À la recherche du méchant perdu

Eric Leblond
eleblond@stamus-networks.com

Stamus Networks

Résumé. La détection d'intrusion réseau n'est pas un sujet mort [7]. Deux des outils open source les plus populaires, Snort [1] et Suricata [6], analysent le trafic réseau sous le prisme de signatures et alertent quand ils trouvent quelque chose. La source des attaques est habituellement définie comme étant la source du flux déclenchant l'alerte. Ce papier traite des conséquences de ce choix au niveau de l'automatisation de l'analyse et de la recherche des chaines de compromissions.
Il expliquera comment améliorer les signatures d'IDS et quels en sont les bénéfices. Nous utiliserons Suricata pour montrer les gains de la nouvelle approche.

1 IDS réseau : signatures et alertes

1.1 Le langage de signature

Les logiciels de détection d'intrusion réseaux comme Suricata ou Snort sont basés sur des signatures. Une signature exprime un motif dans le trafic réseau dont l'occurence déclenchera une alerte. Ces alertes sont ensuite utilisées pour analyser une attaque potentielle, une violation de la politique de sécurité...

Snort et Suricata utilisent un langage similaire. La signature suivante est un exemple typique :

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (
        msg:"ET SCAN NMAP -f -sX"; fragbits:!M;
        dsize:0; flags:FPU,12; ack:0; window:2048;
        classtype:attempted-recon; sid:2000546; rev:7;)
```

- `alert` : la signature va générer une alerte si les filtres sont validés ;
- `tcp ... any` : paramètres IP des données. Dans cet exemple, les variables `EXTERNAL_NET` et `HOME_NET` sont définies par l'utilisateur et décrivent le réseau surveillé ;
- `msg` : description de l'alerte, ici nous détectons un des scans Nmap ;
- `fragbits ... window:2028` : filtres caractérisant le trafic ;

– `classtype:attempted-recon` : type de l'alerte. Ici, il s'agit d'un essai de reconnaissance ;
– `sid` : identifiant unique de la signature ;
– `rev` : numéro de révision.

1.2 Format des alertes

Historique Le sujet du format des événements a connu une série d'efforts de standardisation. Mais la plupart des projets sont morts ou n'ont pas d'implémentation de référence. De plus, peu de formats sont bien définis et utilisables. Et il y en a encore moins qui soient utilisés pas les IDS open source.

La seule RFC a avoir été implémentée dans Suricata est la norme Intrusion Detection Message Exchange Format (IDMEF) [5]. Mais le standard de facto dans Suricata est la sortie JSON aussi appelé EVE.

Dans le reste de cet article, nous nous focaliserons sur ces deux formats pour décrire comment leurs usages peuvent être améliorés.

Les événements EVE JSON La sortie principale de Suricata utilise le format JSON qui exprime de manière simple et lisible les événements tout en étant facile à étendre. Ce format gère les objets imbriqués ainsi que les types. Et il est de plus très bien supporté par la plupart des outils d'analyse et systèmes de journalisation.

Une alerte Suricata typique se présente comme suit :

```
{
  "timestamp": "2016-12-07T14:17:45.754203+0100",
  "event_type": "alert",
  "src_ip": "191.18.3.4",
  "src_port": 59540,
  "dest_ip": "10.242.4.2",
  "dest_port": 3389,
  "proto": "TCP",
  "alert": {
    "signature_id": 2013479,
    "signature": "fast Terminal Server Traffic, Potential Outbound
        Scan",
    "category": "Misc activity",
  }
}
```

L'alerte contient les paramètres IP ainsi que les informations sur la signature comme le message ou encore son identifiant. Les alertes peuvent aussi contenir les métadonnées protocolaires, comme l'alerte suivante qui comprend les métadonnées de la requête HTTP ayant déclenchée l'alerte :

```
{
  "timestamp": "2017-01-17T09:53:42.511060-0800",
  "event_type": "alert",
  "src_ip": "82.165.177.154",
  "src_port": 80,
  "dest_ip": "10.170.127.169",
  "dest_port": 58146,
  "proto": "TCP",
  "alert": {
    "signature_id": "1234",
    "signature": "Detect secondary executable payload - xbits 2",
  },
  "http": {
    "hostname": "testmyids.com",
    "url": "/exe/testmyids.exe",
    "http_user_agent": "Wget/1.18 (linux-gnu)",
    "http_content_type": "application/x-msdos-program",
    "http_method": "GET",
    "status": 200,
  }
}
```


Ces métadonnées facilitent le travail de l'analyste en améliorant la description du contexte de l'alerte. Dans le cas de l'événement précédent, la criticité est diminuée par l'étude du contexte puisqu'il s'agit en effet vraisemblablement du téléchargement d'un exécutable Windows depuis un système d'exploitation Linux utilisant wget.

Événements Prelude Suricata supporte aussi le format IDMEF grâce à la bibliothèque Prelude [2]. Ce format définit une description des intrusions indépendantes des systèmes capturant l'information. Il supporte des événements provenant des hôtes et du réseau (logs systèmes, IDS réseaux, etc.).

Une spécificité d'IDMEF est l'introduction du concept de `Source` (attaquant) et de `Target` (cible) pour les acteurs de l'événement.

Nous utiliserons la terminologie `Source` et `Target` dans le reste de l'article et nous verrons dans la suite comment une implémentation correcte de cette notion dans l'IDS mène à des résultats intéressants.

2 Trouver Sources et Targets

2.1 Use the source, Luke

Le but principal de l'IDS est de détecter les attaques de manière à protéger les assets. Donc pour une période donnée, il est intéressant de connaitre deux types de données : la liste des `Sources` et la liste des `Targets`.

Une méthode simple pour construire cette liste est de prendre toutes les alertes pour une période et d'extraire les adresses IP contenues dans les événements. Il reste ensuite à l'analyste à identifier dans cette liste les `Sources` et les `Targets`.

En regardant une signature, un bon candidat pour la `Source` semble être l'adresse IP source dans l'événement.

Pour la signature décrite dans la section 1.1, c'est effectivement le cas. L'alerte est déclenchée lorsque qu'un machine externe scanne un système du réseau `HOME_NET` puisque l'on vérifie l'envoi de données spécifiques vers ce réseau. C'est en ligne avec la définition de IDMEF.

Problème : quand l'adresse source est la `Target` L'exemple précédent ne peut malheureusement être généralisé. Considérons la signature suivante :

```
alert http $HTTP_SERVERS $HTTP_PORTS -> $EXTERNAL_NET any (
    msg:"Fast 403 Error Messages, Possible Web Application Scan";
    flow:from_server,established;
    content:"HTTP/1.1 403"; depth:13;
    threshold: type threshold, track by_dst, count 35, seconds 60;
    reference:url,www.checkupdown.com/status/E403.html;
    reference:url,doc.emergingthreats.net/2009749;
    classtype:attempted-recon; sid:2009749; rev:4;)
```

Cette signature recherche une série de retour d'erreur 403 provenant d'un serveur web pour identifier une tentative de scan. L'adresse IP source dans l'événement généré est donc le serveur scanné et c'est par conséquent la `Target`.

Il est donc impossible de définir la `Source` d'une attaque comme étant l'adresse IP source dans une alerte.

2.2 Le cas de la sortie Prelude

Le module de sortie Prelude n'implémente pas correctement la RFC IDMEF car il utilise l'adresse IP source du paquet déclenchant l'alerte comme `Source` au sens IDMEF. Ce qui n'est pas le cas comme nous venons de le voir.

2.3 Sortie EVE

La sortie EVE connait un problème similaire à celle de Prelude avec une utilisation de l'adresse IP source pour le champ source.

Cependant, cela est moins fondamental dans ce cas puisque le format EVE n'a pas de définition explicite de Source et Target. Il n'y a donc pas à proprement parler de violation de spécification. Mais le format EVE n'en offre pas moins aucun moyen d'indentifier qui est la `Target` d'une attaque.

2.4 Visualisation

Il semble donc qu'il n'y ait pas de moyen de trouver `Source` et `Target` avec une approche automatisée. Il est donc intéressant de voir quelles sont les conséquences en terme de visualisation.

Une visualisation nous permettant d'identifier des chaines de compromissions est l'objectif que nous allons nous fixer.

Pour cela, nous allons construire le graphe où les adresses IP sont des noeuds et où les arêtes du graphe relient les adresses IP qui sont les extrémités d'une alerte. Comme `Source` et `Target` ne sont pas identifiés nous avons un graphe non orienté.

Partant d'une adresse IP, nous pouvons utiliser ce graphe pour découvrir les chaines de compromission potentielles. Si un chemin existe entre deux noeuds c'est qu'il est possible qu'une série d'attaques et de compromissions ait permis à un attaquant de prendre, par rebond, le contrôle d'une machine distante sans l'avoir attaquée directement.

Un tel graphe peut être construit[1] en utilisant graph-tool [8]. Pour obtenir la figure 1 nous avons utilisé les données générées pas Suricata sur un jeu de données spécifiques.

Le jeu de données utilisé est une capture au format PCAP réalisée lors d'un exercice à grande échelle d'attaques et défense de réseau informatique.

Le groupe dans le centre semble montrer une longue liste d'adresse IP reliées par des alertes. Ceci pourrait donc être une chaine de compromission. Mais comme le graphe est non orienté une vérification manuelle sur l'ensemble des noeuds est nécessaire pour déterminer si cela est le cas.

3 Amélioration des signatures

3.1 Utilisation du mot clef metadata

Proposition Pour régler le problème, `Source` et `Target` doivent être définies dans la signature. Une solution est d'utiliser un tag fournissant cet indication.

[1] Le code est disponible, voir [3]

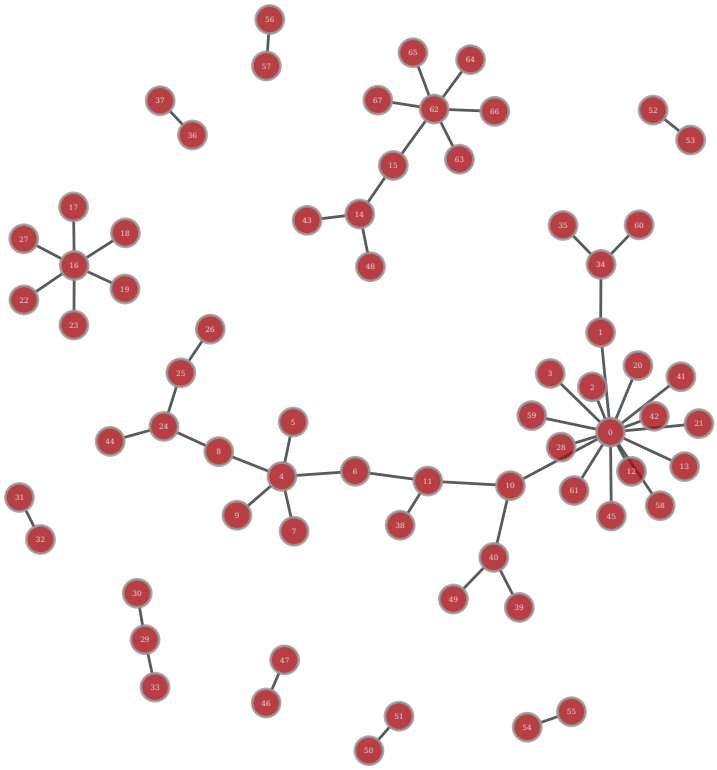

Fig. 1. Graphe non orienté

Cette information doit être compatible avec les versions précédentes pour autoriser la distribution d'un même jeu de règles pour des Suricata différents.

Une option possible est l'utilisation du mot clef `metadata` qui permet la définition dans la signature de clef, valeur arbitraire. Nous proposons d'utiliser la convention suivante :

```
metadata: target [client|server]
```

où `client` et `server` désigne la source de la connexion TCP/IP et respectivement sa destination.

Ainsi, la signature de la section 2.1 devient :

```
alert http $HTTP_SERVERS $HTTP_PORTS -> $EXTERNAL_NET any (
    msg:"Fast 403 Error Messages, Possible Web Application Scan";
    flow:from_server,established;
    content:"HTTP/1.1 403"; depth:13;
    threshold: type threshold, track by_dst, count 35, seconds 60;
    metadata: target server;
    classtype:attempted-recon; sid:2009749; rev:5;)
```

Pour la signature donnée en exemple à la section 1.1, il est logique d'ajouter la ligne `metadata: target client`.

3.2 Implémentation

Implémentation dans Suricata Les modifications pour gérer l'option `target` du mot clef `metadata` ont été implémentées par l'auteur [4]. Une mise à jour des sorties Prelude et EVE a été réalisée.

Événements EVE Les entrées EVE JSON ont été mises à jour pour contenir dans la section `alert` en plus des champs déjà existants deux nouvelles sections nommées `source` et `target` :

```
"alert": {
  "signature_id": 2013479,
  "signature": "fast Terminal Server Traffic, Potential Outbound
      Scan",
  "source": {
    "ip": "198.18.3.4",
    "port": 59540
  },
  "target": {
    "ip": "10.242.4.2",
    "port": 3389
  }
}
```

Le port dans la section `target` est utilisable pour savoir quel est le service impacté sur la cible.

4 Amélioration de la représentation précédente

En utilisant les données présentées dans la partie 2.4, nous avons mis à jour les signatures avec les informations de métadonnées. La graphe de la figure 1 est mis à jour pour avoir un graphe orienté ce qui donne la figure 2.

L'exploitabilité du graphe est améliorée puisque l'on voit clairement que le grand groupe du graphe précédent était en fait une série d'adresses IP sans longue chaine de compromissions. La longueur maximale d'une telle chaine passe de 8 sauts à 3 ce qui limite le travail d'analyse manuel.

5 Conclusion

Les IDS réseaux sont utilisés depuis deux décennies mais oubliaient un point fondamental qui est l'identification des `Source` et `Target`. Ce travail

montre que ceci peut être fait au niveau du logiciel. La difficulté principale est la conversion des jeux de signatures existants. Emerging Threats qui fournit un des principaux jeux de signatures pour Suricata est prêt à lancer la conversion mais avec plus de 20 000 règles à convertir, le travail est colossal.

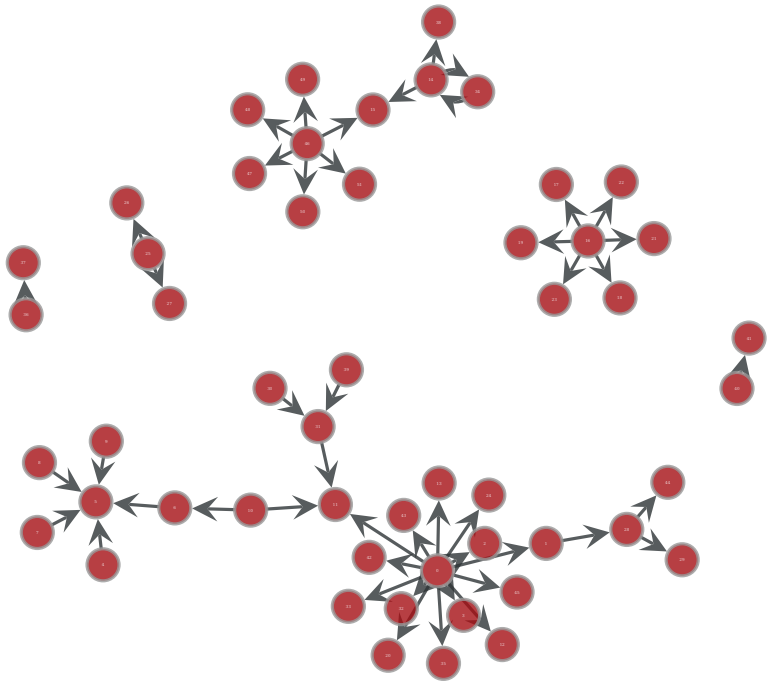

Fig. 2. Graphe orienté

Références

1. Cisco. Snort. https ://www.snort.org/.
2. CS Communication & Systèmes. Prelude SIEM. `https://www.prelude-siem.org/`.
3. Eric Leblond. EVE graph. `https://github.com/regit/eve-graph/`.
4. Eric Leblond. Suricata metadata branch. `https://github.com/regit/suricata/tree/metadata-v3`.
5. H. Debar, D. Curry and B. Feinstein. The Intrusion Detection Message Exchange Format (IDMEF). RFC 4765.
6. Open Information Security Foundation. Suricata. `https://www.suricata.io/`.
7. Richard Stiennon. IDS is dead. Rapport du Gartner group, 2003.
8. Tiago P. Peixoto. Graph Tool. `https://graph-tool.skewed.de`.