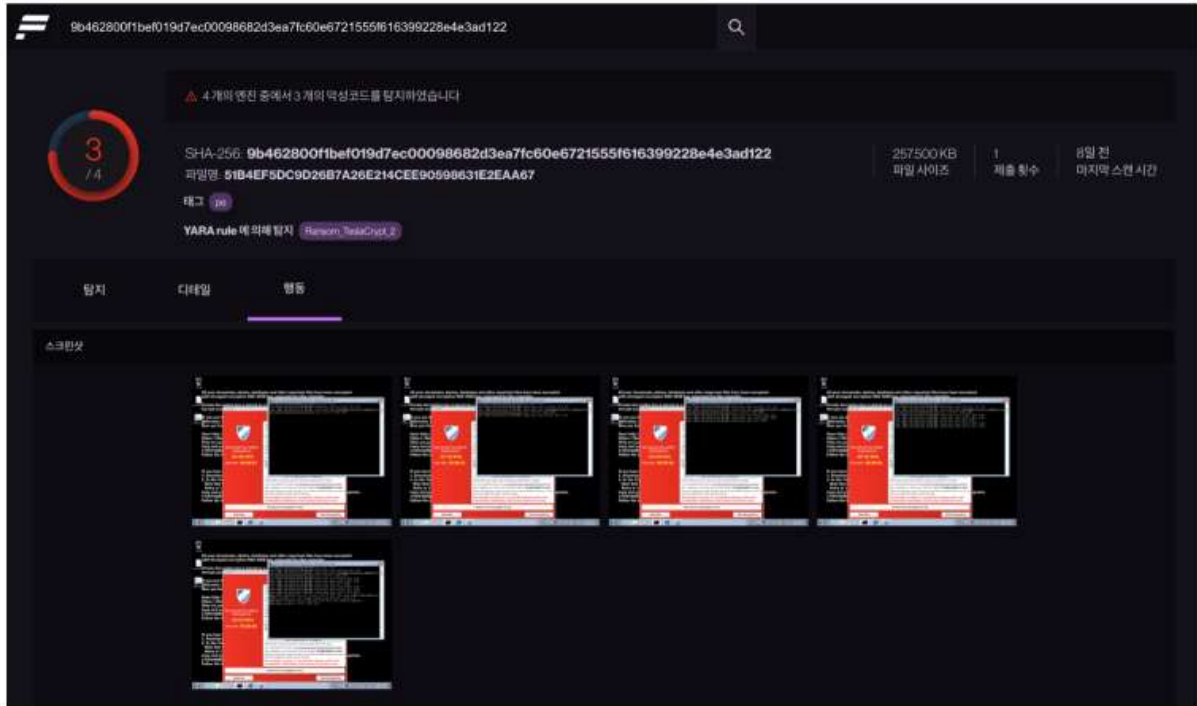
[그림9. 샘플 제출 화면]



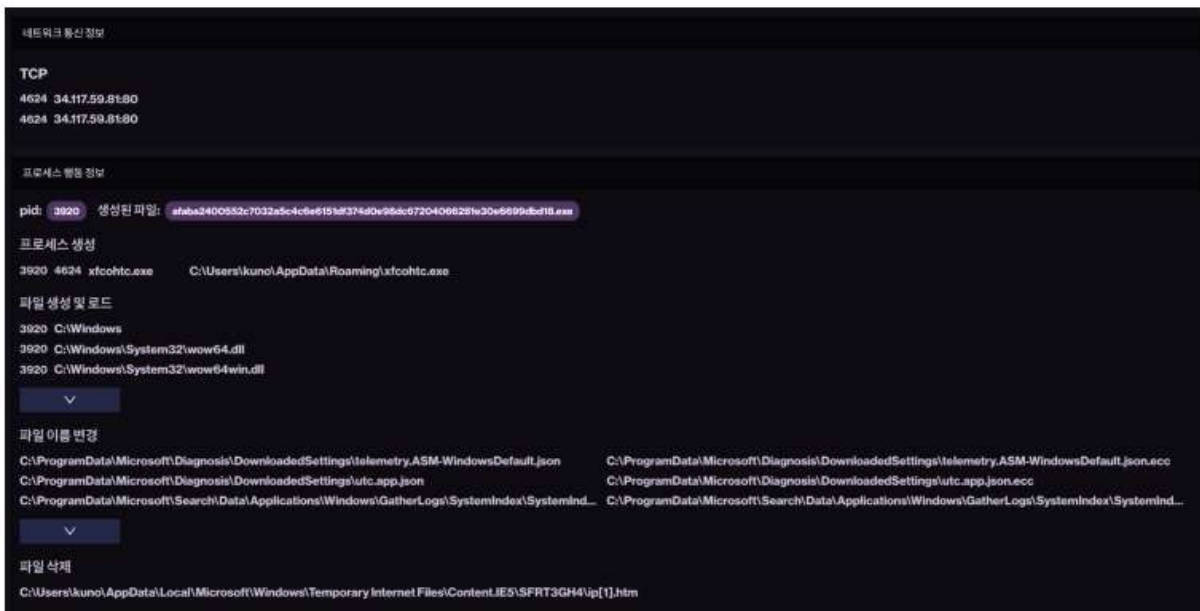
[그림10. 백신 결과 화면]



[그림11. 정적 스캔 결과 화면]



[그림12. 동적 스캔 결과 화면 (1)]



[그림13. 동적 스캔 결과 화면 (2)]

## 5.2 소스코드

<https://github.com/kuno989/friday>

[https://github.com/kuno989/friday\\_agent](https://github.com/kuno989/friday_agent)

[https://github.com/kuno989/friday\\_broker](https://github.com/kuno989/friday_broker)

<https://hub.docker.com/r/kuno989/comodo>

<https://hub.docker.com/r/kuno989/clamav>

<https://hub.docker.com/r/kuno989/windefender>

<https://hub.docker.com/r/kuno989/drweb>

## 5.3 발표자료

# 악성코드 분석 웹서비스

악성코드 자동화 분석 시스템

팀: 시큐리티  
팀원: 김근오 [91514660]  
지도교수: 이병천

중부대학교



## 목차

- 조원 편성
- 주제 선정
- 추진 경과
- 구상도
- 개발 환경 및 개발 내용
- 개발 시스템 운영
- 결론 및 기대 효과

## 조원편성

이름	역할
김근오	프론트엔드, 백엔드, DB 설계 및 구축, 보고서 및 PPT 작성

## 주제 선정

랜섬웨어 피해자의 83%가 범인들에게 돈 낸다

웹하드와 토렌트에서 함부로 파일 받았다간... 'njRAT' 악성코드 감염된다



"P2P 사이트에서 '오징어 게임' 다운로드한 분들, 이제 큰일 났습니다"

안랩 ASEC 분석팀, njRAT 악성코드 분석자료

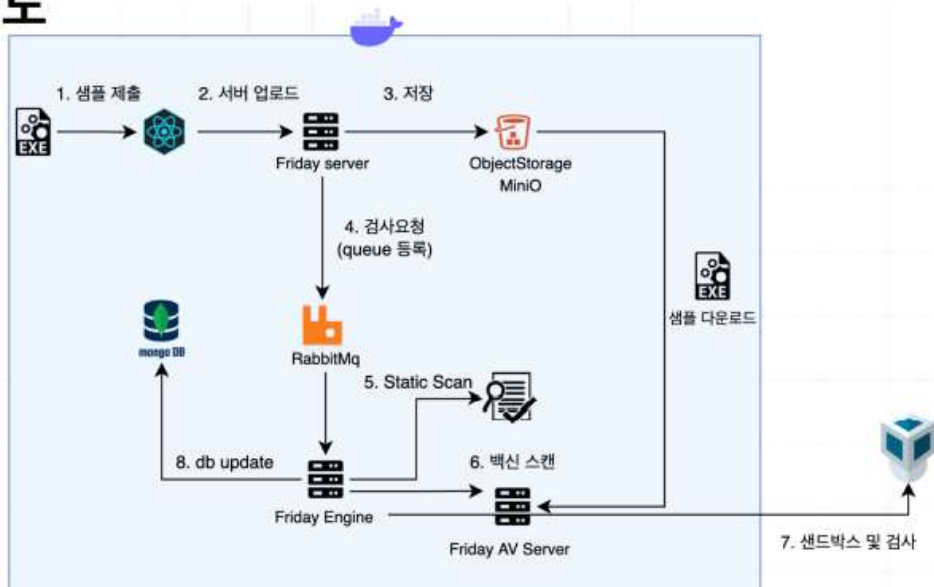
[보안뉴스 원형질 기자] 최근 공격자의 명칭이 오피스 프로그램 공짜로 깔려다 정보 유출...국정원, 감시 나선다  
P2P 사이트에서 불법 다운로드하다 악성코드 감염...개인 정보 유출된다

다양한 경로로 악성코드가 배포되어, 이를 쉽게 검사하기 위한 플랫폼 개발

## 추진 경과

수행업무 / 추진기간 (2021)	4월	5월	6월	7월	8월	9월	10월
자료조사 및 연구	[Progress Bar]						
백엔드 API 설계		[Progress Bar]					
DB 구축 및 프론트엔드 개발			[Progress Bar]				
테스트 및 도커화				[Progress Bar]			

## 구상도



## 개발 환경 및 개발 내용 (1/8)

- Golang
- RabbitMQ
- MiniO
- React.js
- Yara
- MongoDB
- Virtual Box
- Procmon
- Docker

## 개발 환경 및 개발 내용 (2/8)

```
func (m *Mongo) FileSearch(ctx context.Context, sha256 string) (schema.File, error) {
    coll := m.client.Database(m.Config.DB).Collection(m.Config.FileCollection)
    var file schema.File
    err := coll.FindOne(ctx, bson.M{"mongoSchema.FileSha256key": sha256}).Decode(&file)
    return file, err
}

func (m *Mongo) FileCreate(ctx context.Context, uploadFile schema.File) (schema.File, error) {
    coll := m.client.Database(m.Config.DB).Collection(m.Config.FileCollection)
    _, err := coll.InsertOne(ctx, uploadFile)
    if err != nil {
        return schema.File{}, err
    }
    return uploadFile, nil
}

func (m *Mongo) FileUpdate(ctx context.Context, uploadFile schema.File) (schema.File, error) {
    coll := m.client.Database(m.Config.DB).Collection(m.Config.FileCollection)
    _, err := coll.UpdateOne(ctx, bson.M{"sha256": uploadFile.Sha256}, bson.M{"$set": uploadFile})
    if err != nil {
        return schema.File{}, err
    }
    return uploadFile, nil
}

func NewRabbitMq(cfg RabbitMqConfig) (*RabbitMq, func(), error) {
    client, err := amqp.Dial(cfg.URI)
    if err != nil {
        cleanup := func() {
            client.Close()
        }
        return &RabbitMq{
            Config: cfg,
            Client: client,
        }, cleanup, nil
    }
}

func (r *RabbitMq) Channel() (*amqp.Channel, error) {
    ch, err := r.Client.Channel()
    if err != nil {
        return ch, nil
    }
}
```

DB 및 RabbitMQ 선언



## 개발 환경 및 개발 내용 (3/8)

```
func (s *Server) AsyncHandler(msg amqp.Delivery) error {
    body := bytes.ReplaceAll(msg.Body, []byte("x"), []byte("0"))
    var resp rabbitmq.SessionSubject
    if err := json.Unmarshal(body, &resp); err != nil {
        return errors.New("failed to parse message body")
    }
    if !resp.Msg.Reject("reason failed"); err != nil {
        logrus.Errorf("failed to reject message %d", err)
    }
}

filePath := filepath.Join(s.Config.TempPath, resp.Sha256)
file, err := os.Create(filePath)
if err != nil {
    logrus.Errorf("failed creating file %d", err)
}

ctx, cancel := context.WithTimeout(context.Background(), 30*time.Second)
defer cancel()
if err := s.Minio.Download(ctx, resp.MinioObjectKey, file); err != nil {
    logrus.Errorf("failed downloading file %d", err)
}

file.Close()
logrus.Infof("file downloaded to %d", filePath)

res := schema.Result{}
res.Status = processing
var buff []byte

// status 0을 0으로 변경 (status 0)

func (s *Server) defaultScan(path string, res *schema.Result) {
    b, _ := ioutil.ReadFile(path)
    result := utils.BytesHashing(b)
    res.Size = int64(len(b))
    res.Sddeep = result.Sddeep
    res.Md5 = result.Md5
    res.Sha1 = result.Sha1
    res.Sha256 = result.Sha256
    res.Sha512 = result.Sha512
    res.Crc32 = result.Crc32
    logrus.Infof("magic file hashing finished")
    magicErr := magic.Scan(path)
    if err != nil {
        logrus.Errorf("magic scan failed with %d", err)
    }

    res.Magic = magic
    packerRes, err := packer.Scan(path)
    if err != nil {
        logrus.Errorf("packer scan failed with %d", err)
    }

    exifErr := exif.Scan(path)
    if err != nil {
        logrus.Errorf("exif scan failed with %d", err)
    }

    res.Exif = exif
}
```

RabbitMq 로 받은 작업 (파일 정적 스캔)

## 개발 환경 및 개발 내용 (4/8)

```
func getSha512(b []byte) string {
    hash := sha512.New()
    hash.Write(b)
    return hex.EncodeToString(hash.Sum(nil))
}

func getCrc32(b []byte) string {
    checksum := crc32.ChecksumIEEE(b)
    hash := fmt.Sprintf("%08x", checksum)
    return hash
}

func getSddeep(b []byte) (string, error) {
    return sddeep.FuzzyBytes(b)
}

func BytesHashing(b []byte) ErrorResult {
    fuzzy, err := getSddeep(b)
    if err != nil || !fuzzy || !sddeep.ErrFileTooSmall {
        log.Fatalf("sddeep get %d", err)
    }

    result := CryptResult{
        MD5:    getMD5(b),
        Sddeep: fuzzy,
        Sha1:   getSha1(b),
        Sha256: getSha256(b),
        Sha512: getSha512(b),
        Crc32:  getCrc32(b),
    }
}

const tools = "/opt/dist/jeg.sh"

func Scan(path string) ([]string, error) {
    args := []string{path}
    result, err := utils.Cmd(tools, args...)
    if err != nil || !result {
        return output(result), nil
    }
}

func output(output string) []string {
    results := []string{}
    lines := strings.Split(output, "\n")
    for _, line := range lines {
        if line != "" {
            results = append(results, line)
        }
    }
    return results
}

tools := []string{"find", "grep", "cat", "ls", "md5sum", "sha1sum", "sha256sum", "sha512sum", "crc32c", "sddeep", "magic", "exif"}

for _, tool := range tools {
    if !utils.IsStringInSlice(tool, tools) {
        continue
    }

    if !utils.IsStringInSlice(tool, tools) {
        continue
    }

    if !utils.IsStringInSlice(tool, tools) {
        continue
    }
}
```

파일 해시화 및 기본 정보 스캔

## 개발 환경 및 개발 내용 (5/8)

```

switch engine {
case "comodo":
    res, err = comodo_client.ScanFile(comodo_api.NewComodoScannerClient)
case "clamav":
    res, err = clamav_client.ScanFile(clamav_api.NewClamAVScannerClient)
case "windefender":
    res, err = windefender_client.ScanFile(windefender_api.NewWinDefenderClient)
case "drweb":
    res, err = drweb_client.ScanFile(drweb_api.NewDrWebScannerClient)
}

if err != nil {
    // ...
}

func (s *Server) parallelAvScan(path string) map[string]interface{} {
    comodoChannel := make(chan avScanResult)
    clamavChannel := make(chan avScanResult)
    winChannel := make(chan avScanResult)
    drwebChannel := make(chan avScanResult)

    go s.avScan("comodo", path, comodoChannel)
    go s.avScan("clamav", path, clamavChannel)
    go s.avScan("windefender", path, winChannel)
    go s.avScan("drweb", path, drwebChannel)

    avScanResults := map[string]interface{}{}
    engineCounts := 4
    count := 0
    for {
        select {
        case comodoResponse := <- comodoChannel:
            avScanResults["comodo"] = comodoResponse
            count++
        case clamavResponse := <- clamavChannel:
            avScanResults["clamav"] = clamavResponse
            count++
        case windefenderResponse := <- winChannel:
            avScanResults["windefender"] = windefenderResponse
            count++
        case drwebResponse := <- drwebChannel:
            // ...
        }
    }
}

```

백신 서버로 악성코드 스캔 요청

## 개발 환경 및 개발 내용 (6/8)

```

func (s *Server) RegisterHandlers() {
    api := s.Group("/api")
    api.GET("/index", s.index)
    api.GET("/system", s.system)
    api.POST("/download", s.malwareDownload)
    api.POST("/start", s.startAnalysis)
}

func (s *Server) startAnalysis(c echo.Context) error {
    var resp schema.ResponseObject
    if err := s.Bind(c.Request().Form); err != nil {
        return err
    }
    file := filepath.Join(s.config.DownloadPath, resp.FileName)
    logrus.Infof("파일 경로: %s", file)
    s.processCapture()
    time.Sleep(time.Second * 5)
    var imageCapture []string
    var objectKeys []string
    path := filepath.Join(s.config.DownloadPath, resp.FileName)
    imageCapture1 := time.Now().Add(time.Second * 10)
    go func() {
        <- imageCapture1.C
        imagePath := s.captureImage(resp.FileName, time.Now())
        imageCapture = append(imageCapture, imagePath)
    }()
    s.startMalware(path)
    logrus.Infof("파일 경로: %s", file)
}

if resp.FileType == "pe" {
    logrus.Infof("파일 타입: %s", resp.FileType)
    vm, err := virtualbox.GetMachine(s.config.VBoxName)
    if err != nil {
        logrus.Errorf("VirtualBox VM not found: %s", err)
    }
    logrus.Infof("VM 이름: %s", vm.GetName())
    logrus.Infof("CPU: %s, RAM: %s", vm.GetCPU(), vm.GetMemory())

    logrus.Infof("VM 상태: %s", vm.GetState())
    if vm.State == "paused" {
        s.VMRestore()
        time.Sleep(1 * time.Second)
        if vm.State == "paused" || vm.State == "saved" {
            vm.State = "running"
            break
        }
    }
    vm.State = "running" || vm.State == "saved" {
        for {
            time.Sleep(1 * time.Second)
            s.VMStart()
            logrus.Infof("VM 이름: %s, 상태: %s", vm.GetName(), vm.State)
        }
    }
}

```

VirtualBox 실행 및 Agent 로 스캔 요청

## 개발 환경 및 개발 내용 (7/8)

```

type DBModel struct {
    ScreenShots []string      json:"screen_shots"
    ProcessCreate []ProcessCreate json:"process_create"
    CreateFile []CreateFile    json:"create_file"
    ReadFile []ReadFile        json:"read_file"
    RenameFile []RenameFile    json:"rename_file"
    DeleteFile []DeleteFile     json:"deleted_file"
    OpenRegKey []OpenRegKey     json:"open_reg_key"
    GetRegKey []GetRegKey      json:"get_reg_key"
    RegCreateKey []RegCreateKey  json:"create_reg_key"
    SetRegValue []SetRegValue    json:"set_reg_value"
    DeleteRegKey []DeleteRegKey  json:"deleted_reg_key"
    DeleteRegValue []DeleteRegValue json:"deleted_reg_value"
    UDP []UDP                  json:"udp"
    TCP []TCP                  json:"tcp"
    MalName string             json:"malware_name"
    MalPid string              json:"malware_pid"
    SubPid string              json:"sub_pid"
}

type ProcessCreate struct {
    PID string json:"pid"
    ChildPID string json:"child_pid"
    ProcessName string json:"process_name"
    ProcessPath string json:"process_path"
    Operation string json:"operation"
}

type CreateFile struct {
    PID string json:"pid"
    ProcessName string json:"process_name"
    ProcessPath string json:"process_path"
    CreatePath string json:"create_path"
}

type ReadFile struct {
    PID string json:"pid"
    ProcessName string json:"process_name"
    ProcessPath string json:"process_path"
}

type RenameFile struct {
    PID string json:"pid"
    ProcessName string json:"process_name"
    ProcessPath string json:"process_path"
    OriginName string json:"origin_name"
    ChangeName string json:"change_name"
}
    
```

프로세스 행동 추적을 위한 Data Struct

## 개발 환경 및 개발 내용 (8/8)

```

const MainPage: React.FC = () => {
    const history = useHistory();
    const { getRootProps, getInputProps, open, acceptedFile } = useDropzone({
        multiple: true,
        maxFiles: 1,
    });

    useEffect(() => {
        acceptedFiles.forEach(async (files: any) => {
            const UploadFileData = new FormData();
            UploadFileData.append('file', files);
            let resp = await axios.post('/api/file', UploadFileData, {
                headers: { 'Content-Type': 'multipart/form-data' }
            });
            let file_key = resp.data.file_key;
            history.push('/file-analysis/' + sha256(file_key));
        });
    }, [acceptedFiles]);

    return (
        <div className={styles.root}>
            <h2>Malware Analysis "Friday"</h2>
            <p>본 프로그램은 정보보호학자 15회만 접근이 가능한 프로그램입니다.</p>
            <div style={styles.message}>
                <p>Malware Analysis "Friday"</p>
            </div>
            <div style={styles.tabs}>
                <div style={styles.tab}>
                    <span>Detect</span>
                </div>
                <div style={styles.tab}>
                    <span>Analysis</span>
                </div>
            </div>
            <div style={styles.tab_content}>
                <div style={styles.tab_content_detect}>
                    <div style={styles.tab_content_detect_title}>
                        <h3>Detect</h3>
                    </div>
                    <div style={styles.tab_content_detect_body}>
                        <div style={styles.tab_content_detect_body_text}>
                            <p>SHA-256 Hash: {sha256}</p>
                            <p>Infected: {infected}</p>
                            <p>Engine Count: {engineCount}</p>
                            <p>Active Index: {activeIndex}</p>
                        </div>
                    </div>
                </div>
                <div style={styles.tab_content_analysis}>
                    <div style={styles.tab_content_analysis_title}>
                        <h3>Analysis</h3>
                    </div>
                    <div style={styles.tab_content_analysis_body}>
                        <div style={styles.tab_content_analysis_body_text}>
                            <p>SHA-256 Hash: {sha256}</p>
                            <p>Infected: {infected}</p>
                            <p>Engine Count: {engineCount}</p>
                            <p>Active Index: {activeIndex}</p>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
}
    
```

프론트엔드

## 개발 시스템 운영

(1/5)

**FRIDAY**

Malware Analysis "Friday"

본 프로그램은 정보보호학과 15학번 김근우의 졸업작품 프로젝트입니다.

이곳에 분석할 파일을 드래그드롭 해주세요

본 서비스에 제출된 파일은 샘플은 지장되지 않으나, 개인정보는 업로드 하지 않아주세요.

또는 클릭해서 업로드 하기

샘플 파일 제출 화면

## 개발 시스템 운영

(2/5)

9b462800f1bef019d7ec00098682d3ea7fc60e6721555f616399228e4e3ad122

4 개의 엔진 중에서 3 개의 악성코드를 탐지하였습니다

3 / 4

SHA-256: 9b462800f1bef019d7ec00098682d3ea7fc60e6721555f616399228e4e3ad122

파일명: 51B4EF5DC9D26B7A26E214CEE90598631E2EAA67

257,500 KB

1

8일 전

파일 사이즈

제출 횟수

마지막 스캔 시간

태그

YARA rule 에 의해 탐지: Ransom, TeacryptB

탐지

디테일

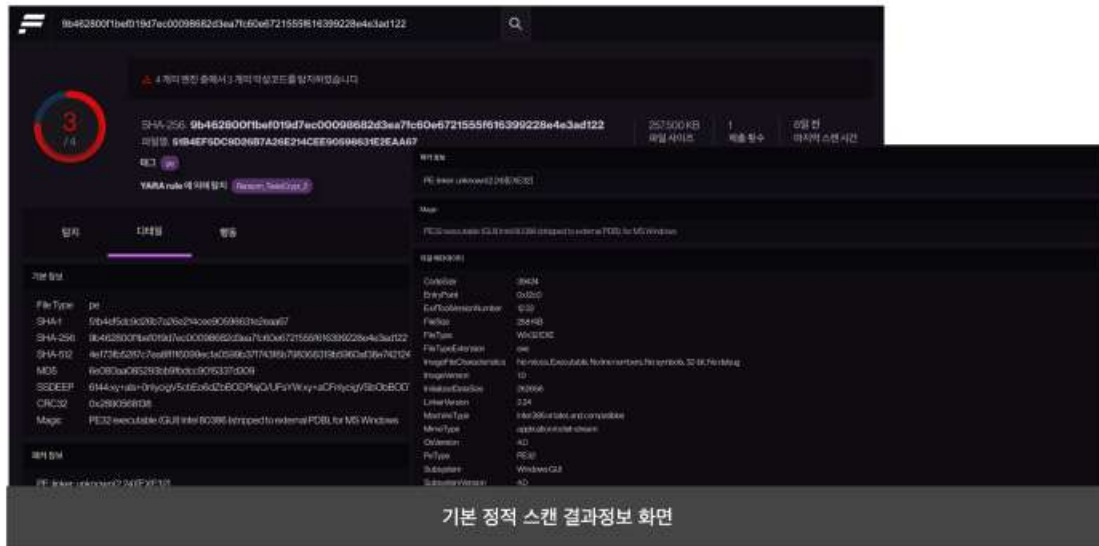
행동

comodo	Malware	drweb	Trojan.A/RN.3605
windefender	Ransom/Wic2r/TeacryptB	clamav	발견되지 않음

백신 검사 결과 제공 화면

# 개발 시스템 운영

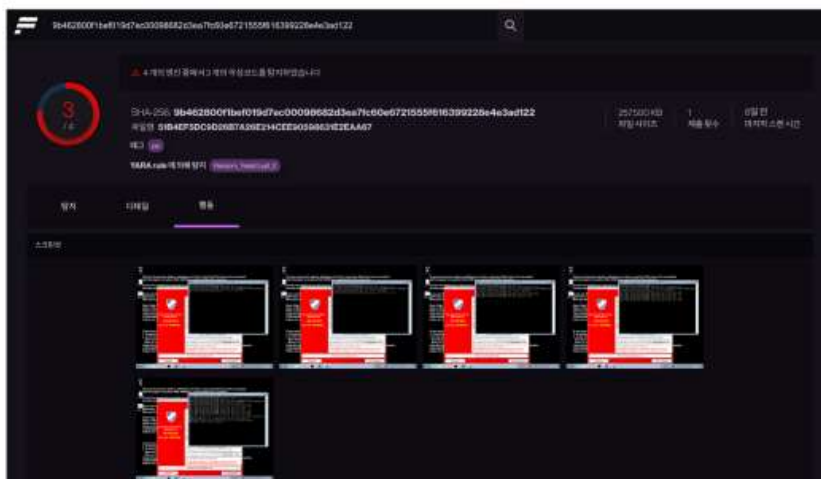
( 3 / 5 )



기본 정적 스캔 결과정보 화면

# 개발 시스템 운영

( 4 / 5 )



VirtualBox 스크린샷 제공 및 악성코드 활동 화면 ( 1 / 2 )

## 개발 시스템 운영

( 5 / 5 )

The screenshot shows a Windows Task Manager window with the following sections:

- 네트워크 활동 정보**
  - TCP: 4034 34.177.59.8180, 4034 34.177.59.8180
- 서비스 활동 정보**
  - pid: 3920, 생성된 파일: C:\Users\kumoi\AppData\Roaming\stcshc.exe
- 프로세스 생성**
  - 3920 4034 stcshc.exe C:\Users\kumoi\AppData\Roaming\stcshc.exe
- 파일 생성 및 로드**
  - 3920 C:\Windows
  - 3920 C:\Windows\System32\wow64.dll
  - 3920 C:\Windows\System32\wow64cpu.dll
- 파일 이름 변경**
  - C:\ProgramData\Microsoft\Diagnosis\DownloadedSettings\telemetry.ASM-WindowsDefault.json C:\ProgramData\Microsoft\Diagnosis\DownloadedSettings\telemetry.ASM-WindowsDefault.json.ecc
  - C:\ProgramData\Microsoft\Diagnosis\DownloadedSettings\utc.app.json C:\ProgramData\Microsoft\Diagnosis\DownloadedSettings\utc.app.json.ecc
  - C:\ProgramData\Microsoft\Search\Data\Applications\Windows\Gather\Log\SystemIndex\SystemIndex... C:\ProgramData\Microsoft\Search\Data\Applications\Windows\Gather\Log\SystemIndex\SystemIndex...
- 파일 삭제**

VirtualBox 스크린샷 제공 및 악성코드 활동 화면 ( 1 / 2 )

## 결론 및 기대 효과

### 결 론

- 백신 및 실제 샌드박스에서 악성코드의 행동을 분석하는 시스템을 구현

### 기대 효과

- 악성코드 분석 및 연구를 할 때 좀 더 쉽게 행동을 추적 가능

**Q&A**

감사합니다