Packet Tagging System for Enhanced Traffic Profiling

André Zúquete, Pedro Correia and Hassan Shamalizadeh

IEETA / Univ. of Aveiro Campus Univ. de Santiago
Aveiro, Portugal, 3810–193
Email: andre.zuquete@ua.pt

***Abstract*—This paper describes the design and implementation of a system for managing the tagging of traffic, in order to create detailed personal and applicational profiles. The ultimate goal of this separation is to facilitate the task of traffic auditing tools, namely in their struggle against botnets. The architecture was designed for domestic or enterprise facilities and uses the 802.1X authentication architecture as the base support infrastructure for dealing with unequivocal traffic binding to specific entities (persons or servers). Simultaneously, such binding uses virtual identities and encryption for preserving the privacy and protection of traffic originators from network eavesdroppers other than authorized traffic auditors. The traffic from each known originator is profiled with some detail, namely it includes a role tag and an application tag. Role tags are defined by originators and only partially follow a standard policy. On the contrary, application tags should follow a standard policy in order to reason about abnormal scenarios raised when correlating traffic from several instances of the same application. A first prototype was developed for Linux, using `iptables` and FreeRADIUS and conveying packet tagging information on a new IP option field.**

I. Introduction

The Internet is growing at an unprecedented rate. With its wide spread use amongst all kinds of people, it has become the favourite platform for network criminals and hackers. Botnets are not a new problem [12], but are becoming a frequent and severe security problem [17]. The risks raised by bots are numerous, both for the hosts where they get installed, where they are able to do all sort of illegal actions, but also for organizations attacked by botnets, which are unable to find and persecute the actual attackers screened by the botnet.

A. Motivation

No matter the malicious goal of a bot, it needs to interact with some managing entity, both for getting instructions and for providing information. This has a direct consequence on the organization were the bot was installed: it generates abnormal traffic. Nevertheless, in most cases it may be difficult to clearly identify that traffic as abnormal, because for doing that we need first some notion about what normal traffic is. However, the traffic is changing all the time, mostly because new applications (or plugins/add-ons) are frequently being installed in computers or old applications are patched/upgraded. Therefore, to have a clear notion of what is normal or abnormal, we need to detect traffic variations on particular applications, or caused by new applications.

B. Problem

A network auditing tool (e.g. the network intrusion prevention and detection system Snort [1]) has limited capacities for building profiles of normal traffic, in order to be able to clearly identify abnormal traffic immediately when it appears. In particular, it cannot extract from the inspected traffic some fundamental information that can help to build detailed, personal profiles. Such profiles are critical for implementing a divide-to-conquer strategy. The fundamental information that we are talking about is traffic authorship, i.e., which user, operating system and application is in fact producing the traffic.

Without this notion of authorship, it becomes very hard to build trustworthy traffic profiles. In fact, IP addresses are not a good source for extracting user authorship, as a person may be using several machines and may get different IP addresses each time it establishes a network session. Therefore, we need to be able to extract from packets a stronger notion of user authorship other than the source IP address.

Furthermore, packets only refer the originating host, not the originating application. Transport ports are not helpful, as they can be used by any application. Once again, this information can be vital to build profiles, since a particular traffic may be normal when generated by one application but unusual when generated by another one. Therefore, we need to be able to extract from packets some information about the originating application, in order to improve the accuracy of profiles. With this information, we are able to build per-application profiles instead of profiles per operation system.

C. Contribution

The goal of this paper was to design and implement a solution for facilitating the traffic separation in many different profiles, in order to facilitate its analysis by auditing tools, such as Snort. The ultimate goal of such analysis is to find evidences of the presence of bots. Furthermore, as network users benefit from this system, because they are the initial victims of bots, we conceived a cooperation strategy where users help auditing tools by providing relevant traffic separation hints.

For facilitating such separation, we designed an architecture for managing virtual identities and for tagging the traffic subject to inspection with extra information useful for traffic separation and analysis. The extra information is composed by a virtual identification of the user using the originating host and identity of the originating application. This extra information can help (authorized) network auditing tools to build accurate personal profiles, which we believe could help a great deal on the detection of abnormal traffic and, consequently, on the detection of bots.

Virtual identities are identifiers composed by two types of attributes: entity pseudonym and role. The entity pseudonym is an semantic-free identifier of a person or network server (host) that can be linked to that person or server by specific authentication services. The role is the reason behind the generation of the traffic. For people, the roles may be ordinary user, network administrator, security inspector, etc. User roles

[1] http://www.snort.org

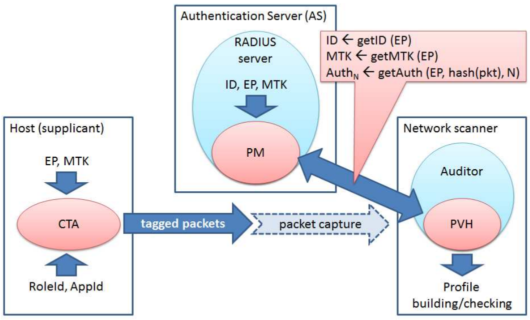


Fig. 1. Proposed architecture for enhanced packet tagging.

may be further detailed to encompass business roles. Roles are also used to separate traffic originated by operating systems from the traffic created by users' applications.

These virtual identities allow auditing systems to improve their accuracy, by providing the means to build more focused network profiles of traffic originators. By using virtual identities, auditors can become insensible to normal network activities that introduce confusion in the mapping of traffic to specific originators. Examples of such activities are dynamic allocation of IP addresses and IP translations performed by NAT boxes.

In this paper, we also studied the feasibility of a real-time traffic tagging and auditing system, that tags each packet of traffic at the originating machine running the profile stamping application. IP packets at present only allow differentiation with protocol-specific identifiers (source and destination addresses/ports, protocol number, etc.). Our differentiation mechanisms allows us to inspect and identify the packets based on the application that generated them, the originating entity and the role it was running.

Additionally, our system enables auditors of the tagged traffic to authenticate the carried tags. This protection is fundamental for preventing profile poisoning attacks, i.e., attacks with fabricated or spoofed tags with the goal of disturbing existing profiles.

Finally, our differentiation tags can also help creating improved application-specific traffic profiles. Accurate application inference is a prerequisite for many network management tasks, such as QoS, accounting and anomaly detection, etc. Service providers commonly infer application classes by means of traffic flow measurements provided by routers [7]. With our system, we do not need to resort to such inference.

II. Architecture

In this section we describe the architecture of our packet tagging system. The system encompasses several components, namely: a Pseudonym Manager, a Client Tagging Application and a Packet Validation Helper. Figure 1 shows the components of our packet tagging system, as well as some interactions among them.

At some initial stage, possibly when a computer joins the network, its owner (or user) initiates a session with a Pseudonym Manager (PM). The goal of this session is to obtain a pseudonym for a network session. For network session we mean period of time where the computer will generate traffic on behalf of its owner/user. Different sessions should yield different pseudonyms, otherwise it could become trivial to eavesdroppers to identify each and every person/service.

Each pseudonym is provided together with a session key. The session key will be used to prove the honesty of packet tags added by the pseudonym holder. In other words, it will be used to prove the origin authentication of the packet tags. Session keys are at start shared only by the PM and the computer that holds the related pseudonym. This allows the PM to act as a validator of packet tags for untrusted third parties. For untrusted third parties we mean network elements/equipments that may be authorized to perform traffic analysis tasks while not being trusted enough for getting access to session keys. For trusted third parties, the PM may convey session keys, in order to offload that costs of tag validation to other hosts.

Hosts connected to the network run voluntarily a Client Tagging Application (CTA). The CTA adds a set of tags to each outbound packet; inbound traffic is not affected or analysed. The packet tags include:

- The entity pseudonym (EP); it would be used to aggregate traffic belonging to a particular entity (person or server), independently of other network identification paradigms (MAC addresses, IP addresses, etc.). Therefore, it will facilitate the task of building personal profiles.
- The identification of the originating application (AppId); it would be used to aggregate traffic belonging to the same application, in the same operating system or not. Therefore it will enable the task of building application's profiles.
- The identification of the application's role (or, in other words, the role of the user exploring the application, RoleId); it will be used to aggregate normal interactions within each role of each person/server. Therefore, it will promote the subdivision of personal profiles in smaller profiles, where anomalies are likely to be detected more easily.

The Packet Validation Helper (PVH) is a library that helps network auditing tools to understand and validate tagged packets. The functionality of this helper is to enable auditing tools to fetch all elements that may be required to properly conduct its activities.

A. Pseudonym Manager (PM)

As previously referred, the PM is the component that keeps bindings between people/servers and their entity pseudonyms (EP). The later are semantic-free numbers, that can only be resolved to identities of people/servers by the PM. Furthermore, the PM keeps a link between pseudonyms and session keys. Finally, auditing applications, using the packet validation helper, interact with the PM to validate packets' tags or to fetch session keys.

Considering all these requirements, the most natural way to deploy a service like the PM is by integrating it with a AAA (Authentication, Authorization and Accounting) service, such as RADIUS [14]. AAA services provide the means to authenticate users (and, indirectly, the hosts they are using), provide the means to authenticate applications (or the hosts where they run) and are usually extensible.

We designed our PM to be integrated with a AAA server (RADIUS) within a 802.1X authentication framework [11], [4]. 802.1X is a scalable, port-based network access control architecture, suitable for authenticating the entities running hosts wishing to attach to a LAN or WLAN.

In the 802.1X jargon, a host wishing to attach to a network is called a supplicant. During the attach process, the supplicant and a central Authentication Server (AS), typically a RADIUS server, authenticate each other using an authentication protocol encapsulated in EAP (Extensible Authentication Protocol [1]).

One of the outcomes of the authentication protocol is a secret key, EMSK (Extended Master Session Key), shared only by the supplicant and the AS.

Our PM was designed as an extension of a AS and using a key derived from EMSK. Such key derivation follows the rules stated in [16], which explains how Usage-Specific Root Keys (USRK) are derived from EMSK. We used the rules of this standard for computing several values, namely pseudonyms and keys (hereafter referred as tagging keys):

$$\begin{aligned}
\text{EP} &= \text{KDF}\,(\text{EAP Session-ID}, \text{“Tagging system”}, \text{EMSK}) \\
\text{MTK} &= \text{KDF}\,(\text{EMSK}, \text{“Master tagging key”}) \\
\text{TSK} &= \text{KDF}\,(\text{MTK}, \text{“Tagging secrecy key”}) \\
\text{TIK} &= \text{KDF}\,(\text{MTK}, \text{“Tagging integrity key”})
\end{aligned}$$

where EP means Entity Pseudonym, MTK means Master Tagging Key, TSK means Tagging Secrecy Key, TIK means Tagging Integrity Key and KDF means key derivation function. According to [16], EP is a *USRKName*, i.e., a name for referring a USRK (in our case, MTK). MTK is our session key, linked with EP; it is a USRK, since it is the root key for our tagging system for each session. The TSK is a key that will be used to provide secrecy to tags, in order to prevent unauthorized eavesdroppers from collecting useful information from them. Finally, TIK is a key that will be used to authenticate tags, allowing authorized auditors to assert their integrity.

The supplicant does the same computations than the PM, reaching the same values for EP and MTK. Thereafter, it will only use EP for tagging outbound traffic and MTK for computing TSK and TIK, which will be used to enforce confidentiality and origin authenticity to traffic tags.

The PM keeps a permanent store with mappings between an EP and the related MTK. It also maintains a mapping between a EP and a descriptor of the entity (person or server). This last mapping is crucial for building entity-related profiles from tagged traffic. Furthermore, for each descriptor the PM must keep a persistent, unique number (ID) for allowing auditing tools to aggregate traffic originated by different EPs without bothering with the details of their real-world identification.

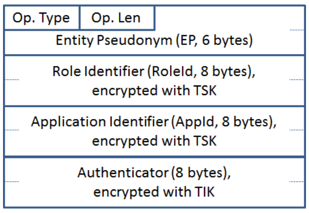
B. Client Tagging Application (CTA)

The CTA is a component that theoretically stays between the host and the network. It is responsible for tagging all outbound traffic with the correct values expected by auditing tools. Note that the CTA does not know if such tools exist; they are transparent for it. Therefore, the CTA simply assumes that they may exist, and as such does its job all the time.

For tagging traffic, the CTA must know the current EP and its MTK. This data is provided by the application that manages the 802.1X supplicant after creating each EAP session. Furthermore, the CTA needs to know the originating application (for fetching its AppId) and the RoleId of the role played by the entity that launched it.

1) Role management and inference: The management of roles is not straightforward, as it raises a new requirement for users that they are not used to. Namely, users have to manage some sort of binding between the applications that run in their machine and a role.

We decided to use the following policy for simplifying the role management workload on Linux hosts with or without graphical interface. An application exploring I/O devices capable of interacting visually with the user (consoles, X terminals, graphical interfaces) is considered a **user application**. Applications other than user applications are considered operating system applications, with the **operating system (OS) role**.



Op. Type | Op. Len
Entity Pseudonym (EP, 6 bytes)
Role Identifier (RoleId, 8 bytes), encrypted with TSK
Application Identifier (AppId, 8 bytes), encrypted with TSK
Authenticator (8 bytes), encrypted with TIK

Fig. 2. Layout of the IP option used to convey our profiling tags

User applications running with a root (0) real UID are considered to be on a **self-administration role**. Note that applications running with elevated privileges due to set-UID mechanisms are not included in this role. User applications with a graphical interface have a role that is given by the virtual desktop where the interface is shown. Non-graphical user applications producing output to a Linux console have a role extracted from an environment variable.

This policy for obtaining the role of an application makes it easier for users to manage roles. In fact, the only workload they have is to (i) create a set of virtual desktops equal to the number of roles they play, (ii) give to each virtual desktop a different, arbitrary role name and (iii) run on each virtual desktop only the applications needed for its role.

The user (entity) has the ability to create as many virtual desktops as he wants in order to clarify the roles he plays. Furthermore, role names are arbitrarily chose by users, there is not a universal policy for choosing names.

On the other hand, users should not go berserk with role names, because that ultimately would introduce entropy in auditing systems. Note that users only gain by collaborating with the tagging system, as it allows auditing tools to detect malicious code running on their machines, namely bots. Therefore, users should maintain a stable set of names for their roles and commit to them faithfully, otherwise they introduce entropy in role-based profiles.

2) Packet tagging: For adding our tags to packets we had to chose a standard way, in order to enable tagged packets to be accepted by all systems. Consequently, we decided to encapsulate tags within a single IP option.

The new IP option will contain the following fields:

- **EP**: 6 byte value, computed as described in Section II-A.
- **RoleId**: 8 byte value. Its 2 high order bits have these values: **00**: OS; **01**: self-administration; **10**: user-defined; **11**: reserved. The other 62 bits are filled with bits from an hash of operating system identity (e.g. as given by the Linux “`uname -sr command`”), for the OS role and for the self-administration role, or the user-provided role name, extracted from the application’s execution environment.
- **AppId**: 8 byte value.
- **Authenticator**: 8 byte value, part of the result of the encryption with TIK of the digest of tags with some parts of the IP packet.

For privacy sake, the role and application fields are encrypted together with TSK, in order to hide information about operating systems and applications used by the originator of the packet. This way, we prevent this useful information to be reveled to others than the authorized network auditing applications.

The computation of the authenticator poses some concerns. On one hand, we would like to bind and authenticator to the entire packet contents, in order to prevent authenticators from being easily copied to other packets. On the other hand, we need to take care with existing network equipments that may change the packet contents transparently for its originator (routers, NAT boxes, etc.). We decided the use the following original packet data for computing the authenticator:

- Destination IP address.
- Modified IP payload. The modification consists in the zeroing of all fields that are likely to be modified by network elements, namely NAT boxes, such as transport source ports (UDP or TCP ports), GRE keys [6], etc.

The total length of an IP option is two bytes more than its optional data. This means that that, in our case, the total length of the tagging option is $2+6+8+8+8 = 32$ bytes (see Fig. 2). This value is intentionally a multiple of 4 bytes, because IP headers must have a length multiple of 32-bit words. It also fits in most IP headers, that have a minimum size of 20 bytes and can grow up to 60 bytes.

C. Packet Validation Helper (PVH)

This component is to be used by authorized auditing tools that need to interpret the IP options field previously described. For doing so, the PVH provides the following functionalities.

First, it allows authorized tools to establish a secure (authenticated) session with an AS (e.g. a RADIUS server). For this goal, it uses the same methods that are normally used by RADIUS clients (Network Access Devices, Network Access Servers, etc.): request/response authentication by means of a secret key, shared with the AS.

Second, the PVH enables an authorized auditor to fetch the key MTK of a given EP from the AS. The purpose of this functionality is to enable the auditor to perform packet validations and inspections using only local resources, i.e. without requiring further help of the AS for that EP.

Third, the PVH enables an authorized auditor to get the decrypted values of the role and application tags given an EP.

Finally, the PVH enables anyone to validate a packet. This is useful for implementing network filters intended to drop invalid packets. The validations are performed both by the auditor and the AS. First, the auditor sends the EP and the packet digest to the AS, which encrypts the later with TIK and returns the relevant part of the result. Then the auditor compares the packet authenticator with the value received from the AS; if there is a mismatch, then the packet authenticator is not valid and the packet is eligible for dropping.

It is worth noting that the authenticator digest should be computed over the encrypted values of the role and application tags, instead of their original values. Otherwise, it would be impossible to validate packets without decrypting these values, which would only create a useless computational overhead without any relevant security gain.

III. Implementation

The CTA was implemented together by a modified supplicant application (`wpasupplicant`) and a new application, the packet marker. The former was written in C, the latter was written in Python. The packet marker is able to run on any Linux machine with a Python interpreter and Gnome with Metacity support – a commonly used window manager that provides the necessary libraries, such as `libwnck`, to get access to details of the graphical interface of running applications.

The PM was implemented as a RADIUS extension. We used the FreeRADIUS distribution and did some modification on the source code to include the PM functionality as an extension to the implemented EAP functionality.

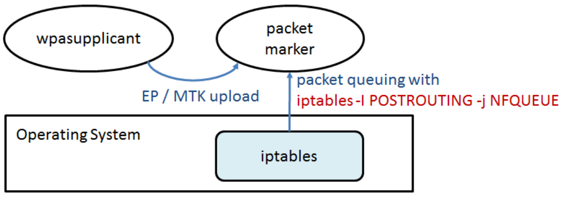


Fig. 3. CTA architecture, using a modified version of `wpasupplicant`, a new packer marker application and the `iptables` queueing facility.

The PVH was implemented as a Linux library, written in C++. It allows a network auditor to interact with the PM using new EAP messages encapsulated in RADIUS messages. For handling RADIUS messages we used the ACE RADIUS library, a portable, open-source C++ implementation of the RADIUS protocol.

A. CTA implementation details

The CTA architecture is presented in Fig. 3. The CTA packet marker intercepts all outbound traffic using the `iptables` queueing facility. To manipulate and dispatch queued packet, the application uses the Scapy Python library [3].

The packet marker gets the values of EP and MTK (check Section II-A) from the local `wpasupplicant`. The later was modified to upload these values to the marker through a UNIX socket. Each time the `wpasupplicant` negotiates a new EAP session, new pseudonyms and master keys are uploaded into the packet marker. Immediately after their upload, the packet marker computes the derived keys TSK and TIK.

The marker needs to infer a RoleId and an AppId only from the packet data. The base strategy for RoleId inference was already presented in Section II-B1; here we explain how we do it starting from each packet contents:

1) From the source IP address and source transport ports, and using the `proc` file system, the marker maps a packet to an origination process identifier (PID). Currently, we only handle UDP and TCP packets; all other packets are not marked.
2) From the originating PID, the marker checks if it is using directly or indirectly a graphical user interface. This is done by querying the graphical window manager. If it has, then it is a user process and, by default, its role is going to be extract from the desktop name where the interface is.
3) From the `proc` file system, the marker checks if the process is using a console (tty device). If so, then it is also a user process and, by default, its role is going to be extracted from the process environment variables (again from `proc` file system).
4) From the `proc` file system, the marker checks if the user process has a real UID of 0. If so, the role is overruled to Self-Administration; otherwise, we use the role obtained with the previous rules.
5) Otherwise, the role is Operating System.

The AppId is currently extracted from the `exe` file of the `proc` file system tree of a process (“`/proc/[PID]/exe`”). Since AppId is limited to 8 bytes, we use the process' file name, either truncated to 8 bytes or null padded.

In the rest of this section we will detail some aspects of the role inference process that was briefly describe above.

B. Packet to process PID mapping

The packet marker keeps a table of known mappings between source transport ports and PIDs. Each time a new source port appears, the marker iterates over `proc` file system, namely in the per-process subdirectories and `net/tcp` and `net/udp` directories, to produce a new table of mappings.

Since different applications may use the same transport port over time, between different updates of the table, table entries are validated before being used. Namely, we verify if the process still exists and is still using the target transport port. The detection of an invalid mapping triggers a complete update of all mappings.

1) Desktop role extraction: The names of desktop workspaces can be configured in runtime using Metacity GUI for desktop management. As an example, the workspace 1 from Gnome window manager can be called “Fun”, Workspace 2 “Work”, and so on, without any limitations on the number of roles that a user can/want to have.

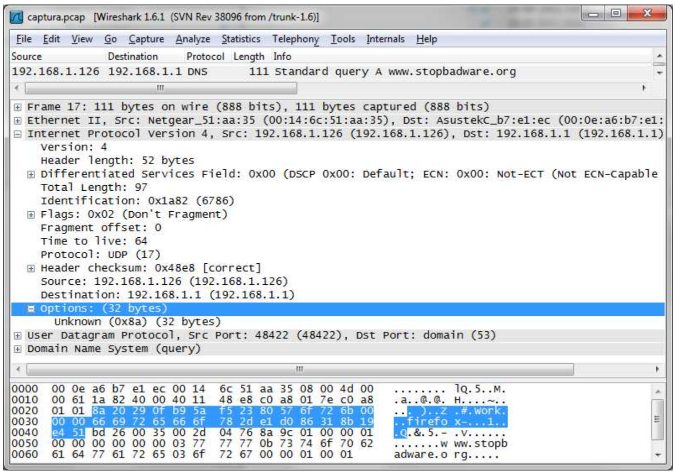
To extract the workspace where an application interface lies, the marker application used the `libwnck` library. This library allows the marker to list workspaces, to iterate over each workspace and find the PID of each application that has an interface object on the workspace. For our purpose, them marker needs to check if a given PID has an interface object of type WnckWindow.

In fact, the marker does not interact with the windows manager each time it needs to mark a packet, because that would be too costly. Instead, the marker keeps an updated list of all PIDs that currently have a WnckWindow object. To keep the list updated, the marker catches relevant window manager events, namely window opening and closing events.

2) Detection of console applications: Console applications running on Linux console use devices `/dev/tty` devices for input or output. The processes used by these applications are detected when the `proc` file system is iterated looking for UDP or TCP sockets and their PID is recorded in a separate table, together with the role extracted from the ROLE environment variable (using the `proc` file system). If this variable is not defined, a warning message is written in the `/dev/tty` console of the process.

3) Packet marking: Once identified the RoleId and AppId of a packet, the packet marker creates the new option that will carry them in the packet. We used the free option number 88_H (Copied Flag=1, Option Class=0 and used copied into RoleId. Option Number=8). The role string is first hashed with MD5 [15] and the low-order 62 bits are copied into the low-order 62 bits of RoleId. Then, RoleId and AppId are encrypted with TSK, using AES-128 [9], and the cleartext authenticator is computed with MD5 over the destination IP address, the new IP option (excluding the authenticator) and the IP payload (excluding the source transport ports); The authenticator is then encrypted with TIK and AES-128 and the least significative 8 bytes are inserted in the IP option. At the end, the IP checksum is recomputed; transport checksums are maintained because they are not affected by IP options.

Figure 4 shows the IP options field of a marked DNS (UDP) packet originated by a Linux host running our CTA. Just for demonstration, we did not encrypt the RoleId and AppId fields, otherwise the entire option value would be incomprehensible. Furthermore, we did not hash the role name to generate the low 62 bits of RoleId; instead, we filled its 7 low-order bytes with the first characters of the role. In the displayed example, we can see that the RoleId is 0x80 (user-defined role) concatenated with “Work” and the AppId is “firefox-”, the 8-byte truncated result of “firefox-bin”.


captura.pcap [Wireshark 1.6.1 (SVN Rev 38096 from /trunk-1.6)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

```
Source        Destination  Protocol Length Info
192.168.1.126 192.168.1.1  DNS         111 Standard query A www.stopbadware.org

⊞ Frame 17: 111 bytes on wire (888 bits), 111 bytes captured (888 bits)
⊞ Ethernet II, Src: Netgear_51:aa:35 (00:14:6c:51:aa:35), Dst: AsustekC_b7:e1:ec (00:0e:a6:b7:e1:
⊟ Internet Protocol Version 4, Src: 192.168.1.126 (192.168.1.126), Dst: 192.168.1.1 (192.168.1.1)
    Version: 4
    Header length: 52 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable
    Total Length: 97
    Identification: 0x1a82 (6786)
  ⊞ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: UDP (17)
  ⊞ Header checksum: 0x48e8 [correct]
    Source: 192.168.1.126 (192.168.1.126)
    Destination: 192.168.1.1 (192.168.1.1)
  ⊟ Options: (32 bytes)
      Unknown (0x8a) (32 bytes)
⊞ User Datagram Protocol, Src Port: 48422 (48422), Dst Port: domain (53)
⊞ Domain Name System (query)

0000  00 0e a6 b7 e1 ec 00 14  6c 51 aa 35 08 00 4d 00   ........ lQ.5..M.
0010  00 61 1a 82 40 00 40 11  48 e8 c0 a8 01 7e c0 a8   .a..@.@. H....~..
0020  01 01 8a 20 29 0f b9 5a  f5 23 80 57 6f 72 6b 00   ... )..Z .#.Work.
0030  00 00 66 69 72 65 66 6f  78 2d e1 d0 86 31 8b 19   ..firefo x-...1..
0040  e4 51 bd 26 00 35 00 2d  04 76 8a 9c 01 00 00 01   .Q.&.5.- .v......
0050  00 00 00 00 00 00 03 77  77 77 0b 73 74 6f 70 62   .......w ww.stopb
0060  61 64 77 61 72 65 03 6f  72 67 00 00 01 00 01      adware.o rg.....
```

Fig. 4. DNS packet marked by our CTA; the IP option with the added tags is highlighted. The RoleId (80_H concatenated with “Work”) and AppId (“firefox-”) fields are in cleartext to facilitate their visualization. Furthermore, the RoleId is not an hash of some value but a direct copy of the 7 first characters of the role name.

C. PVH implementation details

The interaction between PVH and PM uses RADIUS Access Request/Accept/Reject messages, authenticated using a shared secret key. Access Request messages contain an EAP-Message attribute, encoding the called function and its parameters, and a Message Authenticator attribute. Access Accept/Reject messages contain an EAP-Message attribute, encoding the results, and a Message Authenticator attribute. These interactions follow the guidelines for RADIUS EAP extensions [13], [2]. The EAP message attribute on a RADIUS Access Request message will trigger the activity of the RADIUS EAP component, which will thereafter be responsible for handling the request and produce the results.

Our EAP message attribute is a standard EAP message [1] using a free EAP type (7, the first non assigned type). Since this is a new type, we were free to chose the appropriate way to pack the parameters and results conveyed in EAP request/response messages (in the Type-Data field). In the request, the first Type-Value byte specifies the operation and, immediately after, we have the parameters or results.

D. PM implementation details

We implemented PM as an extension of the FreeRADIUS EAP functionality. Each EAP method should generate EMSK on its own particular way, but most of them do not do it currently. Nevertheless, the EAP methods that use TLS [5], such as EAP TLS [18] and PEAP, already produce internally an EAP-EMSK value-pair (pair of name and value) and store in it the list of attributes that are to be used to compose a RADIUS reply.

To provide the required information to our module, we modified the function that computes EMSK for all TLS-related EAP methods, in order to compute also the EAP session ID and store it internally in a value-pair. Then, we modified the generic EAP module to compute EP and MTK from EMSK and EAP Session ID and upload them to our PM. This action takes place immediately before sending an Access-Accept RADIUS reply to the supplicant To interact with the PVH, we added extra code to the parser of EAP types within EAP messages, for calling the appropriate functions of our module.

IV. Security Evaluation

A critical security concern of traffic tagging is privacy preservation from unauthorized eavesdroppers. Since EP is a semantic-free value that changes on each EAP session, personal information gathering from an EP is limited and mainly based on data collected from the execution of other protocols that are usually explored (802.1X, DHCP, etc.). The RoleId and AppId, on the other hand, are transmitted encrypted, thus not yielding any information for the eavesdropper. Even considering the fact that an eavesdropper may aggregate datagrams with the same encrypted values, we do not consider that to be a dangerous information leak: it only reveals that a particular (possibly unknown) application, running under a particular (unknown) role, is responsible for some network activity. Concluding, the tagging system preserves the privacy of tagged traffic as long as the encryption keys TSK are known only by the TM and authorized auditors.

MTK keys are at start locally computed and known by the TM and the CTA of each supplicant, but may be communicated to network auditors on demand. Since the traffic between the TM and a network auditor is not currently encrypted, just authenticated, then an attacker capable of eavesdropping the traffic between an auditor and the TM may collect MTK keys. It is up to the network administrator to ensure that such communication occurs on a protected network.

The authentication used for our tags is based on encrypted, partial MD5 digests, computed over the tags, the destination IP address and the IP payload. MD5 has known collision problems, and furthermore we use only part of its output. Therefore, we believe that a powerful and determined attacker may use an authenticated set of tags from one packet to mark another different packet by exploring digest collisions. On the other hand, we also think that the success rate is very low to make this exploit worthwhile. Furthermore, layer-two segregation strategies, such as basic switching and VLANs, may prevent attackers, namely bots, from collecting tags from other hosts in order to use them for their own fabricated traffic.

The EP is only 6-byte long, which may yield collisions. However, since collisions among EPs do not imply collisions among the related MTKs, a simple cooperation between the PM and network auditors may solve this issue: the PM sends, upon request, all the IDs and MTKs of all the current entities using the same EP. Using the packet authenticator, and the MTK for each ID, the network auditor can easily discriminate the identity of the packet sender.

V. Related Work

To the best of out knowledge, we do not know of any other approach for tagging traffic with our source-related information – identity of originator, role of originator and originating application.

VLAN tagging capabilities introduced by 802.1Q [10] enable network administrators to implement source segregation strategies based on frame tags, managed by access switches. In theory, we could use VLAN tags to add our EP to layer-two frames, but VLAN tags are naturally lost after passing through a gateway or router. And, in any case, switches have no means to provide the other tags we use, RoleId and AppId.

The exploitation of the 802.1X architecture and the EMSK-rooted key hierarchy for adding security to a local network is not a novelty, as it was previously used (cf. [19], [8]). However, it was not used before as the basis for trustworthy packet tagging, but instead for implementing LAN-wide secure interactions and fast 802.1X reauthentications.

VI. Conclusion

We designed and implemented a packet tagging system for enabling detailed traffic profiling based on source origin information. That information includes a pseudonym of an entity (person or server), the role it is playing and the application that produced the traffic. The ultimate goal of such profiling is the detection of abnormal activities on hosts, namely the presence of bots on them.

The system uses the 802.1X architecture for producing and distributing cryptographic credentials to all the stakeholders, namely network clients and network auditors. Traffic tags are voluntarily added by clients, which have all the interest to collaborate with network auditors to protect themselves. The added tags do not compromise the privacy of clients, as them cannot be understood by unauthorized eavesdroppers. Finally, the packing tagging mechanism with an IP option respects the IP standard, therefore tagged packets should not be dropped by network scrubbers or target hosts.

For immediate future work we plan do improve the CTA component, namely to improve its efficiency and to add a more rich identification policy for source applications. We also plan to test the system on field experiments, preferably using known bots, to observe the reaction of users in the usage of roles and to evaluate the usefulness of detailed profile building.

References

[1] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz, “Extensible Authentication Protocol (EAP),” RFC 3748 (Proposed Standard), Jun. 2004.

[2] B. Aboba and P. Calhoun, “RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP),” RFC 3579 (Informational), Sep. 2003.

[3] B. Burns, J. Granick, S. Manzuik, P. Guersch, D. Killion, N. Beauchesne, E. Moret, J. Sobrier, M. Lynn, E. Markham, C. Iezzoni, and P. Biondi, *Security Power Tools*. O’Reilly Media, Inc., Aug. 2007.

[4] P. Congdon, B. Aboba, A. Smith, G. Zorn, and J. Roese, “IEEE 802.1X Remote Authentication Dial In User Service (RADIUS) Usage Guidelines,” RFC 3580 (Informational), Sep. 2003.

[5] T. Dierks and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2,” RFC 5246 (Proposed Standard), Aug. 2008.

[6] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina, “Generic Routing Encapsulation (GRE),” RFC 2784 (Proposed Standard), Mar. 2000.

[7] Y. Jin, N. Duffield, P. Haffner, S. Sen, and Z.-L. Zhang, “Inferring applications at the network layer using collective traffic statistics,” *SIGMETRICS Perform. Eval. Rev.*, vol. 38, pp. 351–352, June 2010.

[8] R. Marques, E. Araújo, and A. Zúquete, “Fast 802.11 handovers with 802.1x reauthentications,” *Security and Communication Networks*, vol. 4, no. 3, pp. 267–283, 2011.

[9] N. I. of Standards and T. (NIST), “ADVANCED ENCRYPTION STANDARD (AES),” FIPS PUB 197, Nov. 2006.

[10] L. S. C. of the IEEE Computer Society, “IEEE Standard for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks,” IEEE Std 802.1Q-2005, May 2006.

[11] ——, “IEEE Standard for Local and Metropolitan Area Networks: Port-Based Network Access Control,” IEEE Std 802.1X-2010, Feb. 2010.

[12] R. Puri, “Bots & Botnet: An Overview,” SANS Institute InfoSec Reading Room, Aug. 2003.

[13] C. Rigney, W. Willats, and P. Calhoun, “RADIUS Extensions,” RFC 2869 (Informational), Jun. 2000.

[14] C. Rigney, S. Willens, A. Rubens, and W. Simpson, “Remote Authentication Dial In User Service (RADIUS),” RFC 2865 (Draft Standard), Jun. 2000.

[15] R. Rivest, “The MD5 Message-Digest Algorithm,” RFC 1321 (Informational), Apr. 1992.

[16] J. Salowey, L. Dondeti, V. Narayanan, and M. Nakhjiri, “Specification for the Derivation of Root Keys from an Extended Master Session Key (EMSK),” RFC 5295 (Proposed Standard), Aug. 2008.

[17] C. Schiller, J. Binkley, G. Evron, C. Willems, T. Bradley, D. Harley, and M. Cross, *Botnets: The Killer Web App*. Syngress, Feb. 2007, iSBN-10: 1597491357, ISBN-13: 978-1597491358.

[18] D. Simon, B. Aboba, and R. Hurst, “The EAP-TLS Authentication Protocol,” RFC 5216 (Proposed Standard), Mar. 2008.

[19] A. Zúquete, “Protection of LAN-wide, P2P interactions: a holistic approach,” *Int. J. Commun. Netw. Distrib. Syst.*, vol. 3, pp. 408–426, Aug. 2009.