Progetto Internet Security

QRadar® Community Edition

INDICE

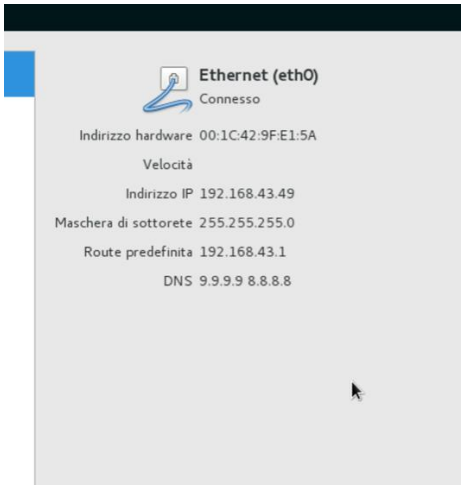
1. Introduzione al progetto

Il software preso in esame è:

IBM Security QRadar 7.3.1 Community Edition.

L'installazione è avvenuta su macchina virtuale tramite l'utilizzo di *Parallels/Virtual Box* usando come SO: *CentOS-7-x86_64-Minimal-1804.iso (7.5). Non è possibile installare QRadar in versioni superiori di CentOS 7.5.*

La prima fase eseguita è stata quella di configurazione della macchina in funzione dei requisiti richiesti da QRadar: si imposta la rete in bridge e si setta un indirizzo ip statico.



Di seguito si procede con la disattivazione di SELinux perché se fosse abilitato causerebbe un reboot continuo della macchina, impedendo di conseguenza l'installazione.

Il tempo di attesa per tutta l'installazione varia dalle 2-4 ore.

Una volta installato, vi si accede al pannello di controllo collegandosi all'indirizzo web: *https://<indirizzoipdellamacchina*>

L'exploit utilizzato è l' *ibm_qradar_unauth_rc*, presente nel framework *Metasploit* e che permette di violare le tre vulnerabilità elencate successivamente andando a sfruttare l'applicazione forense per bypassare l'autenticazione e scrivere un file sul disco. Successivamente effettua un escalation dei privilegi per assumere dei privilegi di root.

2. Funzionamento QRadar Community Edition

IBM Security QRadar colleziona, processa, aggrega e memorizza informazioni di rete in tempo reale in modo da garantire sicurezza nella rete. Inoltre, aiuta i team della sicurezza a individuare e stabilire priorità delle minacce in modo accurato in tutta l'azienda, monitorando ed effettuando varie analisi forensi delle informazioni sulla rete in tempo reale. Tramite un gestore di notifiche,

esso avvisa (in maniera automatica) degli eventuali attacchi così che si possa rispondere rapidamente e ridurre l'impatto degli incidenti. QRadar lavora su due tipi di dati: i dati eventi (Event Data) e i dati flusso (Flow Data). La sua struttura, invece, si basa su tre livelli indipendentemente dalla complessità del campo su cui esso viene reso operativo:

- Data Collection: il primo livello analizza tutte le informazioni acquisite attraverso la rete e le passa al livello successivo;
- Data Processing: tale livello fa passare i dati attraverso il Custom Rules Engine (CRE) che genera avvisi e notifiche e successivamente li memorizza nello storage;
- Data Searches: in questo livello arrivano i dati collezionati e processati da QRadar. Dove è possibile effettuare delle ricerche, analisi, report, avvisi e investigare sugli attacchi.

Vi sono diversi moduli per la piattaforma QRadar come ad esempio QRadarRisk Manager, QRadarVulnerability Manager e QRadarIncidentForensics, QRadarCommunity Edition,QRadarIncidentForensics.

Le versioni che si andranno ad utilizzare sono proprio la QRadar Community Edition e la QRadarIncidentForensics.

3. Analisi delle Vulnerabilità

IBM QRadar SIEM presenta tre vulnerabilità nell'applicazione integrata Web forense che, anche se disabilitata in QRadar Community Edition, permette ancora al codice di funzionare. Una volta concatenate, consentono a un utente malintenzionato di ottenere l'esecuzione di codice remoto non autenticato.

Si divide in tre stadi:

- Il primo stadio ignora l'autenticazione correggendo i cookie di sessione [*BypassAuthentication - CVE 2018 1612*].

- La seconda fase utilizza quei cookie di sessione autenticati per scrivere un file su disco ed eseguire quel file come utente "nobody".
  [*Excute commands as "nobody" - CVE 2018 1418]*

- La terza e ultima fase si verifica quando il file eseguito come "nobody" scrive una voce nel database che fa sì che QRadar esegua uno script shell controllato dall'attaccante come root entro il minuto successivo.

3.1 Bypass Authentication –[CVE 2018 1612]

L'autenticazione in QRadar viene effettuata mediante SEC e QRadarCSFR cookies, che rappresentano una UUID session. Questa viene gestita all'interno della QRadar Console da un Session Manager.

La ForensicsAnalysisServlet, all’arrivo di una richiesta nella quale il richiedente specifica entrambi i tokens da registrare, memorizza i cookies SEC e QRadarCSRF in una HashMapTable temporanea. La funzione doGetOrPost() esegue una serie di azioni come, ad esempio, il recupero di un file dei risultati, la verifica dello stato di una richiesta di analisi, ecc. Il richiedente deve avere i token SEC e QRadarCSRF registrati con il servlet. Per eseguire l'autenticazione per l'azione setSecurityTokens, il servlet verifica se il cookie SEC host.token è stato inviato con la richiesta. Tuttavia, se il parametro forensicsManagedHostIps viene inviato con l'azione setSecurityTokens, doGetOrPost () passerà la richiesta a doPassThrough ()prima di autenticarlo. doPassThrough () convalida anche se la richiesta contiene un cookie SEC valido, a un certo punto. Il problema è che se inviamo l'azione setSecurityTokens, all'inizio della funzione i valori SEC e QRadarCSRF vengono aggiunti alla servletHashMap di token validi prima di essere convalidati.

Dall'ingegnerizzazione inversa del codice, è chiaro che un utente non autenticato può inserire valori SEC e QRadarCSRF arbitrari nei cookie del servletHashMaps. Infine, poter procedere all’autenticazione è necessario che i cookies specificati nella richiesta siano cookies validi. Il controllo, per verificare che essi siano validi, viene effettuato dalla Console confrontando i cookies presenti nella richiesta, precedentemente memorizzati nella HashMapTable temporanea, con i cookies presenti nella HashMapTable dei cookies validi.

Se vi è una corrispondenza allora l’autenticazione va a buon fine.

Provando una fare richiesta al servlet:

```
1. GET /ForensicsAnalysisServlet/?action=someaction HTTP/1.1
2. Cookie: SEC=owned; QRadarCSRF=superowned;
```

si otterrà un errore 403:

```
1. HTTP/1.1 403 Forbidden
```

Viene così inviata la richiesta di aggiungere i valori SEC e QRadarCSRF agli elenchi di token validi. I valori “owned”e “superowned” vengono aggiunti ai token SEC e QRadarCSRF validi:

```
POST /ForensicsAnalysisServlet/?
action=setSecurityTokens&forensicsManagedHostIps=something HTTP/1.1
Cookie: SEC=owned; QRadarCSRF=superowned;
Content-Type: application/json
Content-Length: 44
something1002,something1003,owned,superowned
```

Ottenendo la risposta dal server:

```
HTTP/1.1 200 OK
{"exceptionMessageValue":"javax.servlet.ServletException: No valid
forensics analysis host token data found."}
```

Dunque, a questo punto, i cookies sono stati aggiunti a SECCookiesMap e QradarCSRFCookiesMap, in modo da poter invocare tutte le azioni (anche quelle che richiedono cookie autenticati) in ForensicsAnalysisServlet. Quindi provando a ripetere la richiesta iniziale, si otterranno risultato positivi:

```
1. GET /ForensicsAnalysisServlet/?action=someaction HTTP/1.1
2. Cookie: SEC=owned; QRadarCSRF=superowned;
```

La risposta che si otterrà sarà:

```
HTTP/1.1 200 OK
{"exceptionMessageValue":"javax.servlet.ServletException: No valid
forensics analysis solrDocIds parameter found."}
```

3.2 Excute commands as “nobody”- [CVE 2018 1418]

La seconda vulnerabilità, in questa catena di exploit, si trova nella parte PHP dell'applicazione web forense. L'uso della vulnerabilità “Bypass Authentication”per aggiungere i cookie SEC e QRadarCSRF alle

HashMapsForensicAnalysisServlet, permetterà di invocare qualsiasi funzione nella parte Java dell'applicazione, anche se la parte PHP utilizza uno schema di autenticazione separato che non ha un difetto simile. Tuttavia, accetta qualsiasi richiesta proveniente da localhost senza che sia necessaria l'autenticazione. L'autenticazione viene eseguita nella parte PHP includendo il file DejaVu / qradar_helper.php, che richiama la funzione LoginCurrentUser:

```
public function LoginCurrentUser ($remember, &$errorInfo)
{
        //if local server request don't need to login the user
      if($_SERVER['REMOTE_ADDR'] == $_SERVER['SERVER_ADDR'])
      {
            return true;
      }
}
```

Sfruttando nuovamente ForensicAnalysisServletdoPassThrough (), dopo lo snippet nella vulnerabilità 1, la funzione continuerà a inoltrare la richiesta all'indirizzo o agli indirizzi host immessi nel parametro forensicsManagedHostIps.

Dall'ingegnerizzazione inversa del codice è chiaro che si invierà 127.0.0.1 nel parametro forensicsManagedHostIps e si potrà fare in modo che ForensicAnalysisServlet inoltrerà la nostra richiesta all'applicazione Web PHP ignorando l'autenticazione. Sembrerà, dunque, che le richieste proverranno da localhost.

Nell'applicazione PHP vi è il file.php, che ha una funzionalità "get" la quale consente a un utente autenticato di recuperare determinati file dal filesystem. File.php inoltrerà la richiesta a DejaVu / FileActions.php, che effettuerà alcuni controlli per assicurarsi che il file si trovi in un set limitato di directory:

```
public static function Get()
{
    global $TEMP_DIR, $PRODUCT_NAME, $QRADAR_PRE_URL_PATH;
        $pcapArray = array_key_exists ( 'pcap', $_REQUEST ) ? $_REQUEST ['pcap'] : '';
    $acceptablePaths = array("/store/forensics/case_input","/store/forensics/case_input_staging", "/store/forensics/tmp");
      $docid = array_key_exists('docid', $_GET) ? $_GET['docid'] : '';
       $guitype = array_key_exists('gui', $_GET) ? htmlspecialchars($_GET['gui'], ENT_QUOTES) : 'standard';
       $path = array_key_exists('path', $_GET) ? $_GET['path'] : '';
       if (!empty($path))
       {
            $path = urldecode($path);
            $path = FileActions::validate_path($path, $acceptablePaths);
            if(empty($path))
            {
                QRadarLogger::logQradarError("FileActions.Get(): operation failed");
                return;
            }
       }
...
       if (!empty($docid)) {
          $doc = IndexQuery::GetDocument($docid, $guitype);
          if ($doc) {
             $savedFile = new SavedFile($doc);
             if ($savedFile->hasFile()) {
                if ($savedFile->isLocal())
                   $savedFile->sendFile($guitype);
                else
                   $savedFile->doProxy();
             } else
                send404();
          } else
             send404();

       } else if (!empty($path)) {
          if (file_exists($path)) {
             if (!SavedFile::VetFile($path, $guitype))
                return;
             readfile($path);
          } else
             send404();
```

Inviando un array PHP con diversi parametri pcap, l'applicazione Web comprimerà questi file prima di inviarli:

```
} else if (is_array($pcapArray)) {
    $hostname = array_key_exists('hostname', $_REQUEST) ? $_REQUEST['hostname'] : $_SERVER['SERVER_ADDR'];
    if (count($pcapArray) > 1) {
        $basename = uniqid() . ".zip";
        $zip_filename = $TEMP_DIR . "/" . $basename;
    } else {
        $zip_filename = $pcapArray[0]['pcap'];
        $basename = basename($zip_filename);

    }



    for($i = 0, $j = count($pcapArray); $i < $j ; $i++) {
        $pcapFileList[] = $pcapArray[$i]['pcap'];
    }

    if (count($pcapArray) > 1) {
        // More than one pcap, so zip up the files and send the zip
        $fileList = implode(' ', $pcapFileList);
        //error_log("filename >> ".$filename);
        //error_log( print_r($fileList,TRUE) );
        $cmd = "/usr/bin/zip -qj $zip_filename $fileList 2>&1";
        //error_log("\$cmd =".$cmd);

        $result = exec($cmd, $cmd_output, $cmd_retval);
```

Si otterrà dunque la capacità di eseguire il codice come utente del web server httpd, che è l'utente "nobody" senza privilegi.

È necessario attenzionare la directory in cui verrà scaricato il file. L'utente "nobody" non può scrivere su /tmp, infatti verrà scelto /store/configservices/* sarà utilizzato per varie attività ed è scrivibile da "nobody".

3.3 Privilege Escalation Vulnerability

Per effettuare “Privilege Escalation Vulnerability” si usa uno script Perl presente su QRadar nel percorso /opt/qradar/bin/UpdateConfs.pl che viene eseguito con privilegi di root ogni minuto.

In tale script viene invocata la funzione checkRpm(), che successivamente richiama checkRpmStatus(). Quest'ultima preleverà una table (DB) autoupdate_patch e controllerà che vi siano delle entries ancora da elaborare. Se vi è una entry con estensione .rpm allora verrà invocata la funzione processRpm(), che installerà il pacchetto, altrimenti verrà invocata la funzione installMinor() che eseguirà un "sh -x" [esegui con debug] sulla entry. Queste entries possono essere prelevate nel seguente modo:

```
psql -U qradar -c "select value from autoupdate_conf where key = 'update_download_dir' "
```

Per poter inserire delle entries nella tableautoupdate_patch è necessario che "nobody" possa accedere a tale table, avendo la password di accesso. Una volta ottenuta l'accesso al DB è necessario inserire una entry all'interno della table che punta ad uno script creato ad hoc (ad esempio /store/configservices/staging/updates/script.sh) che verrà eseguito nel minuto successivo con privilegi root.
Dunque, infine, si otterrà una reverse shell remota con privilegi di root.

4. Conclusioni

Nella seguente relazione è stato preso sotto esame il software QRadar Community Edition cercando di effettuare un attacco grazie a delle falle note presenti nella componente QRadarIncidentForensics.

Utilizzando il famoso framework di exploit “Metaspoit” si è riusciti, tramite l’exploit *ibm_qradar_unauth_rc,* ad effettuare un attacco che consisteva nel connettersi e ricevere una shell di root da remoto.

L’analisi ha avuto un esito positivo ed è stato dimostrato che le falle sono ancora presenti.

Infatti, è stato confermato da IBM che le versioni 7.3.0 and 7.3.1 e tutte le versioni rilasciate dalla prima metà del 2014 in poi, sono vulnerabili.

5. About

Giovanni Martucci, X81000188.

Studente presso l'Università di Matematica ed Informatica di Catania