Detecting Protected Layer-3 Rogue APs

Hongda Yin, Guanling Chen, and Jie Wang
Department of Computer Science, University of Massachusetts Lowell
{hyin, glchen, wang}@cs.uml.edu

***Abstract*— Unauthorized rogue access points (APs), such as those brought into a corporate campus by employees, pose a security threat as they may be poorly managed or insufficiently secured. Any attacker in the vicinity can easily get onto the internal network through a rogue AP, bypassing all perimeter security measures. Existing detection solutions work well for detecting layer-2 rogue APs. It is a challenge, however, to accurately detect a layer-3 rogue AP that is protected by WEP or other security measures. In this paper, we describe a new rogue AP detection method to address this problem. Our solution uses a verifier on the internal wired network to send test traffic towards wireless edge, and uses wireless sniffers to identify rouge APs that relay the test packets. To quickly sweep all possible rogue APs, the verifier uses a greedy algorithm to schedule the channels for the sniffers to listen to. To work with the encrypted AP traffic, the sniffers use a probabilistic algorithm that only relies on observed packet size. Using extensive experiments, we show that the proposed approach can robustly detect rogue APs with moderate network overhead.**

I. Introduction

A *rogue AP* is an unauthorized access point plugged into a corporate network, posing a serious security threat to enterprise IT systems. Rogue APs are typically installed by employees in work places for convenience and flexibility. Although users could leverage common security measures such as Wired Equivalent Privacy (WEP) to protect their network communications, such measures may not be consistent with the corporate security policies and they are often inefficient. For example, researchers have identified design flaws in WEP, which can be easily exploited to recover secret keys [1]. Rogue AP exposes internal networks to the outside world, making it easy for people to bypass security measures.

Several vendors (e.g., Aruba Networks, AirMagnet, and AirDefense) sell WLAN Intrusion Detection System (WIDS) products that can detect rogue APs and other security threats. These WIDS solutions typically consist of a set of wireless sniffers that scan airwaves for packet analysis. These sniffers can be *overlaid* with APs, meaning that they are strategically deployed as a separate infrastructure. These sniffers can also be *integrated* with APs, meaning that the APs themselves, in addition to serving wireless clients, also perform IDS functionalities periodically. Researchers have recently proposed to turn existing desktop computers into wireless sniffers to further reduce deployment cost while still providing a reasonable coverage [2].

Detecting rogue APs using wireless sniffers requires that the sniffers listen to all WLAN channels to detect the presence of APs either sequentially or non-sequentially using various channel-surfing strategies [3]. If a detected AP is not on the authorized list, it is flagged as a suspect. The detected suspect, however, may well be a legitimate AP belong to a neighboring coffee shop or a nearby household. The question then becomes how to *automatically* verify whether the suspect AP is actually on the enterprise wired network or not.

One effective approach to verify layer-2 rogue APs is to poll network switches over SNMP to determine MAC addresses associated with each port on the switch. If a wireless sniffer observes any of these MAC addresses in the air, the associated AP must be on the wired network, given that the AP works as a layer-2 bridge. For layer-3 rogue APs, one common verification approach is to have a nearby sniffer actively associate with the suspect AP and ping a known host on the internal network not accessible from outside. If successful, the suspect AP is confirmed to be on the internal network and is indeed a rogue. This approach, however, fails to detect protected APs that require valid MAC addresses or other authentication methods (such as WEP, WPA, and WPA2) for successful association. A recent study confirms that existing solutions are indeed not adequate for detecting protected APs that act as routers [4]. Unfortunately, most off-the-shelf APs are layer-3 devices and it is not unreasonable to assume that the employee will use some kind of protection on those APs, given the increased publicity on wireless insecurity, making them hard to be detected with existing approaches.

In this paper, we propose a new solution to detect protected layer-3 rogue APs. In particular, instead of sending test packets from wireless side, our solution has a *verifier* on the wired network that sends test packets towards the wireless side. Should any wireless sniffer pick up these special packets, we have effectively verified that the suspect AP that relays these packets is indeed on the internal network and thus is a rogue AP.

Our approach needs to address two issues for robust detection. First, we note that a layer-3 rogue AP normally comes with a NAT(network address translation)

module so that multiple devices can share the same connection. It is impossible to send test packets directly to the associated wireless clients, from the wired side, because they have private IP addresses. We note that NAT rewrites outbound packets from associated clients with its own address. Thus, we can use the verifier to monitor the wired traffic and send test packets to the active sources. If an active source is an AP, the test packets will be forwarded by NAT and observed by wireless sniffers. Second, the sniffers may not be able to recognize test packets by examining the payload if the AP has enabled encryption. To solve this problem we devise a probabilistic verification algorithm based on a sequence of packets of specific sizes.

The contribution of this paper is a novel approach to detect protected rogue APs acting as layer-3 routers, which are common cases. We develop an algorithm for the verifier and wireless sniffers to cooperatively verify rogue APs. Using simulations and experiments, we show that the proposed approach can effectively detect rogue APs in a relatively short time period with moderate network overhead. Once a rogue AP is confirmed, the verifier returns its IP address from which the switch port to that address can be found and automatically blocked until the rogue is removed.

In this paper we assume the owner of rogue APs is not malicious: he simply sets up an unauthorized AP for convenience. The verifier and the wireless sniffers themselves are guarded by typical security measures, such as access control and intrusion detection, against external attackers. The details of these methods are not the focus of this paper.

The rest of the paper is organized as follows. We present a network model in Section II. In Sections III and IV we describe our monitoring and verification algorithms, respectively. We present evaluation results in Section V. We discuss potential limitations and scalability improvements of our method in Section VI. Section VII summarizes related work and we conclude in Section VIII.

II. Network Model

We assume that wireless sniffers are deployed to monitor the enterprise airspace. The sniffers employ some channel-hopping strategies to detect the presence of APs. These APs may be using different communication channels. The sniffers are connected to the wired network and can communicate with an internal *verifier*. The sniffers update the verifier about the detected APs and their channels. The verifier may instruct certain sniffers to switch to a particular channel during the verification process. Figure 1 shows a simplified network where sniffer S_1 covers multiple APs, and sniffers S_1 and S_2 have overlapping coverage. C_1 and C_2 are wireless workstations.

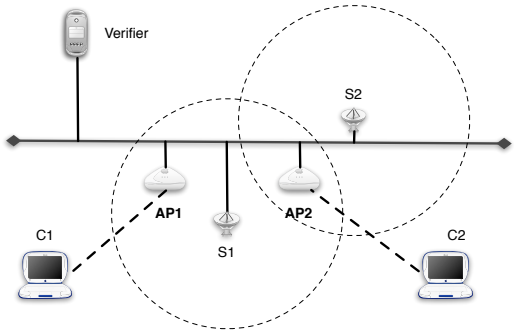

Fig. 1. A simplified network model with APs and sniffers.

A rogue AP could be a layer-2 device or a layer-3 device. While our approach works for both cases, we will focus on layer-3 devices, which are most common for consumer APs. A layer-3 AP typically comes with NAT and each associated wireless clients is assigned with a private IP address. When a client communicates with a wired host with address A_{out} and port number P_{out}, NAT opens a special port on itself (P_{nat}) and rewrites the headers of outbound packets so they look like as if they are coming from its own address A_{nat} and port P_{nat}. At this time, any packet from A_{out} and P_{out} that are sent to A_{nat} and P_{nat} will be forwarded by NAT to the wireless client, thus appearing on the wireless medium and observable by sniffers in range.

NAT ports are opened dynamically when clients initiate communications with outside destinations. Thus, the verifier needs to monitor outbound traffic and send test packets to the sources A_{src} and P_{src}. The test packets, however, need to have forged headers to look like as if they are coming from A_{out} and P_{out}. It is fairly easy to use existing tools to send packets with customized headers. If some of the sniffers report reception of these packets, A_{src} must be the IP address of a rogue AP. Based on the network topology, switches' ARP tables, and possibly DHCP logs, it is feasible to track down exactly which switch port the rogue AP is plugged into.

Note that the verifier essentially injects *spoofed* packets into a normal communication path. We need to be careful what types of packets to inject without disrupting normal communication. TCP packets have a sequence number (SN) header field to ensure reliable data transfer. The receiver maintains a window rcv_wnd and only accepts packets whose SN falling into that window. Thus, the verifier should inject test packets with forged SN set to be outside that window, such as an SN the receiver has recently acknowledged. The receiver will then silently drop these test packets. On the other hand, UDP packets do not have transport-layer SN and all

packets will be forwarded to the application layer. To avoid confusing applications using UDP, we refrain from injecting UDP test packets and the verifier only monitors TCP traffic.

III. Wired Traffic Monitoring

The verifier monitors wired traffic. Every observed host on the internal network is potentially a wireless AP, unless explicitly marked by administrators as wired, such as the well-known addresses of network servers. All addresses allocated to user workstations, however, should be considered susceptible, since an user could configure an AP to use the static IP address assigned to her workstation when DHCP is not available.

Verifying a host takes a certain amount of time for sniffers to switch channels and analyze observed packets (Section IV). If there is a burst of new sources observed in a short period, these sources are queued by the verifier and tested sequentially. The verifier always picks the host with the oldest timestamp for verification. All hosts in the queue will be updated with new timestamp if more traffic from them is observed. The NAT port, however, could expire for being idle for too long when the verifier tests a source that has been waiting in the queue for a while. The verifier will skip testing any source that has not been updated for certain amount of time, say 5 minutes, to avoid sending traffic to an invalid port number. These sources will be tested next time when their traffic is observed.

Further reducing network load is possible if the verifier can tell, based on traffic patterns, whether the observed hosts are likely connected to a wireless link. Consider IEEE 802.11b, for example, where the time interval between two back-to-back packets from a wireless client has an average value of $810\mu s$ under ideal situation without contention [5]. Note that a wireless client has to wait for a random backoff after successfully sending out a packet. On the other hand, intervals between consecutive packets going through Ethernet connections have a much smaller value, typically less than $50\mu s$ due to a rather different MAC protocol [6]. Thus, it is possible to statistically distinguish wireless and wired sources by analyzing the timing between packets. Wei et. al propose an offline algorithm to use inter-ACK intervals over TCP flows for iterative flow classification [6].

We want to find an online algorithm for the verifier to quickly classify observed hosts, so that the verifier can focus only on testing those classified as wireless sources. If the classification is 100% accurate, then the verifier do not need to take further actions. But we do not know whether an algorithm can achieve this accuracy. Inspired by Wei et al's work [6], we choose to count the *short* packet intervals, i.e., less than $250\mu s$, between TCP packets of both inbound and outbound directions. The verifier classifies any source, whose ratio of inbound and outbound short intervals exceeds a threshold, as a potential wireless host. The reason of doing so is that the number of outbound short intervals, observed after packets having gone through a wireless link, is expected to be much smaller than the inbound number of short intervals, observed before the packets reaching the AP.

In summary, the verifier may test every observed internal host or only test those hosts classified as wireless sources. The first approach is simple to implement. The classification can reduce test traffic, but it may also lead to inaccurate results and longer detection delay. In both cases, a verified non-rogue address will not be tested for some time, after which it becomes susceptible again and is subject for verification. We evaluate these two methods in Section V.

IV. Rogue AP Verification

The verifier sends test packets to observed sources and see whether some wireless sniffer can hear these packets. But a rogue AP may encrypt traffic and so sniffers cannot rely on special signature embedded in the application-layer data. One may borrow ideas from *covert channels*, in which the verifier deliberately manipulate the timing between packets without injecting any new packets. The packet intervals thus carry unique information, sometimes called *watermark*, which can be identified by a passive sniffer [7]. While not intrusive, this approach seems less appealing in a wireless environment because 802.11 MAC contention may cause timing-based analysis less robust. Also, it requires customizing routers on the data path, which is a non-trivial task and will degrade routing performance.

A. Packet size selection

We choose to send test packets with sizes not frequently seen on the suspect APs. Namely, the sniffers report empirical distribution of packet sizes observed from APs to the verifier, who then selects an unusual size for test packets. This simple approach has an implicit assumption that sizes of the downstream packets from APs are not uniformly distributed. We analyzed a network trace collected from a WLAN made available to attendees of a four-day academic conference (Sigcomm 2004) [8]. Figure 2 shows the PDF of one AP's downstream data packet sizes (the injected test packets will appear as 802.11 data packets on wireless). It is clear that most packet sizes appear infrequently. Analysis of other APs confirms this observation.

In general, we want to choose a relatively small packet size so that the verifier demands less bandwidth. Also, a small packet will unlikely be fragmented by APs and thus will not be missed by sniffers. Once a size is chosen, the verifier notifies the sniffers to only watch for packets

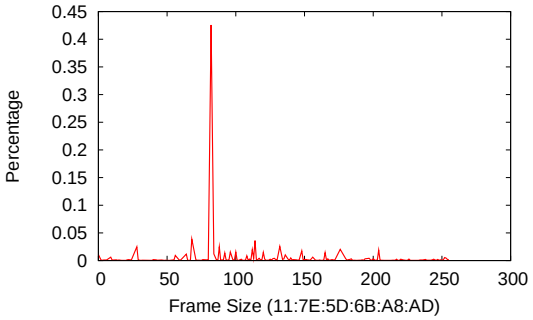

Fig. 2. Size distribution of downstream data packets.

with this particular size. To get a frame size N, the verifier should send out TCP packets with application payload of N minus the size of all the protocol headers captured by the sniffer. Note that the sniffer should also consider the overhead of encrypted frames. For example, there are additional 12 bytes overhead for WEP frames (3 bytes IV, 1 byte key number, and two 4-bytes ICVs).

Suppose that the two geographically close APs in Figure 1 belong to two companies, where S_1 may overhear the test packets from the other AP if the verification processes occur simultaneously in these two networks. In this case, S_1 may falsely conclude that another company's AP is a rogue. To mitigate this problem, the verifier chooses randomly a packet size from the M least observed packet sizes on the network, minimizing the chance of colliding verification time and test packet size.

B. Binary hypothesis testing

To avoid false positives caused by normal packets that happen to have the same size of the test packets, the verifier sends more than one test packet to improve the robustness of detection. The question is how many test packets the verifier should send. Note that we may not observe a back-to-back packet train of the same size, because normal packets of different sizes may be inserted as the test packets going through the shared wireless medium. The APs may also send other frames, such as beacons, in the middle of a test packet sequence.

We use Sequential Hypothesis Testing theory to determine the number of test packets that can achieve desired detection accuracy [9]. Assume that the probability to see data packets with the chosen test size s from a monitored AP is p, and the sniffers determine independently whether the AP is relaying test packets based on observed downstream traffic. Intuitively, the more packets with the test size are observed, the more likely a verification is in process and the AP is a rogue. For a given AP being monitored by some sniffer, let X_i be a random variable that represents the size of i^{th} downstream data packet, where

$$X_i = \begin{cases} 0, & \text{if the packet size} \neq s \\ 1, & \text{if the packet size} = s \end{cases}$$

We consider two hypotheses, H_0 and H_1, where H_0 states that the monitored AP is not relaying test packets (thus not a rogue), and H_1 states that the AP is relaying test packets (thus a rogue). Assume that the random variables $X_i|H_j$ are independent and uniformly distributed, conditional on the hypothesis H_j. We express the distribution of X_i as below:

$$Pr[X_i = 0|H_0] = \theta_0, Pr[X_i = 1|H_0] = 1 - \theta_0$$
$$Pr[X_i = 0|H_1] = \theta_1, Pr[X_i = 1|H_1] = 1 - \theta_1$$

We can specify the detection performance using the detection probability, P_D, and the false positive probability, P_F. In particular, we can choose desired values for α and β so that

$$P_F \leq \alpha \quad \text{and} \quad P_D \geq \beta \tag{1}$$

where typical values might be $\alpha = 0.01$ and $\beta = 0.99$.

The goal of the verification algorithm is to determine which hypothesis is true while satisfying the performance condition (1). Following [9], as each packet is observed we calculate the likelihood ratio as follows:

$$\Lambda(X) = \frac{Pr[X|H_1]}{Pr[X|H_0]} = \prod_{i=1}^{n} \frac{Pr[X_i|H_1]}{Pr[X_i|H_0]} \tag{2}$$

where X is the vector of events (packet size is s or not) observed so far and $Pr[X|H_i]$ represents the conditional probability mass function of the event stream X given that model H_j is true. The likelihood ratio is then compared to an *upper* threshold, η_1, and a *lower* threshold, η_0. If $\Lambda(X) \leq \eta_0$ then we accept hypothesis H_0. If $\Lambda(X) \geq \eta_1$ then we accept hypothesis H_1. If $\eta_0 < \Lambda(X) < \eta_1$ then we wait for the next observation and update $\Lambda(X)$. These two thresholds can be upper and lower bounded by simple expressions of P_F and P_D ($\eta_1 = \frac{\beta}{\alpha}$ and $\eta_0 = \frac{1-\beta}{1-\alpha}$), from which we can compute the expected number of observations needed before the verification algorithm accepts one hypothesis [10]:

$$E[N|H_0] = \frac{\alpha ln\frac{\beta}{\alpha} + (1-\alpha) ln\frac{1-\beta}{1-\alpha}}{\theta_0 ln\frac{\theta_1}{\theta_0} + (1-\theta_0) ln\frac{1-\theta_1}{1-\theta_0}}$$
$$E[N|H_1] = \frac{\beta ln\frac{\beta}{\alpha} + (1-\beta) ln\frac{1-\beta}{1-\alpha}}{\theta_1 ln\frac{\theta_1}{\theta_0} + (1-\theta_1) ln\frac{1-\theta_1}{1-\theta_0}} \tag{3}$$

Given a suspect AP, θ_0 can be empirically calculated by sniffers. The verifier injects a sequence of test packets of the same size, producing a new probability θ_1. Given desired performance conditions α and β (1), we can

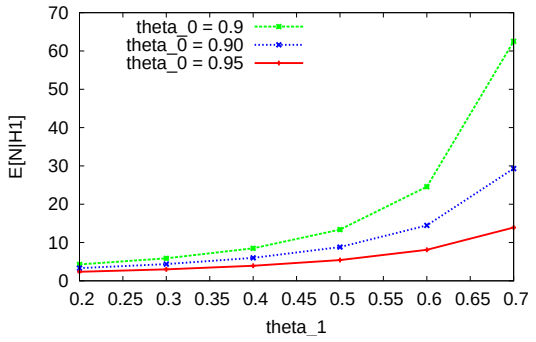

Fig. 3. The expected number of observations to accept H_1.

Input: ip address to be verified

1) compute J (the set of sniffers j) and I (the set of targets i detected by any j)
2) $\forall$ target i in I, compute S_i
3) select the sniffer j in S_i, whose r_j is smallest (r_j is the current round scheduled for j)
4) if there are multiple sniffers having the same minimum rounds, select the sniffer j whose t_j is smallest
5) once sniffer j is selected to cover i, we schedule j to tune into channel c_i at round r_j
6) remove all targets on channel c_i covered by j from I and place them under sniffer i in row r_j++
7) repeat from step 2) until I is empty

Fig. 4. Sniffer channel scheduling algorithm.

establish how θ_1 relates to the number of observations needed (3) for the verifier to accept H_0 or H_1 (deciding whether the AP is a rogue or not). Figure 3 shows that we want to choose packet size with small probability appearing on normal communications ($1-\theta_0$) to reduce the number of observations. It also shows a tradeoff for θ_1, for the verifier needs to inject more packets for smaller θ_1 and thus yielding quicker algorithm termination.

The number of test packets relates to the current load of the monitored AP. If the rate of downstream data packets is measured as R, the verifier needs to inject $R/2$ test packets to bring θ_1 to about 0.7. Fortunately, R is often small, particularly for rogue APs, so the verifier does not have to send a huge amount of test packets. We computed the number (and the total bytes) of downstream data packets of an AP deployed in our department over an 8.5-hour period in the afternoon. The load on that AP was fairly light and most of the time the R is less than 10 packets per second and the bandwidth usage is less than 8 Kbps.

C. Sniffer channel scheduling

When the verifier starts to test a source, it does not know which suspect AP will relay the test packets that will be seen by the sniffers. The sniffers, however, may have detected multiple suspected APs on different channels. For example, if there is only one sniffer S_1 in Figure 1 and the two APs run on different channels, S_1 has to cover both APs but can only tune to different channels sequentially. So for every source verification, the verifier needs to repeat the test packets when scheduling sniffers over different channels to cover all suspect APs, from any of which the test packets may be relayed.

Assume that there are N suspect APs and M sniffers. We label a suspect AP by i and a sniffer by j, where $1 \leq i \leq N$ and $1 \leq j \leq M$. We call a suspect AP a *target*. Each target i transmits on a channel c_i, which could be any value between 1 and C_{max}. Here C_{max} is the maximum number of channels available. For example, $C_{max} = 13$ for 802.11b. A sniffer can hear multiple targets if they are in its range, even if they may not be on the same channel. But a sniffer can only listen to one channel at a time and must switch channels to monitor other targets in range. A target may also be covered by multiple sniffers if it falls in their ranges.

We use S_i to denote the set of sniffers that can cover target i and s_i the size of S_i. We use T_j to denote the set of *channels* of the targets that can be heard by sniffer j and t_j the size of T_j. For each source to be verified, the verifier can obtain these two sets based on previous sniffer reports. Each verification contains multiple test rounds, r_k. At each round the verifier schedules sniffers to certain channels and sends test packets to the source being tested. The goal of the verifier is to minimize the number of sniffers and the number of test rounds such that all targets are covered by at least one sniffer. We refer this problem as Minimum Channel Cover (MCC) problem. We show that MCC is NP-hard. In particular, we reduce the NP-complete Minimum (Set) Cover problem [11] to the decision problem of MCC (see Appendix).

We use a greedy algorithm to approximate the optimal solution of MCC. Consider a matrix where the columns are sniffers from 1 to M, and the rows are test rounds from r_1 to r_k. Our algorithm is to place all targets into the matrix cells, subject to the constraints that all targets in one cell must be on the same channel and they are all in the range of the sniffer of that column.

For a given target i, we want to use a sniffer j in S_i to cover i, such that t_j is small that may reduce other sniffers' work. This heuristic can be illustrated in Figure 1, where we want to use sniffer S_2 to cover AP_2 and S_1 to cover AP_1, so only one test round is needed. If multiple sniffers can cover a target being considered, we want to allocate the target for the sniffer with smallest test rounds scheduled for the purpose of reducing the overall number of test rounds. Combining these heuristics, we get the algorithm shown in Figure 4.

V. Evaluation Results

In this section we present experimental results of our detection method. We first evaluate wired traffic monitoring methods using extensive network traces. We then evaluate the sniffer channel scheduling algorithm using simulation. Finally, we present empirical implementation of the proposed rogue AP detector.

A. Wired traffic monitoring

The verifier monitors wired traffic and test internal addresses. The premise of this approach is that the verification load could be amortized over time. To evaluate this approach, we need extensive long-term network traces. Unfortunately, we do not have such kind of data from real enterprise networks. Instead, we use traces collected from Dartmouth campus WLAN as a baseline evaluation. These traces only contain traffic from and to wireless hosts. We took two 10-day data sets collected in November 2003, one collected from APs in a library building and the other collected from APs in a residential hall. There are more than 1200 unique IP addresses in each trace. We scan each trace chronically, where in 75% of the cases the time needed to observe a previously unseen address is greater than 100 seconds.

We simulated the verifier to test every IP address in these two traces. We used 30 seconds as a fairly conservative value for the time needed to verify a source (in practice, verification only needs a couple of seconds when the number of targets is small). Thus, a total of 120 hosts can be verified in an hour. When the verifier was testing an address, all newly arrived hosts were queued. Any newly arrived packet from a queued host would pull that host to the end of the queue and the verifier always picked the head of the queue when it started next verification. If the host to be verified had not generated any packets for 5 minutes, the verifier would simply ignore it to avoid sending test packets to an expired NAT port number.

Figure 5 shows how the length of verification queue changed over 10 days. The length of both queues had never exceeded 20. The queues, however, did have some hosts remaining untested, roughly 6 for the library trace and 10 for the residential trace. This means that there were some hosts that appeared early in the trace, expired in the queue before they could be verified, and were never seen again (so no more traffic triggering verification). Most likely this is an artifact caused by device mobility.

Figure 6 shows the distribution of the verification delay since the first time a request was observed from a host. In more than 98% of the cases a host could be verified within 5 minutes since its first appearance. A few hosts were not verified after a couple of days, most likely when these mobile hosts visited the monitored APs again. We also found that in more than 95% and 99% of the cases a host could be verified within 50 seconds since its last update for the library and residential traces, respectively. These results suggest that the verifier worked effectively for a moderate-size network.

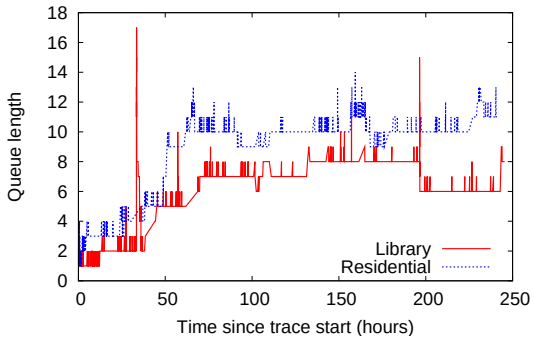

Fig. 5. Length of verification queue (campus WLAN).

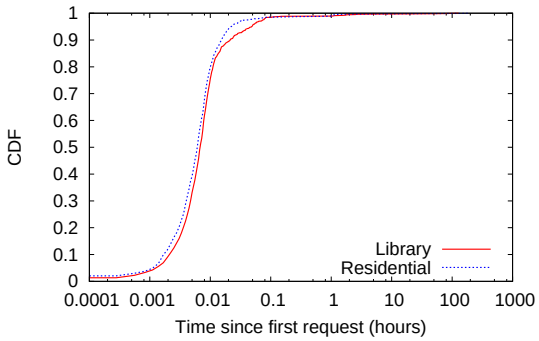

Fig. 6. Verification delay since first request (campus WLAN).

While the Dartmouth trace is a long-term trace, it only consists of wireless hosts and does not represent a typical enterprise where many hosts are wired. So we took a one-day enterprise network trace presented in [12], and merged the data sets collected from oddly and evenly numbered router ports to produce two time-continuous traces. We then considered each trace emulating a verifier that monitors a port for one hour, moves on to the next port for another hour, and so on. We found that in about 46% or 66% of the cases the time intervals needed to observe a new host are less than 10 seconds. We expect that the intervals would be larger if we had longer-time traces from the same router ports.

Figure 7 shows the change of the verification queue length. As expected, the queue length jumped and then gradually decreased as the verifier moved to a new port. This pattern is quite visible for the even-port trace. At the 6th hour of the odd-port trace, the monitor started on an active subnet and 539 hosts were observed during that hour. Many of these hosts could not be verified in time during this period and they stayed in the queue.

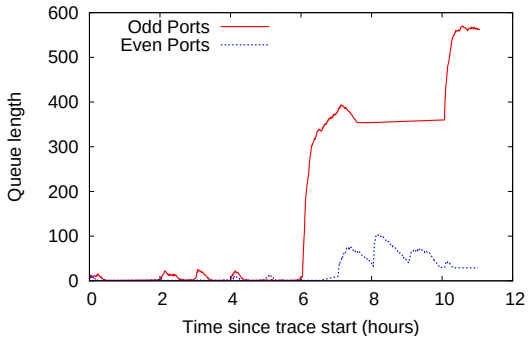

Fig. 7. Length of verification queue (enterprise LAN).

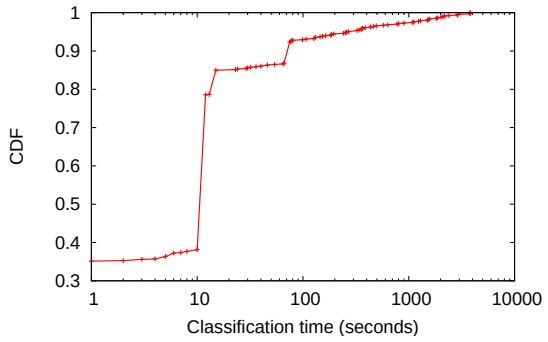

Fig. 9. Classification time (HTTP proxy).

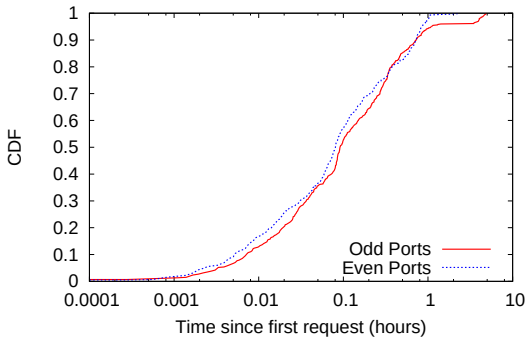

Fig. 8. Verification delay since first request (enterprise LAN).

We observed a similar pattern at the 10th hour for another busy subnet. We note that once the monitor cycles through these ports, the queued hosts will be verified once the monitor visits previous ports again.

Figure 8 shows the distribution of the verification delay since the first time a request was observed from a host. For those hosts that were verified, 77% of them were verified within 20 minutes since their first appearance for both traces. Also, 49% of them were verified within 100 seconds since their last update. For all these tests, the verifier could achieve even faster speed if it could test a source quicker than 30 seconds.

Note that the verifier could reduce its workload by only testing likely wireless sources (Section III). To evaluate this approach, we set up a Linksys AP (model WRT 54G), configured to use 802.11b, which was plugged into our department network mimicking a rogue AP. One of the author's laptop used that AP as the main network connection whenever the author was in his office. The Web browser on that laptop was configured to use a Web proxy running tcpdump to collect the HTTP traffic. Thus, all the Web transactions from that laptop went through the AP and the Web proxy, recorded in the tcpdump trace. Then we ran the classification algorithm through this trace to see whether we could classify the IP address of the AP as wireless. While we only had one trace, we started classification at different time in the trace. We got total 667 classification over a 14-day trace with 30-minute separation. Our classifier achieved 100% accuracy and Figure 9 shows the distribution of how long the classifier took to make a decision since it saw the first request. In about 93% of the cases the classifier could conclude in less than 100 seconds.

Another question is whether the classifier can correctly identify the wired sources to avoid testing them further. We ran the classification over the previous one-day enterprise network traces, where 568 unique IP addresses were classified as wireless and 227 were classified as wired. The median time to classify a wireless and a wired source is 86 and 476 seconds, respectively. While we do not have the ground truth to tell the accuracy of the classification over this trace, it is likely, assuming most of the enterprise hosts are wired, that the algorithm had correctly classified about 30% of wired hosts and thus could reduce testing traffic significantly. We are currently tuning the algorithm to further reduce false positives.

B. *Sniffer channel scheduling*

Here we evaluate the sniffer channel scheduling algorithm. Ideally we want to use fewest sniffers to cover all the suspect APs. A sniffer can only tune into different channels sequentially, so we also want to have minimum number of channel tuning to speed up the verification. We used simulation for evaluation so we could systematically change parameters. On a 500x500 site, we randomly placed 50 APs, each using a random channel from the maximum value to be $MaxChan$. We set the sniffering range to be either 100 or 200, and changed the number of randomly placed sniffers.

Figure 10 shows the maximum number of tuning time of all sniffers, namely, the maximum number of channels some sniffers need to switch. We divided the number of sniffers that have detected APs by the total number of APs detected by sniffers to get the sniffer/AP ratio. As this ratio increases, the tuning time decreases. This is as expected because the work of covering the APs can be

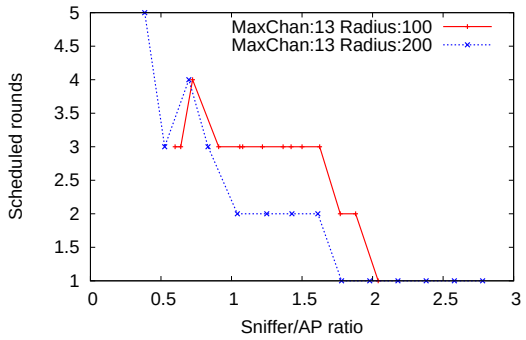

Fig. 10. Number of scheduled rounds for sniffers.

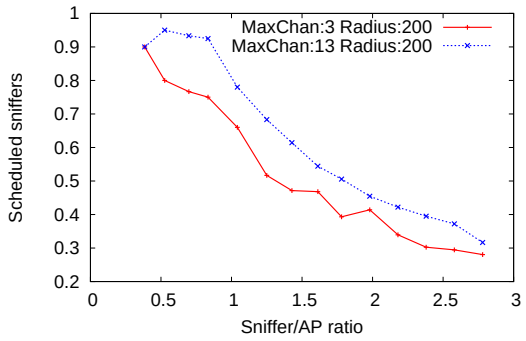

Fig. 11. Number of sniffers to be instrumented.

shared by more sniffers. With a larger sniffering radius, a sniffer can cover more APs with the same channels so the tuning time also decreases.

Figure 11 shows the ratio of used sniffers, since not all sniffers are needed due to overlapping coverage. We can see that the number of required sniffers decreases as the sniffer/AP density increases. On the other hand, if the APs run over a larger channel range, more sniffers are needed to cover them. In practice, however, people tend to use the default channel or choose from orthogonal channels (only 3 for 802.11b).

C. Empirical implementation

We have implemented the proposed rogue AP detector in C++ using the Click framework [13], which is designed for packet routing but is also suitable for network measurement and analysis. A Click module on the verifier monitors all the traffic passing through its Ethernet NIC, and communicates through TCP connections with other Click modules on the sniffers that monitor wireless traffic. The sniffers run on IBM Thinkpad T42 installed with Ubuntu Linux 5.1 and Madwifi driver. The sniffers simply call `iwconfig` to switch channels. We installed the verifier on a Web proxy and we instrumented both a wireless and wired client browsing through that proxy.

For every newly observed IP address, the verifier notifies the sniffers which channel to listen and packet size to observe. Then it uses an open source tool, Nemesis, to craft headers of the test packets so they look like from TCP peers of the sources being verified. After sending the test packets, the verifier sleeps for 2 seconds before instructing sniffers to report back the number of observed test packets, so it can decide whether the address being tested belong to a rogue AP. Note that the verifier sleeps for a fixed time interval to make sure all the test packets have been delivered by the potential AP before making a conclusion. In this lab environment, the verifier could reliably detect the AP of the instrumented wireless client with 100% accuracy (no false negatives) and had never classified the address of the instrumented wired client as an AP (no false positives).

VI. Discussion

Our research is conducted within the context of project MAP [14], in collaboration with Dartmouth College and Aruba Networks. MAP aims to build a scalable measurement infrastructure using wireless sniffers, based on which various online analysis algorithms are designed to detect WLAN security threats. Our detector will be integrated with MAP as an independent detector. Within MAP, we envision a building-wide system deployment with 20–50 wireless sniffers and 1 rogue AP verifier. Assume that there are 1,000 active hosts in the building to be verified using 30 100-byte test packets for each host, the imposed network load is about 3MB over a relatively long period. We do not consider this load as a bottleneck, given that many organizations already run a scanner to periodically check various properties of the internal hosts as part of their security operations.

The verifier, on the other hand, may have to handle a large amount of traffic generated inside the building. While the processing is fairly lightweight (only checking transport-layer header), there are ways to further reduce the overhead. For example, many routers can be configured to export NetFlow (or sFlow) records marking the endpoints of the active flows. The verifier can greatly increase its scalability by using this information instead of parsing the whole packet streams. This approach will also reduce some privacy concerns since only IP addresses and port numbers are exposed to the verifier. The verifier, in this case, need to randomly select a TCP sequence number since NetFlow does not provide this information. The probability of SN collision, however, is relatively low given the large SN range. We plan to investigate this tradeoff as a future work.

Our approach has a potential limitation on detecting the rogue APs configured as VPN endpoints, which will drop the verifier's forged test packets since they do not have valid authentication headers, thus the wireless sniffers cannot see the test packets. Currently our solution is to have the verifier to mark the IP addresses that only

have one communication peer (VPN tunneling effect) and alert administrator for further checkup.

Our solution can be easily combined with other rogue AP detection methods. For example, a wireless sniffer may first try to associate with an open AP and ping the internal verifier. Our method can only be activated when the association fails or the AP requires authentication, to further reduce the number of needed verification steps.

VII. Related Work

Almost every WLAN security vendor provides some form of rogue AP detection. Such detection could be as simple as detecting unknown APs compared against an authorized list. More advanced detection uses the associate-and-ping approach described in Section I. The first approach may lead to numerous false positives, classifying neighboring APs as rogues on the internal network. The later approach fails for protected APs that requires authentication for association [4].

RogueScanner by Network Chemistry takes a *collaborative* approach, in which the detector collects various information from network devices and send it back to a centralized server for classification. This raises both privacy and security concerns on sending internal network information to a third party. This approach has to build a huge database on device profiles and needs user feedback on any device that is not in the database. Thus it needs to trust user-input data not to poison their database in a malicious or unintentional way.

DIAR proposes three more types of tests besides active association, leading to perhaps a most comprehensive solution for rogue AP detection [2]. First, one can use MAC address test that requires compiling a list of known MAC addresses on the corporate networks, but it only works for link-layer APs. Second, one can run DHCP test to identify device OS types using signatures from DHCP requests, which only works for APs configured to use DHCP. Finally, the wireless sniffers can replay some captured packets and see whether any wired monitor can detect these packets. DIAR uses several heuristics to reduce false alarms. This approach can bypass encryption problem but require running a wired monitor in each subnet. On the other hand, we target to run only one verifier for monitoring, for example, the NetFlow data from all routers in a building-wide deployment.

Beyah et al. propose to detect rogue APs from the wired side since wired and wireless hosts exhibit different inter-packet temporal distributions [15]. Similarly, Wei et al. use inter-packet timing to classify whether a source is wireless and wired [6], which can be used to detect rogue APs. These statistical methods often lead to inaccurate results due to natural variations of traffic patterns. Our method takes a step further, to employ a verification procedure to determine whether a source is actually wireless or not by leveraging wireless sniffers.

If we know the precise location of a detected AP, we can determine that it is a rogue if it is located inside the enterprise. Existing WLAN localization algorithms can achieve 3–5 meter accuracy [16], but they typically require extensive manual training to build RF maps and thus is not realistic to deploy for any large campus.

VIII. Conclusion

We propose a new method where a wired verifier coordinates with wireless sniffers to reliably detect protected layer-3 rogue APs. The verifier sends test traffic from internal network to the wireless edge and reports the address of confirmed rogue AP for automatic blocking. With trace-based simulations, we show that the verifier's workload may be amortized over time when monitoring a large number of active hosts. Using sequential hypothesis testing theory, the verifier sends a sequence of test packets with specific size so the sniffers can verify the rogue AP that may have encrypted its traffic. In practice, our greedy sniffer channel scheduling algorithm can quickly allocate sniffers to cover all suspect APs. We have implemented the rogue AP detector and will deploy it to a large campus WLAN.

Acknowledgments

This work is supported in part by NSF under Award CCF-0429906 and by the Science and Technology Directorate of the U.S. Department of Homeland Security under Award NBCH2050002. Points of view in this document are those of the authors and do not necessarily represent the official position of NSF or the U.S. Department of Homeland Security. We thank MAP project team at Dartmouth College and Aruba Networks for the constructive discussions on the proposed detection method. David Martin also provided valuable comments on an early draft of this paper. We also thank the Dartmouth CRAWDAD team and the ICSI/LBNL group who made efforts to release the network traces used in our evaluation.

References

[1] A. Bittau, M. Handley, and J. Lackey, "The Final Nail in WEP's Coffin," in *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2006.

[2] P. Bahl, R. Chandra, J. Padhye, L. Ravindranath, M. Singh, A. Wolman, and B. Zill, "Enhancing the Security of Corporate Wi-Fi Networks Using DAIR," in *Proceedings of the Fourth International Conference on Mobile Systems, Applications, and Services*, Uppsala, Sweden, June 2006.

[3] U. Deshpande, T. Henderson, and D. Kotz, "Channel Sampling Strategies for Monitoring Wireless Networks," in *Proceedings of the Second Workshop on Wireless Network Measurements*, Boston, MA, Apr. 2006.

[4] F. Bulk, "Safe inside a Bubble," Networkcomputing.com, June 2006. [Online]. Available: http://www.networkcomputing.com/channels/wireless/showArticle.jhtml?articleID=189400826

[5] S. Garg, M. Kappes, and A. S. Krishnakumar, "On the Effect of Contention-Window Sizes in IEEE 802.11b Networks," Avaya Labs Research, Tech. Rep. ALR-2002-024, 2002.
[6] W. Wei, S. Jaiswal, J. Kurose, and D. Towsley, "Identifying 802.11 Traffic from Passive Measurements Using Iterative Bayesian Inference," in *Proceedings of the 25th Annual Joint Conference of the IEEE Computer and Communications Societies*, Barcelona, Spain, Apr. 2006.
[7] X. Wang and D. S. Reeves, "Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays," in *Proceedings of the 10th ACM Conference on Computer and Communications Security*, Washington, DC, Oct. 2003, pp. 20–29.
[8] M. Rodrig, C. Reis, R. Mahajan, D. Wetherall, and J. Zahorjan, "Measurement-based characterization of 802.11 in a hotspot setting," in *Proceeding of the ACM SIGCOMM Workshop on Experimental Approaches to Wireless Network Design and Analysis*, Philadelphia, PA, Aug. 2005, pp. 5–10.
[9] A. Wald, *Sequential Analysis*. John Wiley & Sons, 1947.
[10] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan, "Fast Portscan Detection Using Sequential Hypothesis Testing," in *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, Berkeley, CA, May 2004, pp. 211–225.
[11] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.
[12] R. Pang and B. Tierney, "A First Look at Modern Enterprise Traffic," in *Proceedings of the Fifth ACM Internet Measurement Conference*, Berkeley, CA, Oct. 2005, pp. 15–28.
[13] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, Aug. 2000.
[14] "MAP: Security through Measurement for Wireless LANs," Dartmouth College, July 2006, http://www.cs.dartmouth.edu/~map/.
[15] R. Beyah, S. Kangude, G. Yu, B. Strickland, and J. Copeland, "Rogue access point detection using temporal traffic characteristics," in *Proceedings of IEEE Global Communications Conference*, Dec. 2004.
[16] P. Bahl and V. N. Padmanabhan, "RADAR: An In-Building RF-Based User Location and Tracking System," in *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, Tel Aviv, Israel, Mar. 2000.

APPENDIX

We are given a finite set of APs and a finite set of sniffers that are used to monitor these APs. Each AP operates on a specific channel. A sniffer is a network device that can monitor a set of APs at the same time as long as they satisfy the following three conditions:

1) These APs operate on the same channel.
2) These APs are within the sensing range of the sniffer.
3) The sniffer is tuned to the same channel of these APs.

We consider the following optimization problem:

MINIMUM-COST WIRELESS NETWORK SNIFFERS (MWNS)

Instance: A finite set T of APs and a finite set F of sniffers. Each sniffer can monitor a subset of APs with appropriate channel tuning.

Output: A smallest subset of sniffers that can monitor all APs with the minimum number of tuning.

Theorem 1: MWSN is NP-hard.

To show that Theorem 1 is NP-hard, we consider the following decision version of MWNS:

WIRELESS NETWORK SNIFFERS (WNS)

Instance: (T, F, α, β), where T is a finite set of APs, F a finite set of sniffers, $\alpha > 1$ and $\beta \geq 1$ are integers.

Question: Does there exist a subset $F' \subseteq F$, with $|F'| \leq \alpha$, such that all APs in T can be monitored by sniffers in F' with at most β times of tuning?

The following lemma is straightforward:

Lemma 2: If MWNS is solvable in polynomial time, then so is WNS.

Thus, to show that MWNS is NP-hard, it suffices to show that WNS is NP-complete.

Theorem 3: WNS is NP-complete.

Proof: We will reduce Minimum Cover to WNS. Recall that Minimum Cover is the following NP-complete problem (Garey and Johnson 1979):

MINIMUM COVER (MC)

Instance: Collection C of subsets of a finite set S, positive integer $K \leq |C|$.

Question: Does C contain a cover for S of size K or less, i.e., a subset $C' \subseteq C$ with $|C'| \leq K$ such that every element of S belong to at least one member of C?

Let (S, C, K) be a given instance of MC. Define a polynomial-time computable reduction f as follows:

$$f(S, C, K) = (T_S, F_C, K, 0),$$

where T_S is equal to S, i.e., each element in S is viewed as an AP with the same channel; F_C is the set of sniffers, where each sniffer corresponds to an element in C, i.e., this sniffer has exactly those Ps in this element of C.

Assume that (S, C, K) is a positive instance of MC, that is, there exists $C' \subseteq C$ such that $|C'| \leq K$ and every element in S is included in at least one element in C'. Write

$$\begin{aligned} C' &= \{C_1, \ldots, C_k\} \\ k &\leq K \end{aligned}$$

Let

$$F_{C'} = \{s_1, \ldots, s_k\},$$

where each sniffer s_i covers exactly those APs in C_i. Thus, every AP in T_S is covered by at least one sniffer in $F_{C'}$. Moreover, there is no tuning needed, for all APs have the same channel. Thus, $(T_S, F_C, K, 0)$ is also a positive instance of WNS.

Now assume that $(T_S, F_C, K, 0)$ is a positive instance of WNS. This means that there is a subset of sniffers of size at most K that cover all APs in T_S without tuning. This is equivalent to saying that (S, C, K) is a positive instance of MC. ■