See Like a Bat:
Using Echo-Analysis to Detect Man-in-the-Middle Attacks in LANs

Yisroel Mirsky, Naor Kalbo, Yuval Elovici, and Asaf Shabtai

[1]Department of Software and Information Systems Engineering,
Ben-Gurion University, Beer-Sheva, Israel.

{yisroel, kalbo}@post.bgu.ac.il, {elovici, shabtaia}@bgu.ac.il

Abstract. *Although Man-in-the-Middle (MitM) attacks on LANs have been known for some time, they are still considered a significant threat. This is because these attacks are relatively easy to achieve, yet challenging to detect. For example, a planted network bridge or compromised switch leaves no forensic evidence.*

In this talk, we present Vesper: a novel plug-and-play MitM detector for local area networks. Vesper uses a technique inspired from the domain of acoustic signal processing. Analogous to how echoes in a cave capture the shape and construction of the environment, so to can a short and intense pulse of ICMP echo requests model the link between two network hosts. Vesper sends these probes to a target network host and then uses the reflected signal to summarize the channel environment (think sonar). Vesper uses neural networks called autoencoders to profile the link with each host, and to detect when the environment changes. Using this technique, Vesper can detect MitM attacks with high accuracy, to the extent that it can distinguish between identical networking devices. Vesper is implemented at the software level and is therefore is cross platform. We evaluate Vesper on LANs consisting of video surveillance cameras, servers, and hundreds of PC workstations. We show how Vesper works across multiple network switches and in the presence of traffic. We also investigate several possible adversarial attacks against Vesper, and demonstrate how Vesper mitigates these attacks. Finally, we show how Vesper can be used to fingerprint network devices remotely (e.g., for tamper protection). To demonstrate this, we show how Vesper can differentiate between 40 identical Raspberry Pis.

A version of the vesper tool can be downloaded online.[1]

[1]https://github.com/ymirsky/Vesper

Contents

1. Introduction

A Man-in-the-Middle attack (MitM) is where a malicious third party takes control of a communication channel between two or more endpoints by intercepting and forwarding the traffic in transit. An attacker in the middle has the capability of harming the confidentiality, integrity, and availability of the user's content, by eavesdropping, manipulating, crafting, and dropping traffic on the network. In general, the MitM attack model on a local area network (LAN) has three steps: (1) gain access to the network, (2) intercept traffic in transit, and (3) manipulate, craft, or drop traffic.

Depending on the scenario, access can be achieved by connecting to a public Wi-Fi access point (e.g. at a café, airport...) or by connecting physically to an exposed network cable or network switch. The attacker can also conduct this attack remotely via a malware which has infected a trusted computer within the existing network [1]. After gaining access, interception can be achieved by exploiting known vulnerabilities in network protocols. For example, the attacker can poison a host's address resolution protocol (ARP) table to capture local traffic [2–4], or spoofing a domain name server (DNS) to intercept all web traffic [5–7]. The attacker can easily exploit these vulnerabilities with free tools which work out-of-the-box such as Ettercap, Cain and Abel, Evilgrade, arpspoof, dsniff, and many others.

Although MitM attacks on LANs have been known for some time, they are still considered a significant threat [8, 9], and have gained academic attention over the years. This is likely because the attack is relatively easy to achieve, yet challenging to detect [10]. Encryption can protect the integrity and confidentiality of the traffic in transit. However, according to [11], 30% of the world's web traffic is not encrypted. Furthermore, in many cases networked systems do not encrypt their traffic by default (e.g., SCADA control system [12]). Moreover, even if the traffic is encrypted, encryption protocols may have flaws [13, 14], be misconfigured, or simply left out by a manufacturer (e.g. CVE-2017-15643). We also note that LAN-based MitM attacks are used in APTs to achieve lateral movement [15]. Therefore, there is a need for detecting the presence of a MitM, even when encryption is employed.

1.1. The Proposed Solution

Our proposed solution is inspired by signal processing domain. In a dynamic system, the output (reaction) of the system to a short input signal is called impulse response. A common use of impulse responses is the modeling and recreation of acoustic environments, such as small rooms or concert halls. As an intuitive example, one can hear the IR of a room by clapping their hands. The sound of the clap changes based on the size, shape, and materials of the room.

Using this concept, we propose a MitM detector called *Vesper*. Vesper bats are the largest and best-known family of the bat species. Akin to it name, our detector captures the impulse response of a LAN by measuring the round-trip-times (RTT) resulting from an intense burst of ICMP echo requests. The payload sizes of the ICMP request packets are modulated according to an excitation signal. As a result, the impulse response extracted from the RTTs can be used to model the network environment in the perspective of two communicating hosts. When a third party intercepts traffic, the harmonic composition of the impulse response between the hosts changes significantly. These changes can be

detected using an anomaly detector.

The described impulse response analysis is enough to detect all LAN based MitM attacks. However, although the described echo analysis seems difficult to evade, there are several adversarial attacks which can be performed. For example, Eve can spoof the ICMP replies on behalf of the victim or even replay previously recorded bursts. Therefore, *Vesper* monitors three features in total to resist adversarial attacks: the impulse response's energy, the overall delay, and the packets' jitter distribution. The three features are weak at detecting some adversarial attacks, but strong at detecting others. Therefore, when combined, these features provide good protection against adversarial attacks.

Altogether, *Vesper* (1) probes a link with an end-host with a modulated ICMP excitation signal, (2) extracts three features from the response, and (3) detects MitM attacks and potential adversarial attacks using an autoencoder neural network as an anomaly detector. In this talk, we show how *Vesper* works with various devices, in the presence of diverse traffic, across multiple switches, and for long durations of time. We also show how *Vesper* is robust against adversarial attacks, and can differentiate between devices with identical hardware.

The current version of the Vesper MitM detection tool can be downloaded from `https://github.com/ymirsky/Vesper`. The tool is written in python and wraps C/C++ code using cython. The the current version only works on Linux machines and has been tested on Kali. We plan to port Vesper to Windows in the near future.

1.2. Other MitM Detection Methods

In the past decade, many different detection schemes have been proposed in order to address MitM attacks. In general, the solutions to MitM attacks on LANs address a specific flaw in a protocol [3, 5, 6, 16, 17]. As an example, consider the infamous vulnerability in the Address Resolution Protocol (ARP). The vulnerability gives untrusted hosts the ability to spread spoofed ARP messages, causing network traffic to be routed to the attacker's device. Solutions to this flaw include improved protocols [2, 18, 19] and the integration of new security features [20].

Intrusion Detection Systems (IDS) have been proposed as a more generic way for dealing with MitM attacks. These IDSs include software-based IDSs [21] and hybrid hardware/software IDSs (e.g., an add-on component plugged into the mirror port of a switch) [10].

However, these solutions have limitations:

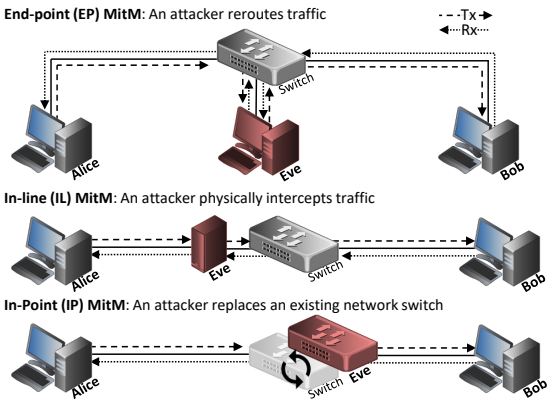
Generalization. Many of these solutions address a flaw in a specific protocol, and therefore cannot be generalized to other or unknown MitM attacks occurring in the LAN. For example, detecting an exploitation of the ARP protocol does not solve the issue of an IL MitM.

Portability. Some of these solutions require additional hardware or other costly resources. For example, a separate network host which acts as an IDS.

False Positives. Network traffic tends to be noisy, making it difficult to detect the presence of a MitM based on traffic patterns and packet contents. Therefore, searching network packets for evidence of a MitM may lead to a large number of false positives [10].

Regardless, all of the related solutions are weak to IL MitM attacks, since they leave no forensic evidence in the packets. On the other hand, *Vesper* does not require additional hardware and can detect EP, IL, and IP MitM attacks. Furthermore, *Vesper* is robust since it analyzes its own probes and not the traffic of others.

The rest of this talk is organized as follows. In section 2, we present the MitM attack model. In section 3, we provide a background on echo analysis, and introduce our technique (ping echo analysis). In section 4, we present the framework for MitM detection in LANs via ping echo analysis (*Vesper*). In section 5, we present evaluations of *Vesper* on several different networks. In section 6, we present possible adversarial attacks against *Vesper* and the respective countermeasures. Finally, in section 8, we conclude our talk.



(a) The three LAN MitM attack topologies.

TD: Traffic Diversion **EP**: End-Point
NB: Network Bridge **IL**: In-Line
DH: Dedicated Hardware **IP**: In-Point
- low, ○ medium, ● high, ★ very-high

		MitM				Wiretap
		1	**2**	**3**	**4**	5
Type	*Topology*	EP	IL	IL	IP	IL
	Implementation	TD	NB	DH	DH	DH
Attack Vectors	*Gain physical access to switch*	X	X	X	X	X
	Connect to Ethernet wall socket	X				
	Install malware on existing host	X				
	Install device on strategic wire		X	X		X
Traffic Intercepted	*All traffic in LAN*	X				
	Local traffic only		X	X	X	X
Can the MitM alter and inject traffic?		X	X	X	X	
Complexity of the MitM attack		-	○	●	★	○
Vesper's detection of the MitM attack*		★	★	●	○	NA

*Based on the average Equal Error Rates (EER) from the evaluations: very-high: ≤ 0.003, high: 0.003-0.005, medium: 0.005-0.01, low: > 0.01

(b) Summary of MitM Attacks on a LAN

Figure 1: A summary of the MitM attack topologies, implementations, under the scope of Vesper's detection.

2. The Attack Model

In this section we describe the MitM attack model used throughout the talk. We also enumerate the attacker's requirements, attack vectors, and capabilities.

2.1. Attack Scenario

Let Alice and Bob be victims located on the same LAN segment, where the LAN segment may contain one or more network switches. Let Eve be the attacker whose objective is to perform a MitM attack between Alice and Bob. In other words, Eve wants to manipulate the traffic sent between Alice and Bob, while being able to craft new traffic as well (e.g., sending ARP packets). Eve has physical access to the LAN's infrastructure, and can install malware on a network host other than Alice and Bob.

2.2. Attack Topologies

Eve can accomplish her objective by establishing one of the following MitM topologies (illustrated in Fig. 1a):

End-Point (EP) MitM. Eve either adds a new host, or compromises an existing host on the LAN. Eve then causes the traffic in transit between the Alice and Bob to flow through her device (e.g., via ARP poisoning or some other protocol-based MitM attack).

In-Line (IL) MitM. Eve locates an exposed network cable which Alice and Bob use to communicate. Eve then covertly installs a device which passes all traffic from one side of the wire to the other, while being able to manipulate/inject traffic.

In-Point (IP) MitM. Eve locates an exposed network switch which Alice and Bob use to communicate. Eve then swaps the switch with a new switch that has additional logic enabling her to manipulate/inject traffic.

Unlike the EP MitM, IL and IP MitM attacks can only be accomplished by introducing additional hardware. Therefore, these attacks require physical access to the LAN.

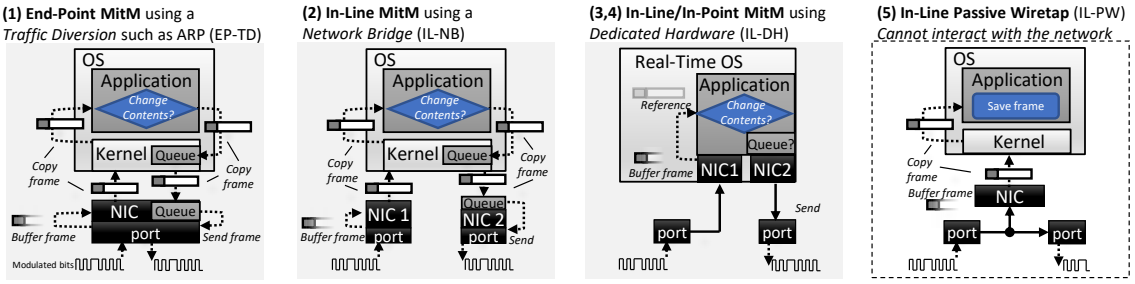


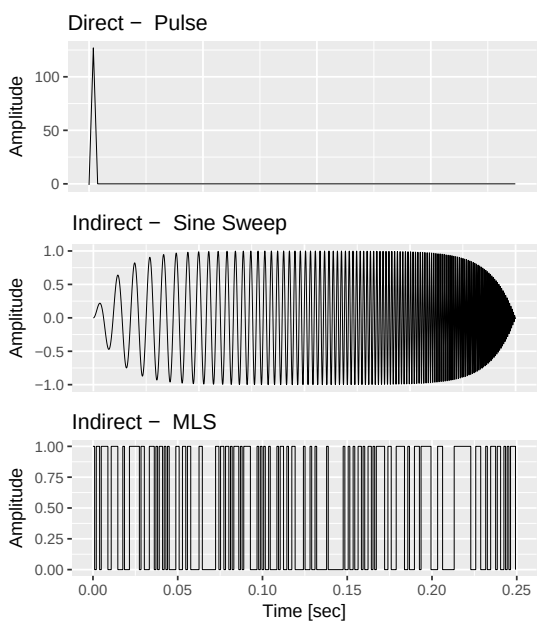
Figure 2: The packet interception process for (1) an EP MitM, (2-3) an IL MitM, (4) an IP MitM, and (4) a passive wiretap.

2.3. Classes of Attacks

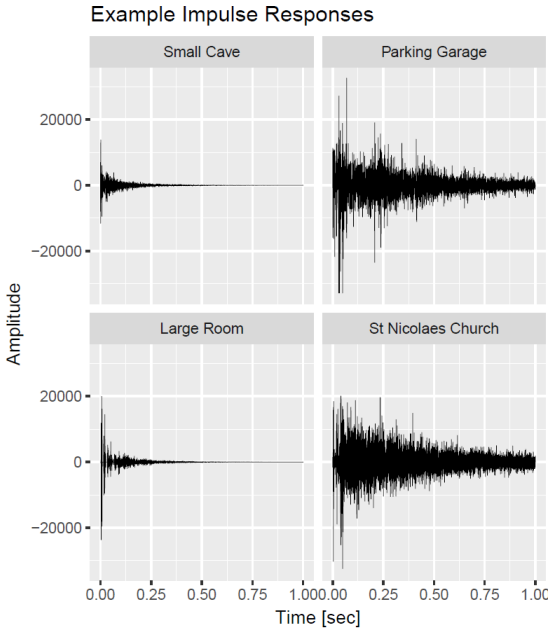
MitM attacks in a LAN vary based on their stealth and complexity. For example, a more covert attack is typically more difficult for the attacker to accomplish. We categorize the class of a MitM attack based on the MitM topology, and implementation used. Table 1b summarizes these classes, and their notations which we use throughout the talk.

We note that although an In-Point Dedicated Hardware (IP-DH) MitM attack is very hard to detect, it is also very hard for the attacker to accomplish. This is because (1) network switches are typically stored under lock and key, and (2) modern switches provide a password protected administrator console (the attacker must copy the configurations prior to the swap).

There are two reasons why such a MitM will buffer each and every inbound packet: (1) to avoid signal collisions on the media when transmitting crafted/altered packets, and (2) to capture and alter relevant packets before they reach their intended destination. In the latter case, the MitM must parse every frame in order to determine the frame's relevancy to the attack, and cannot retroactively stop a transmitted frame. Therefore, the interception process (hardware and/or software) will affect the timing of network traffic. We note that since passive wiretaps only observe traffic, they are not MitM attacks and therefore not in the scope of this talk. However, *Vesper* can detect a MitM which is presently eavesdropping (not currently altering traffic) because a MitM always buffers each packet upon reception. Fig. 2 illustrates the basic packet interception process for all MitM implementations.



(a) Two common excitation signals used to extract impulse responses from acoustic environments.



(b) Example impulse responses extracted from various acoustic environments.

Figure 3: A summary of the MitM attack topologies, implementations, under the scope of Vesper's detection.

3. Echo Analysis

In this section, we present the probing technique used by *Vesper* to capture the presence of a MitM. Later in section 4, we show how *Vesper* uses this technique to actively detect MitM attacks.

3.1. Acoustic Signal Processing

Our technique is inspired by the domain of acoustic signal processing. Therefore, we will now briefly cover this topic to give you a bit of a background.

In the domain of acoustic signal processing, a sound which reverberates through the air, and the environment (e.g., room) which reflects and affects the vibrations as they propagate, are the signal x and LTI system S respectively. An acoustic engineer can model S by extracting its impulse response h. This can be achieved by emitting an excitation signal x at one location while simultaneously recording the resulting signal y at another location. In this case, the input to S is generated by a speaker and the output is captured by a microphone.

There are several methods for extracting an acoustic impulse response with an excitation signal. These methods can be categorized as either being direct or indirect (visualized in Fig. 3a):

Direct Methods. Direct methods involve an excitation signal x which is similar to that of a Dirac function, so that $y = h$. However, since it is impossible to generate a true Dirac signal in an acoustic environment, short loud sounds are used instead. For example, popping a balloon, generating a spark, and firing a gun.

Indirect Methods. An approximation of h can be obtained *indirectly* from a non-Dirac excitation signal. The process involves deconvolving the excitation signal x with the resulting output signal y [22]. One well-known excitation signal is the maximal length sequence (MLS) signal. An MLS is a pseudorandom binary sequence generated from maximal linear feedback shift registers. With m registers, the generator produces a random binary sequence of length $N = 2^m - 1$ which is spectrally flat. As a result, an MLS excitation signal produces all frequencies, closely resembles white noise, and is robust in noisy and populated environments [23]. A code snippet is available in this talk's the supplementary material, and the reader may download our MLS generator's source code and demo online.[2]

Once the impulse response h has been extracted from S, it can be used to perform a convolution reverb (a digital simulation of an audio environment on sound). For example, the response can be used to make a recorded piece of music to sound like it was played in a particular cave or arena. We can see from this that h is dependent on the shape of the room, the materials of the surfaces, and the positioning of the speaker and microphone. Any alteration to these physical parameters will cause a noticeable affect on the impulse response. In other words, an impulse response can be seen as an *acoustical signature* of the environment.

To illustrate this concept, Fig. 3b presents four impulse responses extracted from different environments. The initial Dirac pulse (e.g., balloon pop sound) can be seen at the beginning, followed by dynamic reverberations and echoes (i.e., spikes). The figure shows that each environment has its own unique signature due to their unique constructs.

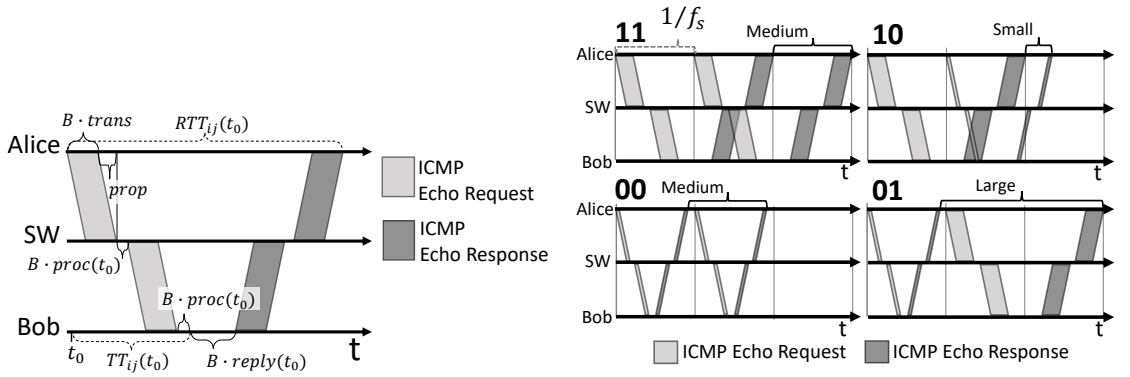
3.2. Ping Signal Processing

Our approach to MitM detection is to (1) model a LAN as an acoustic environment, (2) emit excitation signals, (3) model the echoed response signals, and (4) detect abnormal changes in newly sampled responses.

In networks, there are no reverberations of sound waves. However, switches, network interfaces, and operating systems all affect a packet's travel time across a network. The hardware, buffers, caches, and even the software versions of the devices which interact with the packets, all affect packet timing. When a device processes a burst of packets, the device has dynamic reaction with respect to the packets' sizes. This affects the packets' processing times, which are in turn, then propagated to the next node in the network. This is analogous to how a sound wave is affected as it reverberates off various surfaces.

To capture packet timing between a local host and an end-host, one can use the Internet Control Message Protocol (ICMP) [24]. The ICMP is a popular protocol used to gain feedback about problems in an IP network. One of the features of this protocol is the `Echo_Request` command, commonly used to determine whether a host is operational. When a host sends another host an `Echo_Request`, the target host returns an ICMP `Echo_Reply`. Upon receiving the `Echo_Reply`, the sender can measure the round-trip-time (RTT) to and from the receiver. This process is referred to as 'pinging'. To according to the ICMP standard (RFC 1122: 3.2.2.6), one may include data (a payload) in an `Echo_Request`. In this case, the receiver must include the same data in the `Echo_Reply`.

[2] `https://github.com/ymirsky/MLSgen`



(a) An illustration of the ICMP RTT model. The propagation time has been exaggerated for visualization purposes.

(b) The affect which two subsequent packets (modulated between max and min payload sizes) have on jitter.

Figure 4: The behavior of ping RTT times in a LAN.

The RTT of an ICMP packet over a LAN is dependent on the number of switches (hops) traversed, since interactive networking elements (e.g., switches) must fully buffer each received frame before transmitting. The RTT is also dependent on the current load and the hardware/software implementation of each networking element along the path of the ping. Fig. 4a visualizes the timing of a ping to a host and back over a switch, and Fig 4b visualizes how two packets with different sizes sent quickly at a set rate affect jitter (inter-packet arrival times). Note, $proc$ denotes the packet processing time and $reply$ denoted the `ICMP ECHO` response processing time –both of which dynamically change according to the load, model, software, and other characteristics.

3.2.1. System Definition (S)

Let S be a LAN environment consisting of one or more switches and numerous hosts. Let S_{ij} be the LAN in the perspective of host i communicating with host j, where i and j are within the same LAN.

Let g be our generator signal which produces our system's input signal x. The signal g is defined as a sequence of ICMP `Echo_Request` frames transmitted at a rate of f_s Hz, where $g[n] \in \{42, 43, \ldots, 1542\}$ are the number of bytes which are transmitted in the ICMP `Echo_Request`: 42 bytes for the Ethernet, IPv4, and ICMP protocol headers, plus an additional 0-1500 bytes for the ICMP payload).

Let the input signal x be defined as the packet transmission times as a result of g. More formally,

$$x[n] = g[n] \cdot trans \tag{1}$$

Finally, let the output signal y be defined as the sequence of RTTs, computed from the respective ICMP `Echo_Reply` packets' arrival times. More formally,

$$y[n] = T_{rx}[n] - T_{tx}[n] \tag{2}$$

where $T_{tx}[n]$ is the transmission timestamp of the n-th `Echo_Request` in x, and $T_{rx}[n]$ is the reception time of the resulting `Echo_Reply`.

It is clear that there is a direct relationship between the packet transmission times x and the RTTs y as generated by the signal g (evident in Fig. 5).

3.2.2. Ping Excitation Signal (x)

In order to capture a characterization of S_{ij}, we use an MLS excitation signal as the generator signal g.

To transmit a binary MLS signal s, we modulate is over g as the minimum and maximum ICMP payload sizes. For example, one possible $N = 7$ length MLS may be $s = \{1, 1, 1, 0, 1, 0, 0\}$. In this case, s would be translated into the transmission signal $g = \{1542, 1542, 1542, 42, 1542, 42, 42\}$. Fig. 5 illustrates an $m = 10$ bit ($N = 1023$ length) sequence (g) modulated as the input signal x, and then received as the output signal y.

There are several reasons why we use the MLS method over other known excitation methods:

- The MLS method is known to be robust in noisy environments, such as a room populated with conversing individuals [23]. Network traffic can affect S_{ij}, thus it is appropriate to assume the system will be noisy.
- An MLS of sufficient length has subsequences of '1's. This results in bursts of pings which have the maximum size of 1500. This burst causes a momentary stress on the network which is reflected in the output y, thus better capturing the network's characteristics.
- An MLS is randomly generated each time, thus raising the difficulty for an attacker to perform a replay attack (discussed in detail later in section 6).

When the random sized packets of the modulated MLS signal g are sent back-to-back at a fast rate, the electronics, caching mechanisms, CPU schedulers, and queuing algorithms of each network element dynamically affect the respective $proc(t)$ and $reply(t)$ in response to the varying load. Since the transmission times x of the payloads in g reflect an MLS signal, y captures S_{ij}'s fingerprint (impulse response).

Empirical evidence can be shown via linear regression. We found that the k-th RTT in y has a dependency on the *random* payload sizes of previously transmitted ICMP requests in x.In Fig. 6, we illustrate this dependency. The figure shows the RTT distribution of the i-th ping sent in a burst of 50 pings, over 1500 trials. For example, the first box plot is the RTT distribution of the first ping sent in each of the 1500 bursts. If there were no dependency, then all of the distributions would have been the same. However, the figure shows that the system has a dynamic response to the burst of packets. I.e., we get much better accuracy if we model the response and not just naively average the RTTs times.

Fig. 6 also shows that the first pings are noisier than those which follow (e.g., due to caching). This is another reason why we must send g at a fast rate, and not as individual pings. In our system, we set the transmission rate of g to

$$f_s \equiv \frac{2}{\mu_{RTT^*}} \tag{3}$$

where μ_{RTT^*} denotes the average RTT time of largest ping possible (a 1542 byte Ethernet

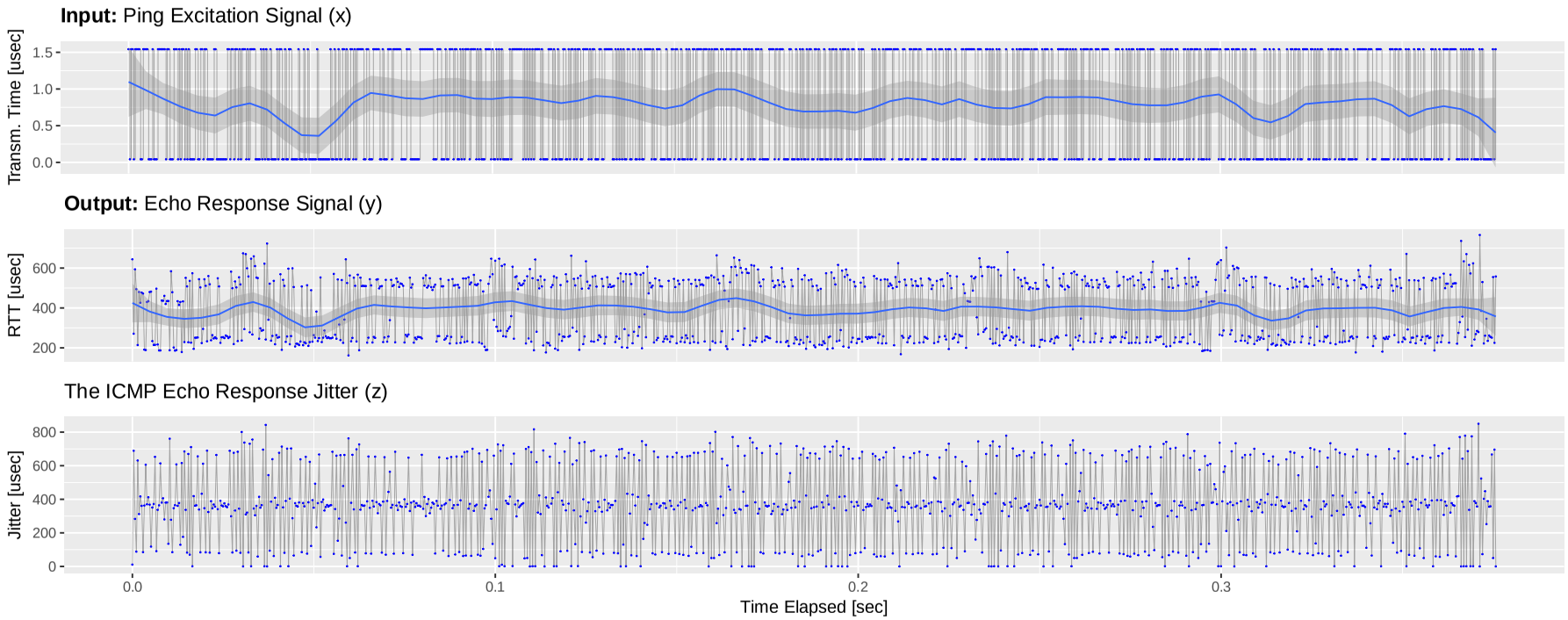


Figure 5: Sample signals from a Linux PC (i) and an IP-based security camera (j). Top: the 10 bit MLS ping excitation signal x, sent from i to j. Middle: the resulting ping echo response signal y, timed by i. Bottom: the ICMP echo response jitter z, computed by i. Signals x and y have been fitted to a Loess curve (blue line) for better visualization.

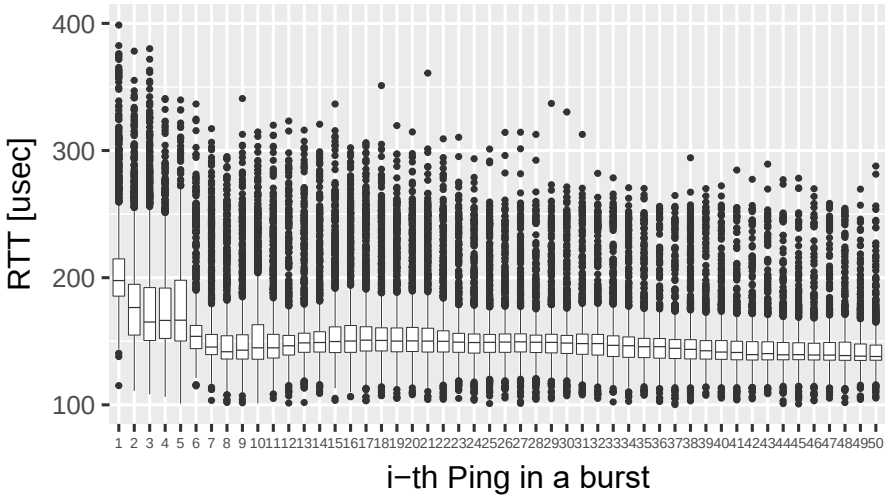


Figure 6: The RTT distribution of the i-th ping in a burst of 50 packets. The data was collected over 1500 trials with a direct host-to-host Ethernet connection.

frame). This rate ensures that y captures the system well, while not overloading the end-host.

3.2.3. ICMP Echo Response Jitter (z)

In order to form a robust MitM detector and to resist adversarial attacks, we also examine the jitter of the ICMP echo response signal. Jitter is the time lapse between two consecutive packet arrivals. We denote the jitter values of the ICMP echo responses as z, defined as

$$z[t] = T_{rx}[t] - T_{rx}[t-1] \tag{4}$$

The bottom of Fig. 5 plots an example of the jitter resulting from the MLS signal x. In this example, we can see that the jitter is distributed at distinct high (650 usec), medium (390 usec), and low (50 usec) levels. The three distinct levels are the result of the transitions between adjacent bits in the MLS binary sequence. For instance, whenever a '10' appears in the sequence, the jitter is small. This is because the RTT of a 42 byte packet ('0') is shorter than that of a 1542 byte packet ('1'). Since the pings are sent at a rate of f_s, the response for the '0' arrives shortly after the response for the '1'. Fig. 4b illustrates the effect of all four possible transitions.

Although z is not part of our system model S_{ij}, it captures additional characteristics of the channel between i and j. For example, additional processing delays and moments of stress on the participating network elements.

4. Vesper

In this section, we present the MitM detector *Vesper*: the framework, machine learning process, and deployment.

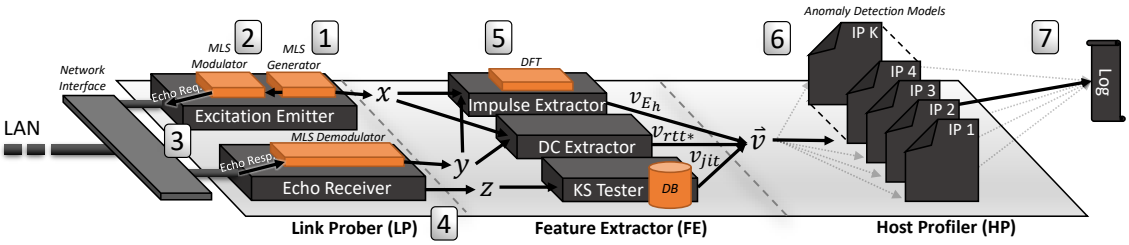


Figure 7: The framework of *Vesper*, deployed on a network host.

4.1. Overview

Vesper is a *plug-and-play* man-in-the-middle detector based on ping echo-analysis. The detector is installed on a local host within a LAN, and protects the local host from MitM attacks originating from within the same LAN. In this section, we use Γ to denote the set of known remote hosts in the LAN, excluding the local host.

Vesper's framework has the following main components:

- **Orchestrator (OR):** The component responsible for adding new local IPs (hosts) automatically, and deciding which link in the LAN should be probed when.
- **Link Prober (LP):** The component responsible for probing the hosts in Γ. Each probe produces an MLS excitation signal (x), which results in the echo response signal (y) and the echo response jitter (z).
- **Feature Extractor (FE):** The component responsible for summarizing the result of a probe. The summary forms a feature vector $\vec{v} \in \mathbb{R}^3$.
- **Host Profiler (HP):** The component responsible for detecting the presence of a MitM using $\vec{v}$. It accomplishes this by profiling the link to each host $j \in \Gamma$ with an autoencoder. The autoencoder is trained to recognize the link's normal behavior. An autoencoder is a neural network which can be used as an anomaly detector (discussed later in section 4.5).

Vesper operates by performing the following steps (illustrated in Fig. 7):

Vesper's MitM Detection Procedure

I. Orchestrator

1. At a random time, a random network host $j \in \Gamma$ is selected and the detection process is initiated.

II. Link Prober

2. The *MLS Generator* produces the random binary sequence s.
3. The *MLS Modulator* creates the ping excitation signal x based on the binary sequence s.
4. The *Excitation Emitter* sends host j a total of N `Echo_Request` packets, according to x, and at a rate of f_s. In parallel, the *Echo Receiver* captures j's

`Echo_Reply` packets.

5. Once all N `Echo_Reply` packets have been received, the *MLS Demodulator* extracts the echo response signal y and the echo response jitter z.

III. Feature Extractor

6. The *Impulse Extractor*, *DC Extractor*, and *KS-Tester* use x, y, and z to produce the feature vector $\vec{v}$.

IV. Host Profiler

7. The IP address of j is used to retrieve j's autoencoder via a hashmap.
8. Using $\vec{v}$, the autoencoder determines whether or not the link with host j has been significantly altered. If $\vec{v}$ is determined to be normal (with high confidence), then the autoencoder learns from the instance $\vec{v}$. Otherwise, an alert is raised.

We will now discuss the each of *Vesper*'s main components in greater detail.

4.2. Orchestrator (OR)

Whenever a new IP address from the same subnet as *Vesper* is observed in the network traffic, or added by the user, the OR pings that address. If none of the pings traverse a router (indicated by `TTL` field of the IPv4 header) then the address is added to Γ. After sending each probe, at a random time within the next second, the OR selects a random host $i \in \Gamma$ and initiates a probe via the LP.

4.3. Link Prober (LP)

After generating s and x, the LP transmits `Echo_Request` packets to the target host, according to x. The `Echo_Request` packets are transmitted every $\frac{1}{f_s} = \frac{1}{2}\mu_{RTT^*}$ seconds. This means that the transmission and reception of ICMP packets must be performed concurrently on two separate threads: the *Excitation Emitter* and *Echo Receiver*.

In order to measure the RTT of each ping correctly, each `Echo_Request` must be associated with its respective `Echo_Reply`. To accomplish this, the *Excitation Emitter* places the current index of x into the `Sequence_Number` field of the `Echo_Request` header. When a host replies, it copies the same value from the `Echo_Request` into the header of its `Echo_Reply`. The `Identifier` field is used to differentiate between separate excitation signals.

In order to obtain the necessary accuracy, the LP records all transmission and reception timestamps with nanosecond resolution. In C++, and with a Linux kernel, this can be accomplished using the `<time.h>` library's `clock_gettime()` with the `CLOCK_MONOTONIC` option enabled.[3]

When the last `Echo_Reply` is received, y and z are computed, and the raw probe data (x, y, z) is passed to the FE for feature extraction.

[3]The API instructs the OS to collect the time from a CPU register. On a Dell PC, we found the error to be at worst 0.015% w.r.t. the smallest possible RTT (150 usec with a host-to-host direct Ethernet link).

4.4. Feature Extractor (FE)

After each probe, the FE is tasked with extracting the following three features from (x, y, z):

v_{E_h}:	The impulse response energy using x and y
v_{rtt*}:	The mean RTT from the largest packets sent in x
v_{jit}:	The log-likelihood of the jitter's distribution (z)

The feature vector $\vec{v} = \{v_{E_h}, v_{rtt*}, v_{jit}\}$ summarizes the state of the probed channel (system S_{ij}). After the FE computes $\vec{v}$, the vector is passed to the HP for inspection. Fig. 8 plots each of the features before and after a MitM attack.

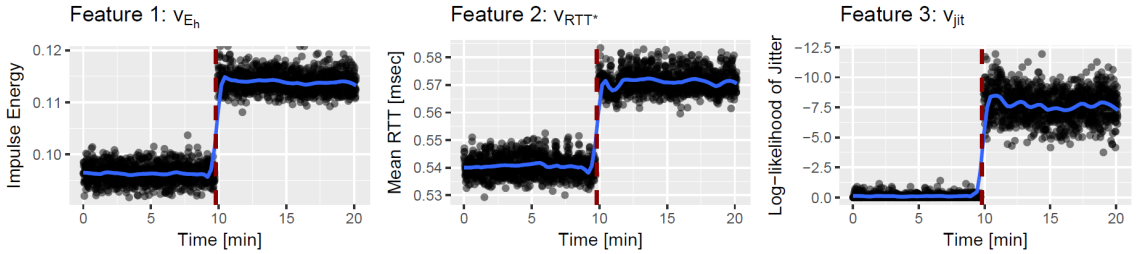


Figure 8: The three features extracted each probe over time where the dashed line indicates the start of an IL-DH MitM attack.

4.4.1. Impulse Response Energy (v_{E_h})

When Eve intercepts Alice's traffic, Eve affects the dynamics of the channel between Alice and Bob. Even when Eve responds to Alice's ICMP traffic on behalf of Bob, the impulse response h, captured by y, changes. This is because Alice has different hardware and software than Bob. To exemplify this concept, Fig. **??** shows the average impulse response h obtained from x and y in the most extreme cases: when an intermediary switch is swapped with an identical switch (both *D-Link DGS-1100*), and when an end-host $j \in \Gamma$ is swapped with an identical device (both *Raspberry Pi 3B*). In both cases, h captures the variations and defects in the hardware of the swapped devices. Thus, h acts as a remote physically unclonable function (PUF) [25] which captures the state of the system S_{ij}, where i is the local host.

The *Impulse Extractor* summarizes the state of the system S_{ij} by measuring the energy of h, denoted E_h. As mentioned in section 3.1, through a process called deconvolution, the response h can be obtained from the output y by knowing the excitation signal x.

In a linear time invariant system, the output can be expressed as

$$y[t] = h[t] * x[t] \tag{5}$$

where $*$ is the convolution operator, and h is the system's impulse response. To extract impulse response in the absence of noise, we can perform the deconvolution

$$h[t] = \mathcal{F}^{-1}\{Y/X\} \tag{6}$$

where $\mathcal{F}^{-1}$ is inverse of the Discreet Fourier Transform (DFT), and where Y and X are the Fourier Transforms of the signals y and x respectively.

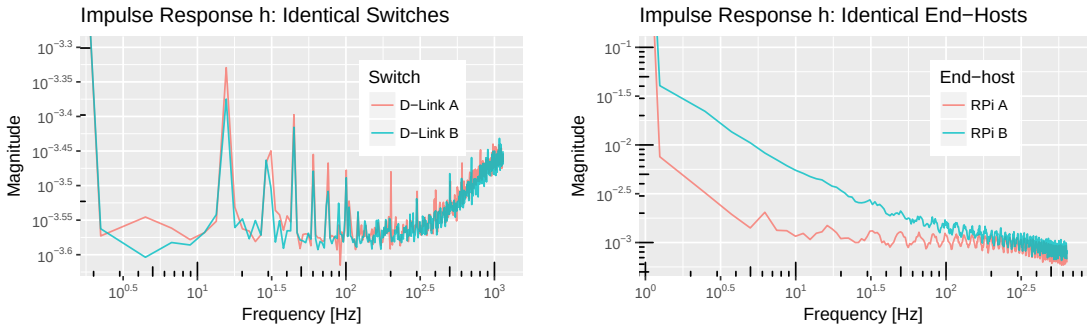


Figure 9: The average impulse response of 100 probes in the case where (top) the intermediary switch is swapped with an identical one, and (bottom) the target end-host is swapped with an identical device (over the same switch).

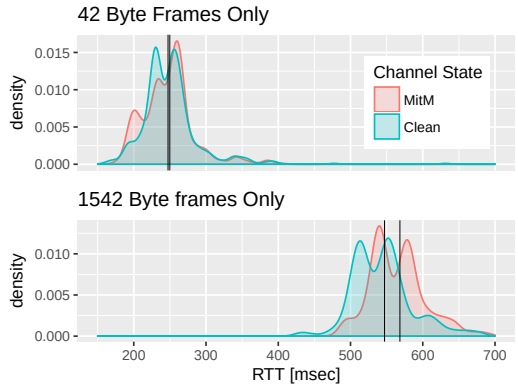


Figure 10: The distribution of y's RTTs, when exclusively considering the 42 byte or 1542 byte frames, with and without an IL-DH MitM. The bars mark each of the means.

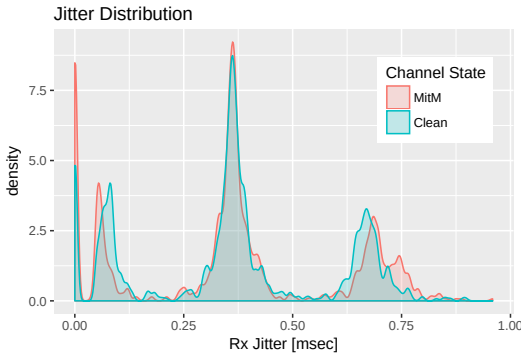


Figure 11: The distribution of a probe's response jitter (z), with and without the presence of an IL-DH MitM.

By assuming periodicity, we can use Parseval's theorem [22] to obtain E_h without the need for computing the inverse DFT in (6) by computing

$$E_h = \frac{1}{N}\sum_{k=1}^{N}\left|\frac{Y[k]}{X[k]}\right|^2 \tag{7}$$

The resulting value used as the feature v_{E_h} in $\vec{v}$.

4.4.2. Mean RTT of the Largest Packets (v_{rtt*})

In Fig. 1a, it can be seen that a both MitM attack scenarios add an additional $2 \cdot T_{hop}$ delay to Alice's traffic. We can detect this additional delay by averaging the values (RTTs) in y.

We note that the duration of T_{hop} increases with the length of the Ethernet frame. Approximately 50% of the packets in x have the maximum length of 1542 bytes. By averaging the RTTs of those frames only, we obtain a better separation between benign

the malicious scenarios. Fig. 10 illustrates the benefit of averaging the 1542 byte frames in each response y.

This average is extracted from each y by the *DC Extractor*, and used as the feature v_{RTT*} in $\vec{v}$.

4.4.3. Log-likelihood of the Jitter's Distribution (v_{jit})

As mentioned in section 3.2.3, the jitter of the `Echo_Reply` packets (z) captures the behavior of the networking elements between the sender and receiver. Concretely, since x is transmitted at a rate of f_s, it can be expected that some packets may be being queued, and then transmitted back-to-back. This dynamic behavior characterizes the network's elements, and thus fingerprints the connection with host j.

Fig. 11 plots z's distribution, with and without the presence of a MitM attack. Fig. 11 shows that the three levels of jitter (refer to section 3.2.3) are affected by the attack. To detect abnormalities in this distribution, the FE performs a two-sample Kolmogorov-Smirnov (KS) test. The KS test is a nonparametric statistical test which results in a probability value (p-value) that indicates how likely two sample distributions come from the same distribution. We denote this value as $p_{X,Y}$, where X and Y are tested distributions.

The *KS Tester* stores m samples of host j's jitter distributions. These samples are used as references for the channel's expected behavior. We denote host j's references as the set $Z_j = z_1, z_2, \ldots, z_m$. Although m is a parameter of $Vesper$, in practice $m = 5$ works well.

Let z_0 denote the jitter distribution given to the FE for feature extraction. With z_0, the *KS Tester* computes the value

$$p_{jit} = max\{p_{z_0,z_1}, p_{z_0,z_2}, \ldots p_{z_0,z_m}\} \geq \phi \tag{8}$$

where ϕ is a probability threshold (we use 0.1 in our tool). The last k computations of p_{jit} (from previous probes) are averaged to form the feature v_{jit}. In practice, we found that $k = 15$ produces good results. In (8), we take the maximum p-value, since this makes the feature more robust against false positives.

4.5. Host Profiler (HP)

The HP component uses local outlier factor (LOF) to perform the basic anomaly detection. First, we will explain in how LOF works, and then we will explain how the HP uses them to detect anomalies in the link with host j.

4.5.1. Local Outlier Factor

Local Outlier Factor (LOF) is an anomaly detection algorithm based on the lazy learner concept of the K-nearest neighbors (KNN) algorithm. The KNN algorithm is used for anomaly detection as follows:

Let a dataset X of m numerical points (observations/feature vectors) which represent normal behavior. To determine whether or not a new observation x we perform the following:

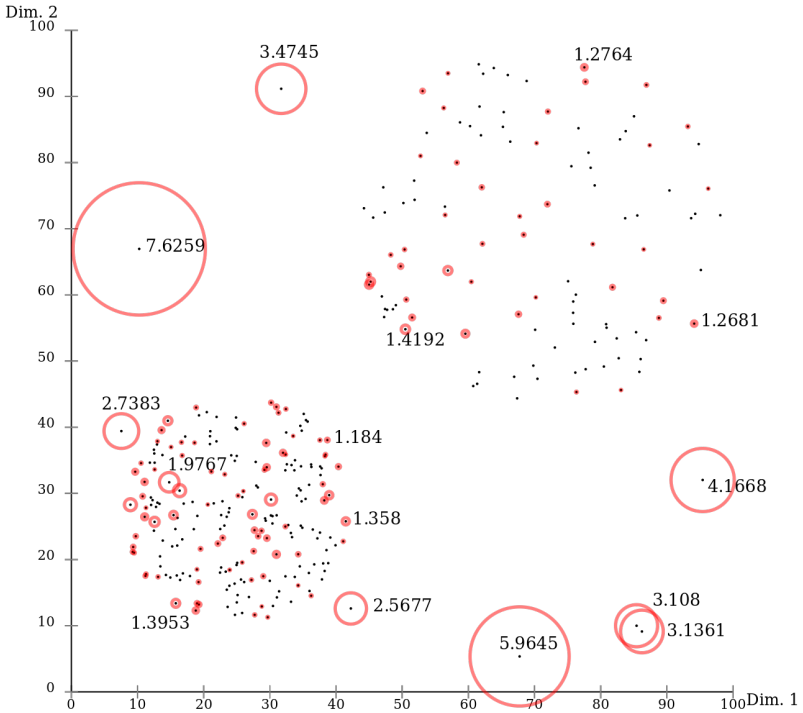


Figure 12: LOF scores visualized. While the upper right cluster has a comparable density to the outliers close to the bottom left cluster, they are detected correctly.

1. Retrieve the K nearest neighbors to x in X
2. compute the average distance from each neighbor and x
3. If the average distance is over some set threshold, then x is an outlier (abnormal), otherwise, x is an inlier (normal).

The KNN anomaly detection algorithm works quite well, except when the data has varying densities. In which case, the concept of distance to ones' neighbors becomes contextual. LOF resolves this by considering the local density of the discovered neighbors. The anomaly score for x is a value on the range $[0, \inf)$ where scores below 1 indicate inliers and scores above 1 indicate outliers. Fig. 12 illustrates how LOF attributes these scores according to local density. For a detailed description of how these scores are computed, see [26, 27].

In *Vesper*, we use euclidean distance as our LOF distance measure. Therefore, we normalize our data with z-score normalization before applying LOF. In order to determine whether or not the observation $\vec{x}$ is an anomaly, we set a cut-off probability p_{thr}, and raise an alert if $LOF(x) > p_{thr}$.

4.5.2. The Anomaly Detection Procedure

To detect anomalies, the HP component maintains a set of observations (profile) for each $j \in \Gamma$ (denoted LOF_j). The task of LOF_j is to (1) learn the normal behavior of the system S_{ij} via each probe $\vec{v}$ taken from host j, and (2) raise an alert if a sample $\vec{v}$ is abnormal. The HP accomplishes this in a plug-and-play fashion updating LOF_j with the first several probes observed. This assumes that there is no MitM in the network during

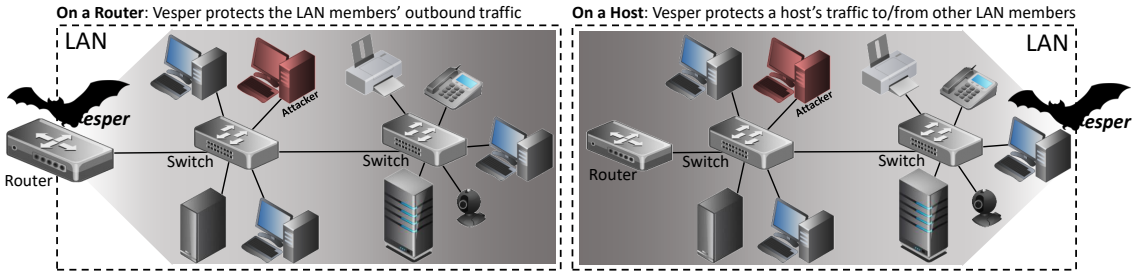


Figure 13: Two ways to deploy *Vesper* in a LAN.

the initial deployment. In summary, the HP performs the following steps when instance $\vec{v}$ arrives:

The Procedure of the HP Component

1. The set LOF_j is retrieved via a hashmap, with host j as the key.
2. $LOF_j(\vec{v})$ is executed propagated through A_j to produce the anomaly score s.
3. **`if`** $s > \phi$, **`and`** the grace period is over, an *alert* is raised.
4. **`else if`** we are in training, then update LOF_j with $\vec{v}$.

In the tool, an averaging window is applied to the alerts (0:normal 1:alert) to increase robustness. The size of the window is configurable: larger windows are more robust but increase the detection delay.

4.6. Deployments

There are two ways one can deploy *Vesper* on a LAN (see Fig. 13). In order to protect the link between host i and j *Vesper* only needs to be running on host i. However, this trust is one-sided since j would be unaware of the state of his link with i. Therefore, to secure all links in a LAN in a fully trusted manner, all hosts in the LAN must be running an instance of *Vesper*. This kind of deployment can be practical in large LANs if *Vesper* (1) sends probes to all LAN end-hosts as usual during an initial training phase (e.g., one day), and then (2) have the OR only send probes to IP addresses with whom the local host is *currently* communicating with.

Another option is to install one instance of *Vesper* on the network gateway (router). Although this does not secure the links between each host of the LAN, it does secure the inbound and outbound traffic. We note that both deployments only protect a host from MitM attacks originating within the same LAN. In the future, we plan extending *Vesper* to work across routers as well.

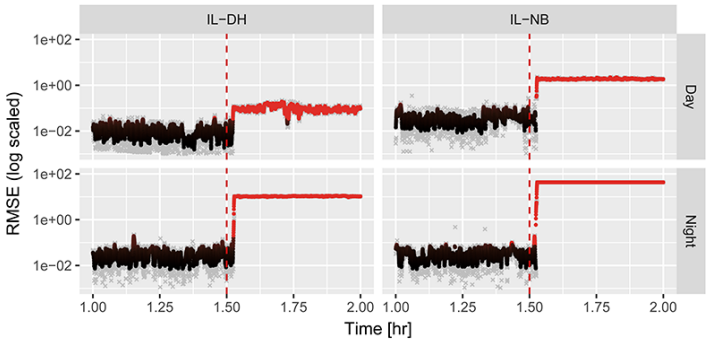


Figure 14: *Vesper's* performance in detecting IL MitMs in a large LAN over multiple intermediary switches. The scores before applying 10 sec averaging window are marked in gray.

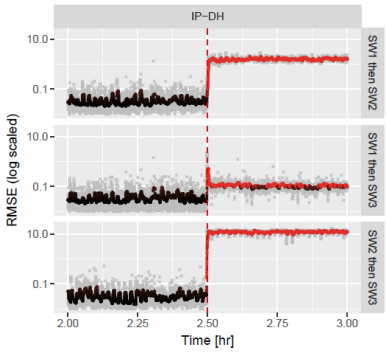


Figure 15: *Vesper* detecting IP-DH MitM attacks. The scores before applying a 1 min averaging window are marked in gray.

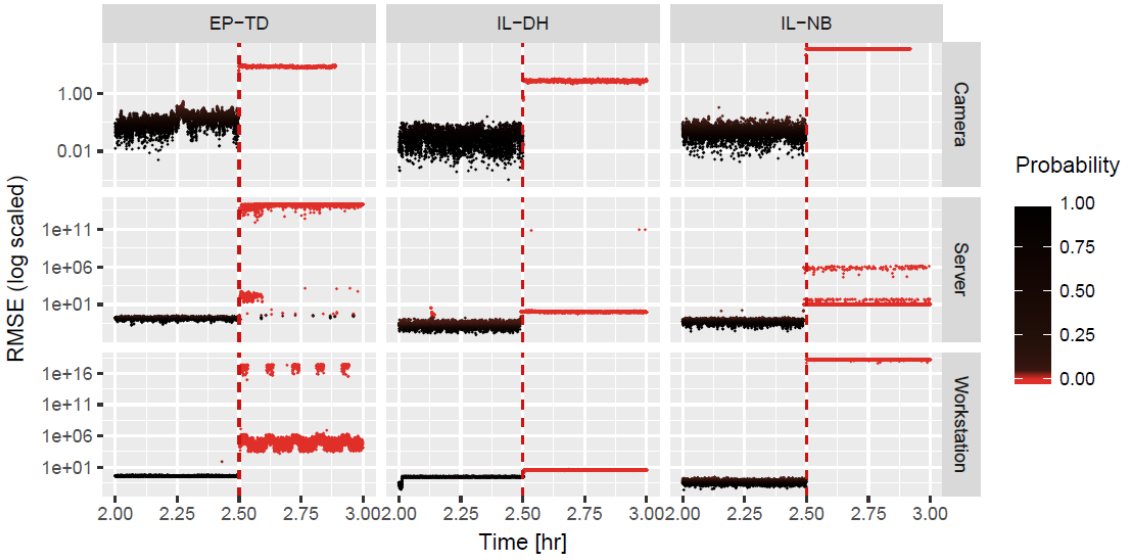


Figure 16: Plots of the anomaly scores produced by *Vesper* when installed on a laptop PC (*Alice*), and in the presence of one intermediary switch. The rows indicate the victim (*Bob*), and the columns indicate the MitM attack (*Eve*).

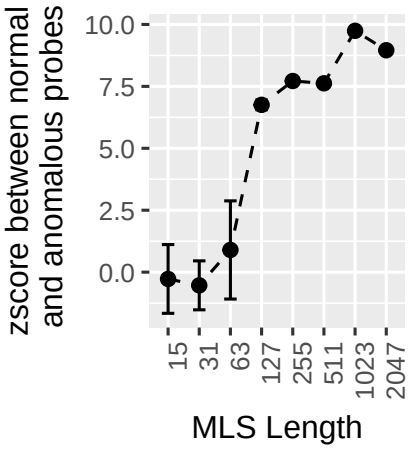


Figure 17: How MLS signal size affects the detection of an IL MitM.

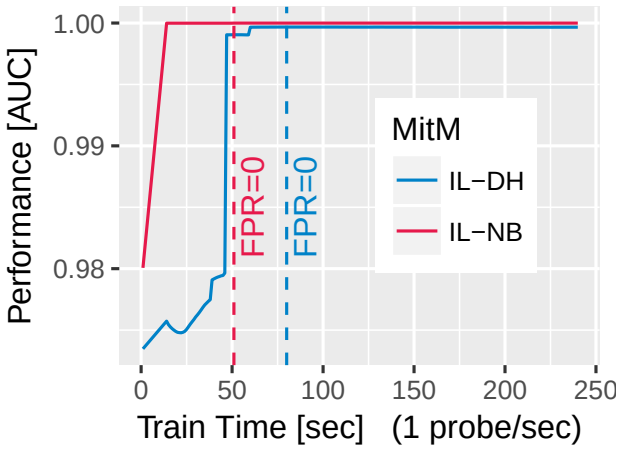


Figure 18: The performance while retraining a model over multiple switches during the day.

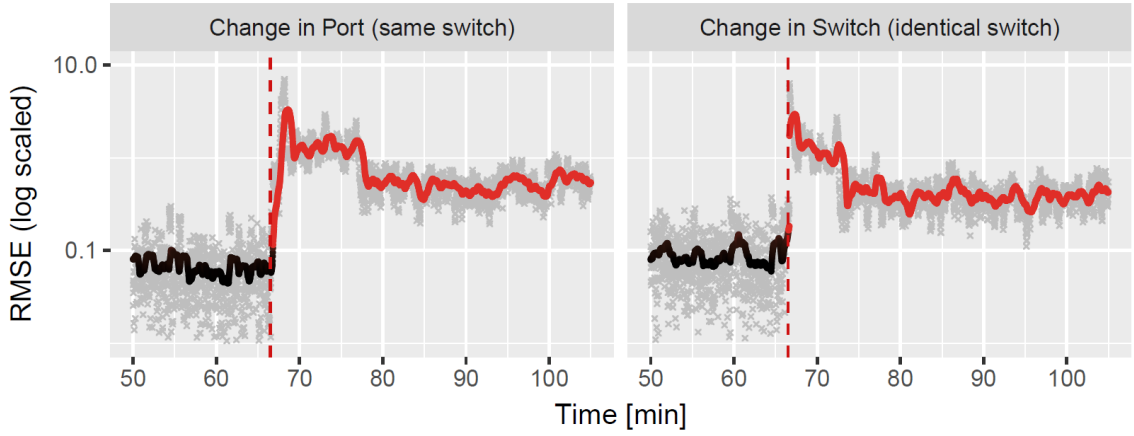


Figure 19: *Vesper* detecting IP-DH attacks using an *identical* switch. Left: when a port is changed on the same switch. Right: when an *identical* switch replaces the current switch.

5. Evaluation

In this section, we refer to the device on which *Vesper* is installed as *Alice*, and *Bob* as the device whose channel with *Alice* is under attack.

We evaluated *Vesper* in detecting End-Point Traffic Diversion (EP-TD), In-Line Network Bridge (IL-NB), In-Line Dedicated Hardware (IL-DH) and In-Point Dedicated Hardware (IP-DH) MitM attacks (see Table 1b). For the EP-TD MitM attack, we used a Kali Linux Desktop PC which performed an ARP poising attack. For the IL-NB and IL-DH MitM attacks, we used a Raspberry Pi 3B and a 1Gbps Ethernet switch respectively. The Pi was provided with an extra Ethernet adapter, and configured to operate as a network bridge. For the IP-DH MitM attacks, we used three 1Gbps switches: an advanced feature (SW1), basic (SW2), and PoE (SW3) switch.

In this section, we refer to each of the devices in the above setups as the attacker *Eve*. In our experiments, we used a C++ implementation of *Vesper*, set the autoencoder learning rate to $l = 0.1$, the *KS-Tester* parameters to $m = 5$ and $k = 15$, and the MLS probe length to $N = 1023$. The T_{tx} and T_{rx} times were obtained with ± 83 ns accuracy.[4]

In order to evaluate *Vesper*'s accuracy when operating in different sized LANs, we examined two setups: (1) when *Alice* connects to the same switch as *Bob* (one intermediary switch), and (2) when *Alice* connects several switches away from *Bob* (multiple intermediary switches). In both setups, *Vesper* was evaluated in the presence of a wide variety of real-world traffic, while the end-hosts were actively using the network. We will now present our experimental results accordingly (a summary of the experiments and results can be found in Table 1).

5.1. One Intermediary Switch

For the EP[5] and IL[6] MitMs, we experimented on two LANs (1) a surveillance camera network, and (2) a LAN segment populated with active servers. The surveillance network consisted 8 high-end HD Sony cameras and three PCs. The server LAN segment consisted of one large switch connected to 61 active servers. *Alice* was a Kali Linux laptop PC (Intel i5 CPU), and *Bob* was either a camera (SNC-EB602R), a Windows desktop PC workstation (Intel i7 CPU), or a data server (Intel Xeon E5-2660 CPU) in each experiment. The desktop PC was located in the surveillance network.

We performed the EP-TD,[7] IL-NB,[8] and IL-DH[9] attacks on each of the three versions of *Bob*, with duration of 3 hours each. Fig. 16 shows RMSE scores produced by *Vesper* in each of experiments (at the moment of the attack). Each point in the figure represents the result of a single probe, and the color indicates the probe's abnormality.

For the IP-DH[10] attack, we trained *Vesper* on one switch, and then swapped the switch with a different one. Fig. 15 presents the results from the IP-DH attacks.

[4]For the purpose of reproducibility, we have published a ping timing tool based on the experiment code used to test Vesper in this talk `https://github.com/ymirsky/burstPing`

[5]End-Point: An attack originating from an end-host

[6]In-Line: An attack originating from a device placed mid-wire

[7]End-Point Traffic Diversion: When traffic is diverted to an end-host first

[8]In-Line Network Bridge: The installation of network bridge mid-wire

[9]In-Line Dedicated Hardware: A dedicated MitM device installed mid-wire

[10]In-Point Dedicated Hardware: Replacing a switch with an infected one

Table 1: A summary of this talk's experiments and results with regards to *Vesper*'s MitM detection performance.

	Figure:	**13**									**14**			**15**				**18**		**19**		**21**
	Description of Experiment:	*Attacks over one intermediary switch in different network setups: surveillance network, small office, and large server network*									*Attacks which swap an existing switch with a different switch*			*Attacks over multiple intermediary switches during daytime and nighttime traffic*				*Attacks which involve the same or identical switch*		*Attacks which occur after one full week of runtime*		*Attacks with adversarial bypass evasion*
MitM Attack	*Topology*	EP	IL	IL	EP	IL	IL	EP	IL	IL	IP	IP	IP	IL	IL	IL	IL	IP	IP	IL	IL	IL
	Implementation	TD	DH	NB	TD	DH	NB	TD	DH	NB	DH	DH	DH	NB	DH	NB	DH	DH	DH	NB	DH	NB
	Alice (Vesper)	Dell Laptop			Dell Laptop			Dell Laptop			Dell PC			Dell PC (Lab Workstation)				Dell PC		Dell PC		Laptop
	Bob	SNC-EB602R Camera			Dell PC			Intel Xeon E5 Server			Dell Laptop			Dell PC (Secretary)				SNC-EM602RC		SNC-EM602RC		PC
	Eve	Dell PC	Rasp. PI 3B	1Gbps Switch	Dell PC	Rasp. PI 3B	1Gbps Switch	Dell PC	Rasp. PI 3B	1Gbps Switch	Basic Switch	PoE Switch	PoE Switch	Rasp. PI 3B	1Gbps Switch	Rasp. PI 3B	1Gbps Switch	Different port	Identical switch	Rasp. PI 3B	1Gbps Switch	1Gbps Switch
	Attack Vector	Install Malw.	Add Device	Add Device	Install Malw.	Add Device	Add Device	Install Malw.	Add Device	Add Device	Replace Advan.	Replace Advan.	Replace Basic	Add Device	Add Device	Add Device	Add Device	Topology Change	Replace Switch	Add Device	Add Device	Add Device
Environment	*Description*	Surveillance Network			Surveillance Network			Server Network			Basic LAN			Large Office Network				Basic LAN		Large Of. Network		Surv. Network
	Num. Switches in the LAN	7	7	7	7	7	7	3	3	3	1	1	1	14	14	14	14	1	1	14	14	7
	Num. Intermediary Switches	1	1	1	1	1	1	1	1	1	1	1	1	4+	4+	4+	4+	1	1	2	2	4+
	Num. of hosts in the LAN	13	13	13	13	13	13	61	61	61	2	2	2	379	379	379	379	2	2	379	379	13
	Time of Day	Day	Day	Day	Day	Day	Day	Day	Day	Day	Day	Day	Day	Day	Day	Night	Night	Day	Day	Day	Day	Day
Experiment	*Total Probes Sent*	15,000	15,000	15,000	15,000	15,000	15,000	15,000	15,000	15,000	10,000	10,000	10,000	15,000	15,000	15,000	15,000	15,000	15,000	610,326	610,326	15,000
	Approximate Duration [hr]	4	4	4	4	4	4	4	4	4	3	3	3	4	4	4	4	4	4	168	168	4
***Vesper*'s Detection Performance**	*Area under the Curve (AUC)*	1	0.9998	1	1	0.9997	1	0.9957	0.9952	0.9999	0.9999	0.9973	0.9999	0.9997	0.9978	0.9908	0.9992	1	1	1	1	0.9989
	Equal Error Rate (EER)	0	0.0001	0	0	0.0063	0	0.0088	0.0096	0.0008	0.0002	0.0170	0.0002	0.0004	0.0070	0.0089	0.0016	0	0	0	0	0.0203
	Recall (TPR)	1	0.9997	1	1	0.9931	1	0.9856	0.9984	0.9986	1	0.6536	1	1	0.9890	0.9935	1	1	1	1	1	0.8667
	False Alarm Rate (FPR)*	0	0.0001	0	0	0	0	0	0.0096	0	0.0002	0.0028	0.0002	0.0004	0.0040	0.0089	0.0016	0	0	0	0	0
	Accuracy	1	0.9998	1	1	0.9965	1	0.9928	0.9945	0.9993	0.9999	0.8250	0.9999	0.9998	0.9926	0.9922	0.9992	1	1	1	1	0.9327
	Precision	1	0.9999	1	1	1	1	1	0.9908	1	0.9998	0.9957	0.9998	0.9996	0.9956	0.9904	0.9982	1	1	1	1	1
	Detection Delay [sec]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	30	0	0	8

**By using an averaging window of 60 seconds, all false positives in this table are mitigated. However, doing do incurs and additional delay in the detection time.*

In all experiments, the IL-NB[8] was the easiest MitM attack to detect. This is because the Pi must perform additional logic in the kernel in order to bridge each ICMP packet. In contrast, the IL-DH[9] and IP-DH[10] were the most difficult to detect, because the packet interception was performed by dedicated hardware. We also note that there were 15 false positives in the experiment with the Server and the IL-DH due to a momentary disconnect. However, these FPs can be easily be mitigated by using an averaging window over the scores. This is because the mean of the scores' distribution significantly changes as seen in Fig. 16. However, the trade-off with using averaging window is that it causes a detection delay. Here, we found that a window size of one minute reduces the number of false positives to zero.

5.2. Multiple Intermediary Switches

To evaluate *Vesper* across multiple intermediary switches, we used an organization's office LAN. The LAN consisted of over 379 network hosts connected through 14 large Ethernet switches, some of which used optical fiber uplinks. The hosts consisted of workstations, servers, printers, and IoT devices. The test scenario involved *Alice* (a desktop PC), and *Bob* (the secretary's PC) which were located on opposite sides of the LAN (separated by four large switches). The probes were sent for three hours, and the attacks (*Eve*) were IL[6] MitMs only.

Fig. 14 shows *Vesper*'s RMSE scores during the day (busy hours) and during the night. The additional traffic during the day caused several false positives when detecting the IL-DH[9]. However, by using an averaging window of 10 seconds, we were able to mitigate the errors completely. The results show that *Vesper* can detect IL MitMs sufficiently well in large noisy LANs, especially during off-hours.

5.3. Long Term Evaluation

In the previous subsections, we examined *Vesper*'s capability and robustness in detecting various MitM attacks over different network topologies. In order to evaluate *Vesper*'s long-term performance, we ran *Vesper* for seven days over a network which was burdened with daily traffic. In this scenario, *Vesper* monitored a surveillance camera over two intermediary switches. The first intermediary switch was a large 24 port switch connected to 17 active office PCs, and up-linked via fiber to a larger LAN (359 network hosts). The second intermediary switch was an eight-port PoE switch by D-Link which was connected to IP surveillance cameras. At the end of seven days, we performed an IL-NB[8] MitM attack and then an IL-DH[9] MitM attack. Fig., 20 shows that the fluctuating network traffic harmed the v_{jit} feature, but overall, did not affect *Vesper*'s MitM detection performance. We also note that v_{RTT*} provided a slightly better detection of the attacks than v_{E_h}. However, an adversary can easily evade the detection of v_{RTT*} by replaying timed packets or by simply performing an EP-TD[7] attack such as ARP poisoning. Furthermore, the impulse response feature v_{E_h} can differentiate between end-point devices and, in some cases, identical devices (see figures **??**, 15, and 19). Therefore, v_{E_h} is a stronger MitM indicator than v_{RTT*}. In conclusion, the long term experiment shows that *Vesper* can be a practical long-term solution in a real-world network.

5.4. Profile Train Time

A concern with *Vesper* is that should a change occur in the LAN's topology, the affected models in the HP component must be retrained. When retrained during busy hours, in the

event of an IL[6] MitM, we found that *Vesper* reaches a false positive rate of zero within seconds (5-15 probes) when applied over one switch, and approximately a minute when applied across multiple switches (the large office LAN). Therefore, although *Vesper* is vulnerable during training, the attacker is challenged with deploying the MitM attack within a narrow time window. To make this window even smaller, *Vesper* can send probes at a faster rate during the grace period. Fig. 18 shows *Vesper*'s performance over time, in the case of multiple switches during day-time traffic. The performance was measured in AUC [28], interpretation: (1.0) *Vesper* was a perfect detector, (0.5) *Vesper* was randomly guessing.

5.5. Probe Length

Fig. 17 shows how the parameter N increases the separation between the normal probes and anomalous probes. Although the use of longer probes improves accuracy, there is a trade-off with bandwidth. *Vesper* sends one probe per second, and a probe has an average of $N(\frac{42+1542}{2})$ bytes. For example, with an $N = 1023$, the bandwidth used is approximately 810 Kbps. This rate is practical, especially since the probe traffic is contained within the LAN, and can be sent between hosts which are *currently* communicating (see section 4.6). However, a user should consider the number of *Vesper* instances installed to appropriately configure N according to his/her limitations.

5.6. Digital Fingerprint

Vesper's probes capture a unique digital fingerprint of the target end-host, and the network devices along the way. Even identical devices have different fingerprints due to imperfections from the manufacturing process (e.g., clock skew [29]). In Fig. 15, we showed that *Vesper* can detect when a switch is swapped with a different one due to the change in this fingerprint. In Fig. 19, we show that *Vesper* can detect the change in fingerprint when a switch is replaced with an identical switch. Interestingly, on many switches, we were also able to detect when a port on the same switch is changed. This indicates that a networking device's fingerprint reflects the control module, and sometimes the electronic interfaces as well.

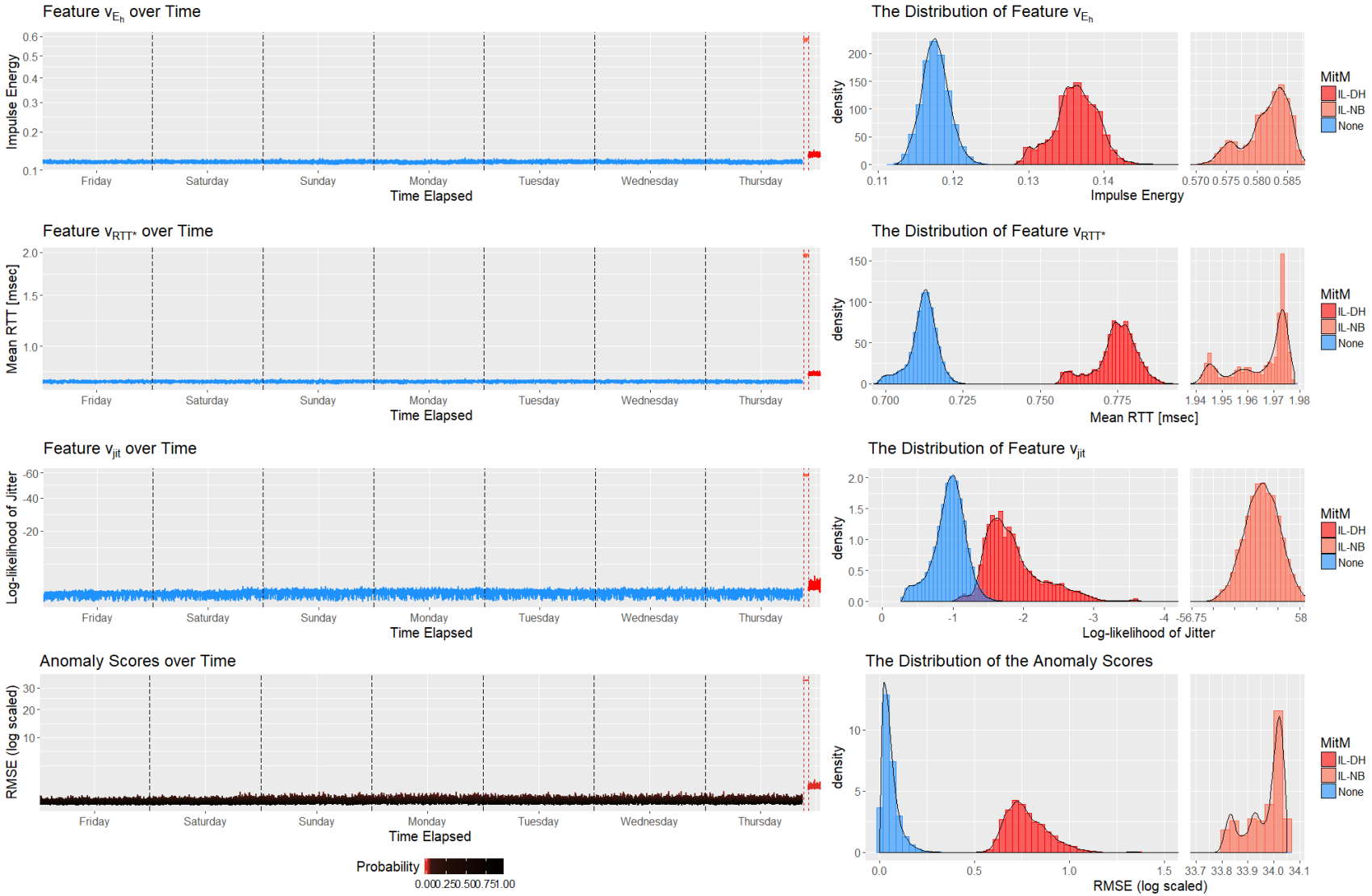


Figure 20: The results of a week-long experiment which concludes with two different MitM attacks. In column 1, the features (rows 1-3) and resulting anomaly scores (row 4) are plotted over time. The first red dashed line indicates the start of an In-Line Network Bridge (IL-NB) MitM attack, and the second red dashed line indicates the start of an In-Line Dedicated Hardware (IL-DH) MitM attack. In column 2, the distributions of the features and their resulting anomaly scores are illustrated.

6. Adversarial Attacks

Our base assumption in this talk is that Eve introduces her MitM attack after *Vesper* has been deployed. However, even with this assumption, Eve can still attempt to evade detection. In this section, we discuss possible tactics which Eve may perform to evade detection. We then discuss and demonstrate how *Vesper* can detect these adversarial attacks.

In general, there are four possible adversarial attacks against *Vesper* (illustrated in Fig. 21): DoS, Spoof, Replay, and Bypass. *Vesper* can detect these evasions through the three features which the FE extracts from each probe. Each feature is strong at detecting some attacks but weak at detecting the others. However, when combined, the three features provide good protection against the attacks. Table 2 maps this relationship. The Table also provides a summary of each feature's strengths and weaknesses. We will now discuss the detection capabilities of each feature with respect to each evasion.

6.1. DoS

In a DoS attack against *Vesper*, Eve drops `Echo_Request` packets *en route* to Bob so that Alice never receives the signal y. All of the features are strong against this attack. This is because there is a large spike in the features' values when a signal is lost.

6.2. Spoof

In a spoof attack against *Vesper*, Eve replies to `Echo_Request` packets *en route* to Bob, on behalf of Bob. In this attack, no additional hops are added to the packet's route to Bob. However, if Eve is topologically close to Alice, then there may be fewer hops altogether.

The features are affected by a spoof attack as follows:

v_{E_h} is modest at detecting a spoof attack in the case of an EP with the same topological distance as Alice↔Bob. This is because there is a possibility that two different impulses will have the same energy. However, when launched from an IL MitM, v_{E_h} is strong because the topological placement of the MitM highly affects v_{E_h}.

v_{rtt*} This feature is modest in detecting a spoof attack by an EP[11] MitM. This is because in the case where the link Alice↔Eve has the same topological distance as Alice↔Bob (e.g., same number of hops), v_{rtt*} may not be affected. However, if Alice↔Eve has a greater topological distance, then v_{rtt*} may increase. Finally, if

[11] End-Point: Such as the use of ARP poisoning

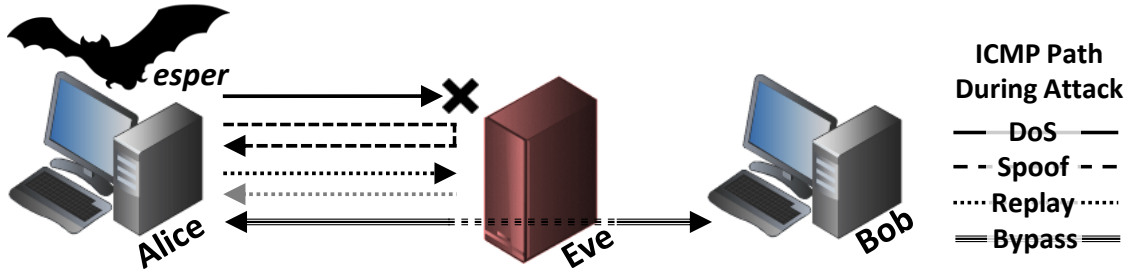


Figure 21: The path of the ICMP packets sent from *Vesper* during each of the adversarial attacks.

Table 2: The capabilities of each feature under each evasive attack, and a summary of the features' strengths and weaknesses.

Feature	Adversarial Attack											
	DoS			*Spoof*			*Replay*			*Bypass*		
	EP	IL	IP	EP	IL	IP	EP	IL	IP	EP	IL	IP
v_{E_h}	●	●	●	○	●	●	●	●	●	-	○	○
v_{rtt*}	●	●	●	○	◌	◌	○	○	◌	-	○	◌
v_{jit}	●	●	●	●	●	●	◌	◌	◌	-	○	○

EP: End-Point MitM
IL: In-Line MitM
IP: In-Point MitM

Detection
◌ *Weak*
○ *Modest*
● *Strong*

Feature	Strengths	Weaknesses
v_{E_h}	Detecting Replay Attacks	Has 1D Collision Space
v_{rtt*}	Detecting Additional Hops	Detecting Spoof Attacks
v_{jit}	Detecting Spoofing Attacks	Detecting Replay Attacks

Alice↔Eve has a smaller distance, then Eve may be advanced enough to delay the response so that the average timing fits the origination distance. Since the relationship is asymmetrical, if both Alice and Bob have an instance of *Vesper*, then Eve will be detected by one of them. With regards to the case of an IL[12] MitM, v_{rtt*} is weak. This is because Eve cannot have a topological distance which greater than Alice↔Bob.

v_{jit} is strong in detecting a spoof attack. This is because the distribution of the probe's jitter captures of the end-host's processing behaviors –which is hard to spoof.

We note that it is difficult to detect a spoof attack in the case where an EP MitM is used, and Eve has the same topological distance as Alice↔Bob. However, even if Eve's MitM device has the exact same hardware, firmware, and software as Bob's device, (1) Eve will only be able to perform a MitM attack on the link with Bob, but not Carol (who has a different device), and (2) there are minute differences in Bob's hardware which truly give Bob's device a unique fingerprint, thus making it difficult for Alice to spoof a reply from Bob.

As mentioned in section 5.6, identical switches and end-devices have unique 'fingerprints' from imperfections in their manufacturing processes [29, 30]. *Vesper* captures the fingerprint of Bob in y. To demonstrate the attacker's challenge in building a MitM device which captures Bob's fingerprint, we had many different devices perform a spoof attack against each other. The experiment was setup in the following way:

1. A set of network hosts Γ from the same LAN are selected, and *Vesper* is installed on a separate host (Alice).
2. *Vesper* (Alice) is trained to protect the link between the local host and host $i \in \Gamma$ (Bob).
3. After 2000 probes, host $j \in \Gamma$ (Eve) replies to *Vesper* instead of host i.
4. Vesper's detection performance id measured after 2000 more probes.
5. Steps 2-4 are repeated for every combination of $i, j \in \Gamma$. We denote the pair (i, j) as spoof attack trial.

The above experiment was performed on two different LANs:

Γ_{as} An assortment of 100 different networked computers and IoTs in a office LAN over multiple switches. This set of network hosts was used to see if a diverse set of

[12] In-Line: Such as an added network bridge

devices have different signatures. In other words, if Eve uses any random device to perform her MitM attack, what would be the likelihood of her spoof evasion to succeed?

Γ_{pi} A set of 46 identical Raspberry Pi 3B devices all connected to a single switch. This set of devices were used in order to determine how likely Eve will succeed at her spoof attack if she replicates Bob's device (and places it at the same topological distance as Alice$\leftrightarrow$Bob). This experiment also demonstrates how unique the fingerprints are between a large set of identical devices.

The results for the experiment on Γ_{as} and Γ_{pi} are summarized in figures 23 and 24. The performance is measured in AUC, where the value has the following interpretation: (1.0) *Vesper* was a perfect detector, (0.5) *Vesper* couldn't differentiate between hosts i and j, and (0.0) *Vesper* thought that host j was host i. The results show that *Vesper* is robust against spoof evasion attacks in all cases. The results on Γ_{as} show us that there is an extremely low likelihood that the attacker's MitM device will be able to spoof the victim's device (or successfully perform an IL[13] MitM attack if she uses just any hardware. The results on Γ_{pi} show that *Vesper* can distinguish between 70% of the identical devices after 10 minutes. Therefore, even if the attacker manages to use the exact same hardware/software as the victim, *Vesper* is still likely to detect the change. Furthermore, Eve will only be able to spoof devices which she has copied (e.g., other Rasp. Pis).

6.3. Replay

In a replay attack against *Vesper*, Eve replays a previously intercepted response signal y back to Alice, whenever a probe x is intercepted on its way to Bob.

The features are affected by a replay attack as follows:

v_{E_h} is strong in detecting all replay attacks. This is because the feature is dependent on the MLS signal, and the MLS sequence is difficult to predict in real-time.[14] We also note that the duration of a signal x is not a constant and can be very noisy due to Alice's host's scheduler. This strengthens the detection capabilities of v_{E_h} in the case of a replay attack because the noise adds a non-deterministic skew to the tx times –which Eve cannot predetermine.

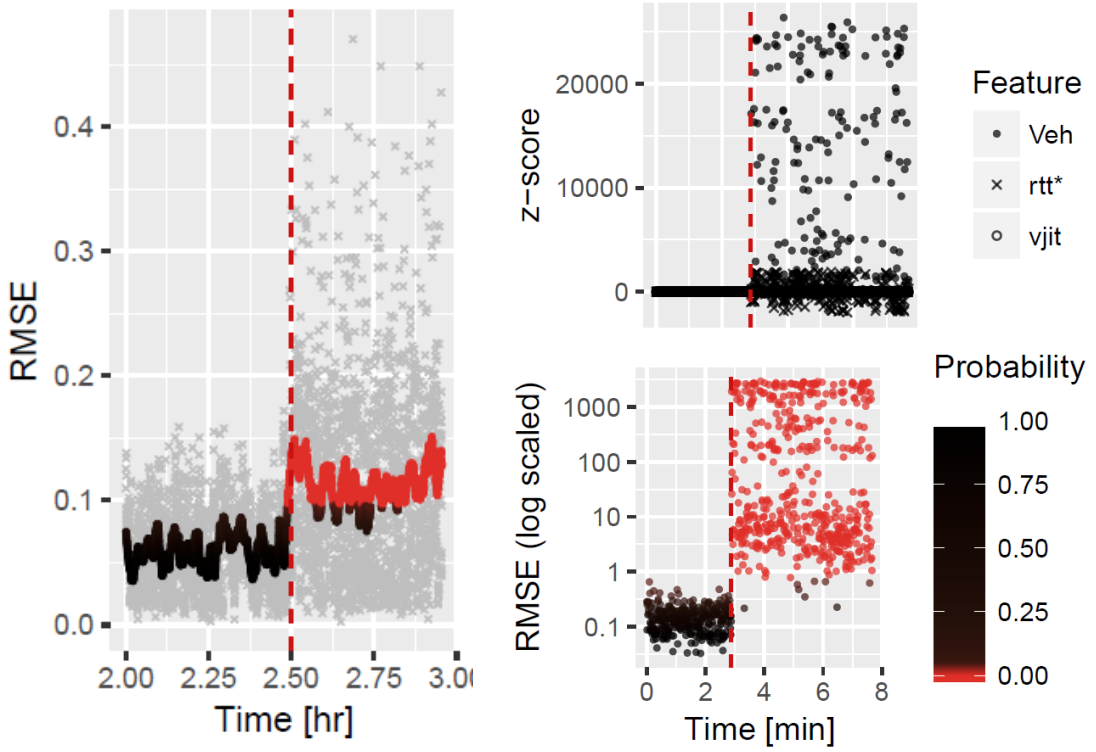
v_{rtt*} This feature is also dependent on the MLS signal. This is because the average RTT is computed on the packets with a size of 1546 only (and not 46). Since the replayed signal will have a different modulation, v_{rtt*} will be lower than it should. However, if Alice uses dedicated networking hardware, she may unintentionally mitigate the noise skew added to the tx times. Therefore, we consider v_{rtt*} modest at detecting replay attacks.

v_{jit} This feature is very weak in detecting replay attacks. This is because v_{jit} is not dependent on the uniqueness of the MLS signal, and thus can be easily copied.

In Fig. 22b, we present the affect which a replay attack has on each of the features (top), and *Vesper's* final anomaly score (bottom). The figure shows that although v_{jit} fails at detecting the attack and that v_{rtt*} is modestly affected, the final score clearly detects Eve's replay evasion due to the combination of v_{rtt*} and v_{E_h}.

[13] In-Line: An attack originating from a device placed mid-wire

[14] This is true if each subsequent MLS seed is determined by a secure pseudo random number generator, such as AES-256 in CTR-mode.



(a) *Vesper*'s detection of a IL-NB MitM using bypass evasion.

(b) *Vesper*'s detection of a replay attack. Top: the features in $\vec{v}$. Bottom: The autoencoder's anomaly scores.

Figure 22: Demonstrations of *Vesper*'s ability to detect bypass and replay attacks.

6.4. Bypass

In a bypass attack against *Vesper*, an advanced attacker uses a special IL[13] device which can choose to either (1) interact with the network acting as a MitM (active-mode), or (2) passively observe traffic acting as a wiretap (passive-mode). To evade detection, Eve is either (A) always in active-mode and switches to passive-mode when an ICMP request is received, OR (B) always in passive-mode and switches to active-mode only when Eve wants to manipulate or inject traffic. *Vesper* can only detect Eve while she is in active-mode.

Vesper can detect Eve if she uses (A). This is because, by the time the first ICMP packet in x is detected by Eve, the frame has already been partially buffered. Therefore, Eve must pass $x[1]$ through her regular interception process before switching over to passive-mode (see Fig. 22a for results). Furthermore, if Eve uses (B), then it is likely that *Vesper* will detect her. This is because Eve must remain in active-mode for long durations in order to be effective. For example, to manipulate streaming/live data, maintain a compromised TCP connection, or to intercept a choice packet.

Another attack may be to learn h in passive-mode, and then apply h to observed probed in active-mode in a spoof attack. However, it is not likely that Eve will be able to learn h from her topological location in the network. This is because, Eve observes the ICMP request packets after they have traversed some portion of the network (switches, cable, ...), and ICMP reply packets before they traverse the other portion of the network and

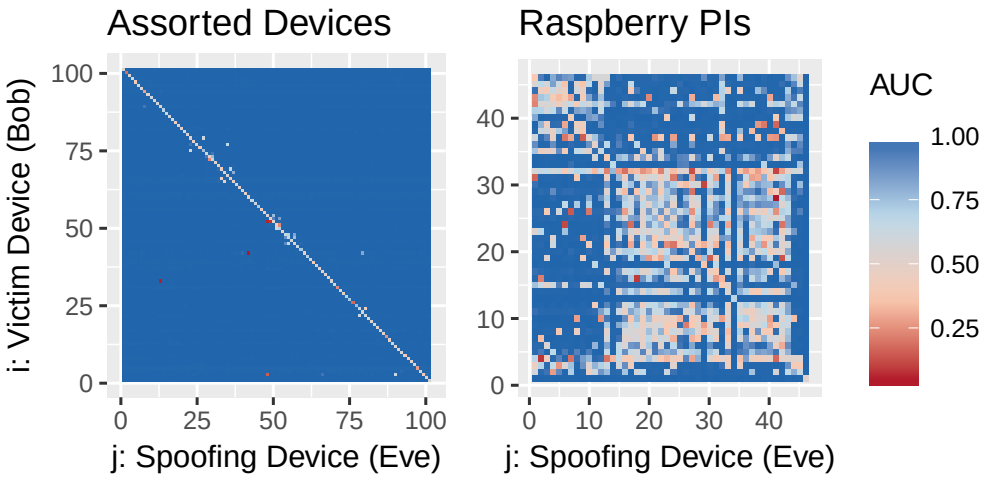


Figure 23: *Vesper*'s AUC in detecting $12,116$ different spoof attack trials with an End-Point MitM topology, when Eve uses an arbitrary device Γ_{as} (left), or a similar device Γ_{pi} (right).

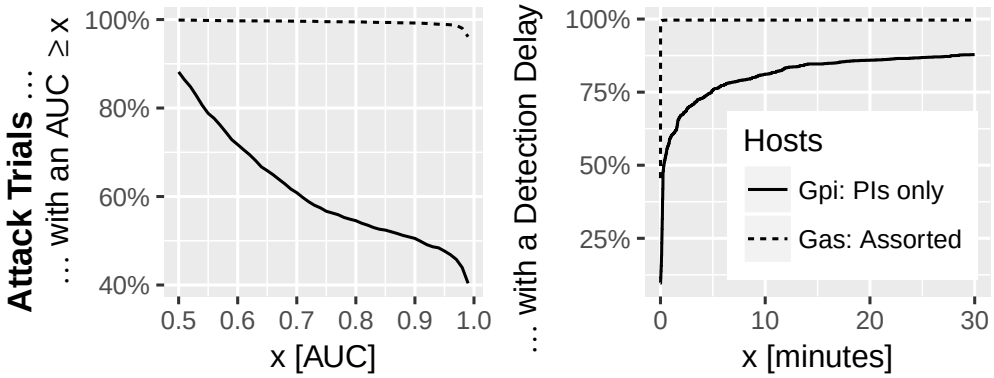


Figure 24: The AUC and detection delays of the spoof attack experiments in terms of percentiles.

Alice's networking stack. Furthermore, Eve will likely leave her fingerprint on the probe when she spoofs it back to Alice (see Fig. 23).

7. The Software Tool

You can download the *Vesper* Tool from `https://github.com/ymirsky/Vesper`. The current version (v1.0) implements most of the features described in this talk. The main features which are currently missing are (1) alerts when *Vesper* is being targeted in an adversarial attack, and (2) fine tunning so that *Vesper* can detect the difference between identical devices. In later distributions, we will be adding these missing features.

7.1. Implementation Notes

Here are some notes on the tool's implementation:

- This is a python implementation of Vesper which wraps C/C++ code using cython. The C/C++ code is used to perform the ICMP probing quickly and accurately. See Fig. 25 for a visualization.
- This implementation uses local outlier factor (LOF) for anomaly detection (Black-Hat'19) and not autoencoders (NDSS'18).
- The current version of vesper has been tuned to detect all of these cases except IP-DH where the exact same model is being used (i.e., the tool can detect the difference between two different 1Gps switches, but not identical ones). The tuned version will be released at a later date.
- This tool currently does not currently implement detection of attacks on Vesper itself.
- This version of the tool will send probes at a constant rate during training and execution. Although you can manually change the rate over time, the probes will constantly be sent (and not just during communications).
- The source code has been tested with Python 2.7.12 on a Linux 64bit machine (Kali Linux). To port the tool to Windows, some C++ libraries must be changed.

7.2. Installation

To install the tool, first consider that *Vesper* will only protect the link from the localhost to other hosts in your LAN (see section 4.6). Later, you will be able to configure which hosts the localhost will monitor.

Installation and Runtime Requirements:

- Linux (tested on Kali)
- Python 2 & some modules on the Internet
- Root privileges (to open raw sockets)

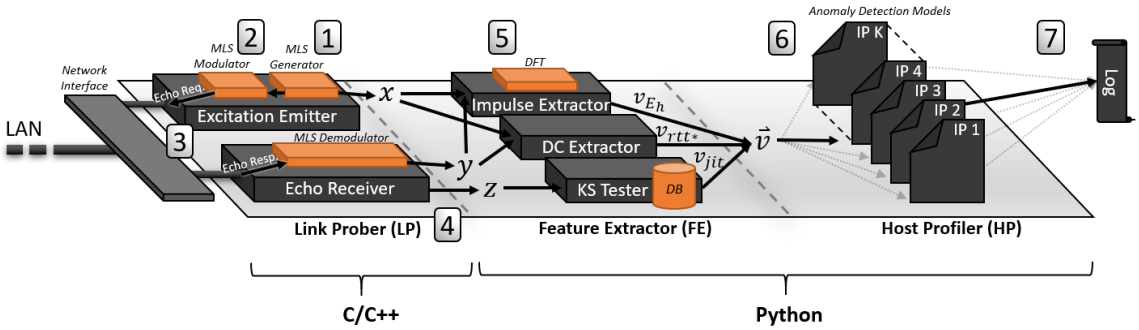


Figure 25: The breakdown of the tool's implementation according to programming language.

First install Python 2 (2..7.12 preferred). Then install the python dependencies: We use cython to wrap C/C++ code and execute it from Python. We also use the prettytable module to display reports to the console.

To install prettytable and cython, run this command in your terminal:

```
$ pip install prettytable cython
```

Note that there are additional python modules which you may need if you are working with a barebone installation of Python:

```
$ pip install scipy sklearn matplotlib cython
```

You are now ready to run *Vesper*!

7.3. Usage Guide

Since the tool uses raw sockets, you must run vesper with sudo privileges. For example:

```
$ sudo python vesper.py
```

The first time you run vesper.py, cython will compile the necessary C++ libraries. When launched, Vesper will monitor the IPv4 addresses in the local file IPs.csv, unless a target IP address is provided as an argument. A profile is trained for each host and is saved to disk (automatically retrieved each time the tool is started). The configuration of the last run is saved to disk (except the real-time plotting toggle argument). Note, this tool only works when monitoring a link contained within a LAN (switches only). Do not provide external IPs. From our experiments, Vesper should also work across multiple switches, VLANs, and optical uplinks.

For complete instructions on how to use vesper, type into the command line

```
$ python vesper.py -h
```

The following will be the output:

```
usage: vesper.py [-h] [-i [I [I ...]]] [-t T] [-p] [-f F] [-r R] [-w W] [--reset]

optional arguments:
  -h, --help       show this help message and exit

  -i [I [I ...]]   Monitor the given IP address(es) <I> only. If an IP's profile
                   exists on disk, it will be loaded and used.
                   You can also provide the path to a file containing a list of
                   IPs, where each entry is on a separate line.
                   Example: python vesper.py -i 192.168.0.12
                   Example: python vesper.py -i 192.168.0.10 192.168.0.42
                   Example: python vesper.py -i ips.csv

  -t T             set the train size with the given number of probes <T>.
                   If profiles already exist, the training will be shortened or
                   expanded accordingly.
                   Default is 200.
                   Example: python vesper.py -i 192.168.0.12 -t 400

  -p               Plot anomaly scores in real-time.
                   Example: python vesper.py -p

  -f F             load/save profiles from the given directory <F>.
                   If is does not exist, it will be created.
                   Default path is ./vesper_profiles.

  -r R             Sets the wait time <R> between each probe in milliseconds.
                   Default is 0.

  -w W             Sets the sliding window size <W> used to average the anomaly
                   scores. A larger window will provide fewer false alarms, but it
                   will also increase the detection delay.
                   Default is 10.

  --reset          Deletes the current configuration and all IP profiles stored on
                   disk before initializing vesper.
```

Fig. 26 shows the result of running

```
$ python vesper.py -i 142.44.32.101 -w 30 -p
```

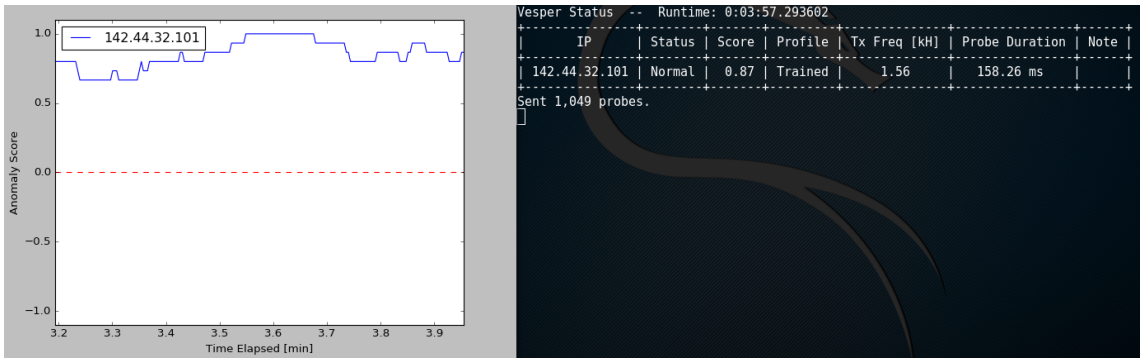


Figure 26: A screenshot of the tool in operation. Negative scores indicate abnormality.

7.4. License

Please visit `https://github.com/ymirsky/Vesper` to read the latest Software License.

8. Conclusion

As of today, MitM attacks still pose a great threat to many LANs. In this talk we have presented a new technique for detecting MitM attacks in LANs via ping echo analysis. We have shown how the technique can be practically applied via a MitM detection framework called *Vesper*. Experimental results show that (1) *Vesper* is capable of detecting end-point, in-line, and in-point MitM attacks, and (2) is robust against possible adversarial attacks. For future work, we plan on applying other ping methods (e.g., TCP SYN), applying noise mitigation techniques, and extending the technique to work over routers, and applying *Vesper* to Wi-Fi networks.

With this talk we have also released a tool which implements most of the functionality of *Vesper*. The tool can be downloaded from `https://github.com/ymirsky/Vesper`

In general, MitM attacks should be prevented by using secure protocols between machines, even in a trusted environment. However, we note that *Vesper* provides an extra line of defense, and can additionally alert the user of an attacker's presence.

References

[1] G. Michael, "Six ways you could become a victim of man-in-the-middle (mitm) attacks this holiday season," HUFFPOST, Tech. Rep., 2016. [Online]. Available: http://www.huffingtonpost.com/michael-gregg/six-ways-you-could-become_b_8545674.html

[2] S. Y. Nam, D. Kim, J. Kim *et al.*, "Enhanced arp: preventing arp poisoning-based man-in-the-middle attacks," *IEEE communications letters*, vol. 14, no. 2, pp. 187–189, 2010.

[3] G. N. Nayak and S. G. Samaddar, "Different flavours of man-in-the-middle attack, consequences and feasible solutions," in *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, vol. 5. IEEE, 2010, pp. 491–495.

[4] Z. Trabelsi and et al., "Nis04-4: Man in the middle intrusion detection," in *Global Telecommunications Conference, 2006. GLOBECOM'06. IEEE*. IEEE, 2006, pp. 1–6.

[5] S. Gangan, "A review of man-in-the-middle attacks," *arXiv preprint arXiv:1504.02115*, 2015.

[6] S. Son and V. Shmatikov, "The hitchhiker's guide to dns cache poisoning," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2010, pp. 466–483.

[7] J. Stewart, "Dns cache poisoning–the next generation," 2003.

[8] CAPEC, "Capec-94: Man in the middle attack," 2014. [Online]. Available: http://capec.mitre.org/data/definitions/94.html

[9] M. Conti, N. Dragoni, and V. Lesyk, "A survey of man in the middle attacks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2027–2051, 2016.

[10] J. Belenguer and C. T. Calafate, "A low-cost embedded ids to monitor and prevent man-in-the-middle attacks on wired lan environments," in *Emerging Security Information, Systems, and Technologies, 2007. SecureWare 2007. The International Conference on*. IEEE, 2007, pp. 122–127.

[11] Let'sEncrypt, "Percentage of web pages loaded by firefox using https," Internet Security Research Group (ISRG), Tech. Rep., 2017. [Online]. Available: https://letsencrypt.org/stats/

[12] B. Zhu, A. Joseph, and S. Sastry, "A taxonomy of cyber attacks on scada systems," in *Internet of things (iThings/CPSCom), 2011 international conference on and 4th international conference on cyber, physical and social computing*. IEEE, 2011, pp. 380–388.

[13] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue, "A messy state of the union: Taming the composite state machines of tls," in *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015, pp. 535–552.

[14] K. Bhargavan and G. Leurent, "Transcript collision attacks: Breaking authentication in tls, ike, and ssh," in *Network and Distributed System Security Symposium–NDSS 2016*, 2016.

[15] TrendMicro, "How do threat actors move deeper into your network?" Tech. Rep., 2013. [Online]. Available: http://about-threats.trendmicro.com/cloud-content/us/ent-primers/pdf/tlp_lateral_movement.pdf

[16] W. is Arpdefender Designed For?, "arpdefender." [Online]. Available: http://www.arpdefender.com/home

[17] I. Green, "Dns spoofing by the man in the middle," 2005.

[18] D. Bruschi, A. Ornaghi, and E. Rosti, "S-arp: a secure address resolution protocol," in *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*. IEEE, 2003, pp. 66–74.

[19] W. Lootah, W. Enck, and P. McDaniel, "Tarp: Ticket-based address resolution protocol," *Computer Networks*, vol. 51, no. 15, pp. 4322–4337, 2007.

[20] Y. Bhaiji, *Network security technologies and solutions (CCIE professional development series)*. Pearson Education, 2008.

[21] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks." in *Lisa*, vol. 99, no. 1, 1999, pp. 229–238.

[22] D. Havelock, S. Kuwano, and M. Vorländer, *Handbook of Signal Processing in Acoustics*. Springer, 2008, no. v. 1-2. [Online]. Available: https://books.google.co.il/books?id=TDPIO01DLSUC

[23] G.-B. Stan, J. J. Embrechts, and D. Archambeau, "Comparison of different impulse response measurement techniques," vol. 50, pp. 249–262, 04 2002.

[24] J. Postel, "Internet control message protocol specification," STD 5, RFC 792, September, Tech. Rep., 1981.

[25] R. Maes, *Introduction and Preview*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–9. [Online]. Available: https://doi.org/10.1007/978-3-642-41395-7_1

[26] "Local outlier factor - wikipedia," https://en.wikipedia.org/wiki/Local_outlier_factor, (Accessed on 03/13/2019).

[27] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *ACM sigmod record*, vol. 29, no. 2. ACM, 2000, pp. 93–104.

[28] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[29] T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device fingerprinting," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 93–108, 2005.

[30] R. M. Gerdes, T. E. Daniels, M. Mina, and S. Russell, "Device identification via analog signal fingerprinting: A matched filter approach." in *NDSS*, 2006.

9. Additional Figures

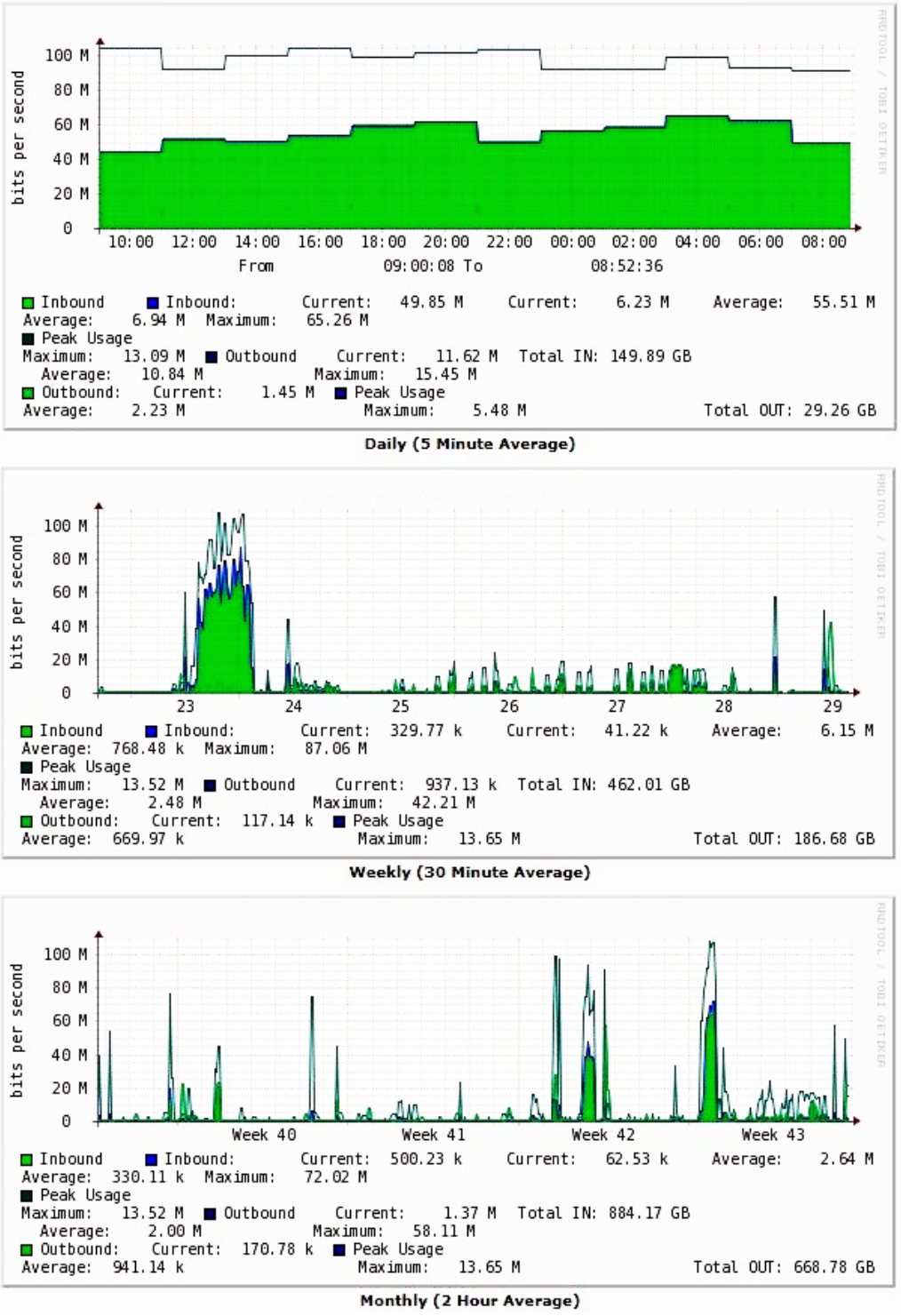


Figure 27: The common hourly, daily, and monthly traffic loads of the 50 port switch used in the detection experiment where the Data Server was the victim (see Fig. 16).

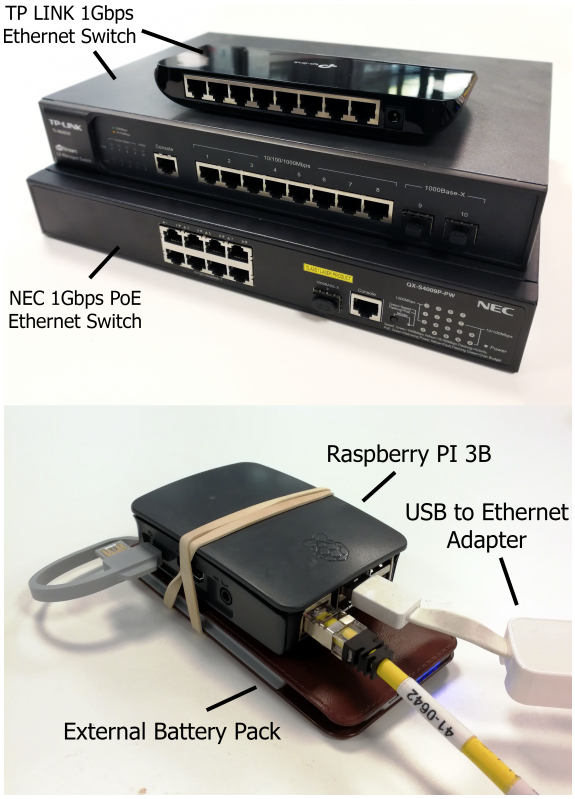


Figure 28: The MitM devices used in this paper. Top: Three 1Gbps Ethernet switches used in the IP-DH experiments, where the middle switch was used in the IL-DH experiments. Bottom: A Raspberry Pi 3B with a battery pack and 1Gbps USB to Ethernet adapter, used in the IL-NB experiments.

Figure 29: Two of the eight Sony IP surveillance cameras used in the experiments. The models were: SNC-EM602RC, SNC-EB600, SNC-EM600, and SNC-EB602R.

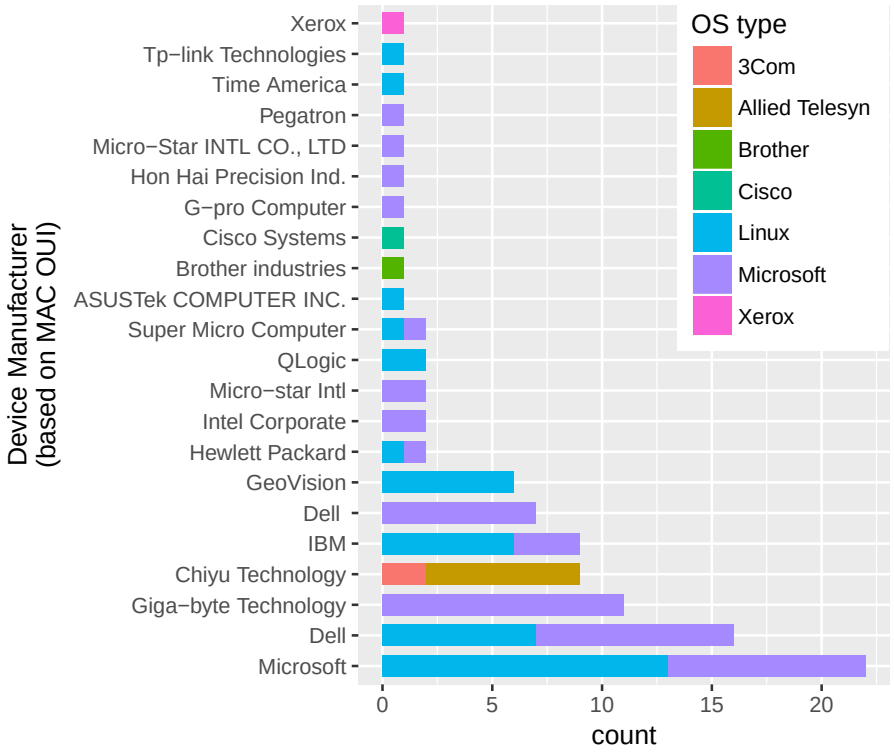


Figure 30: A break-down of the device's operating systems, used in 100 host spoof experiment.

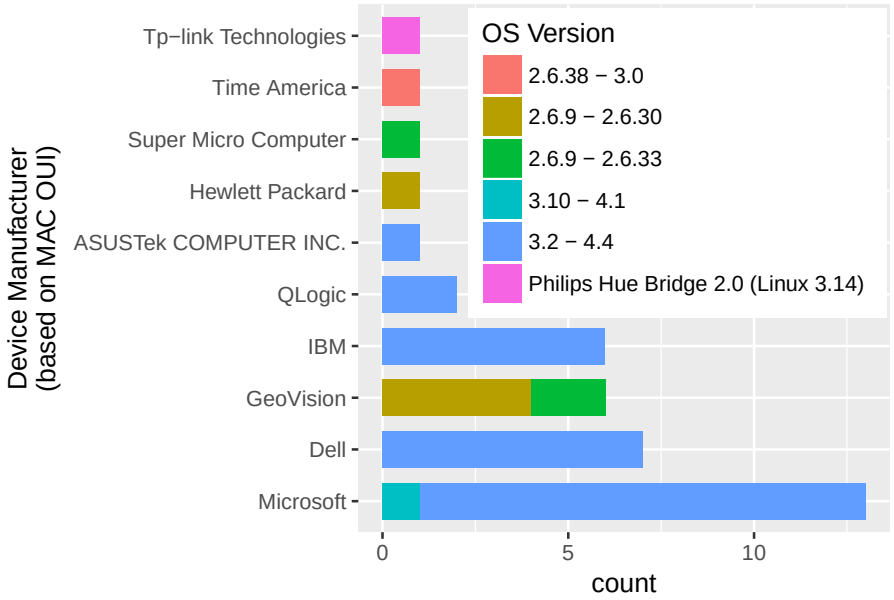


Figure 31: A break-down of the devices running a Linux operating system, used in 100 host spoof experiment.

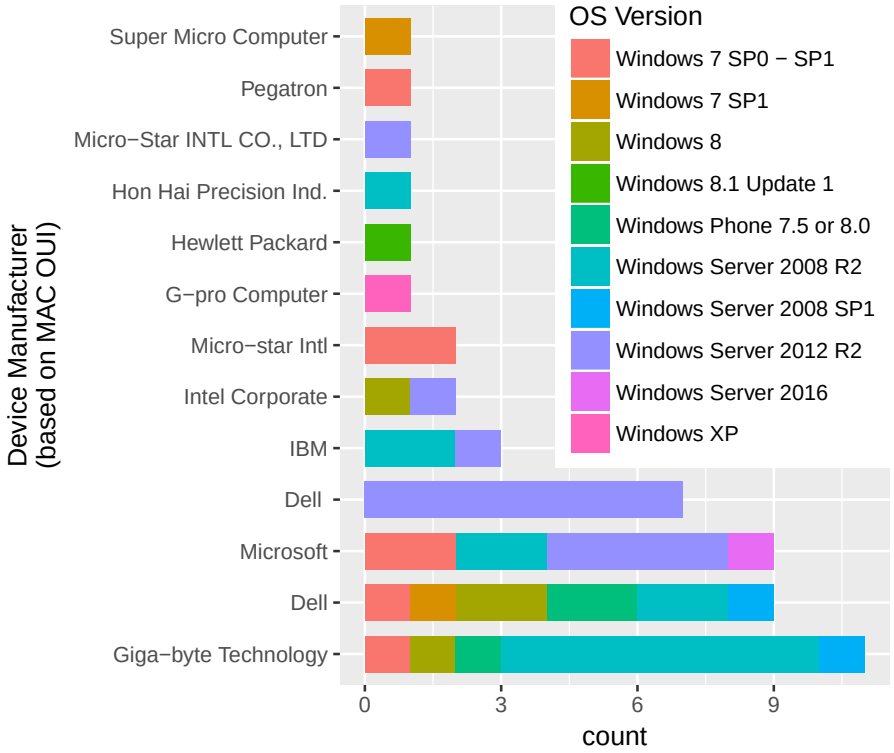


Figure 32: A break-down of the devices running a Microsoft operating system, used in 100 host spoof experiment.

In-Line MitM using *Dedicated Hardware* (IL-DH), and has a bypass to evade detection.

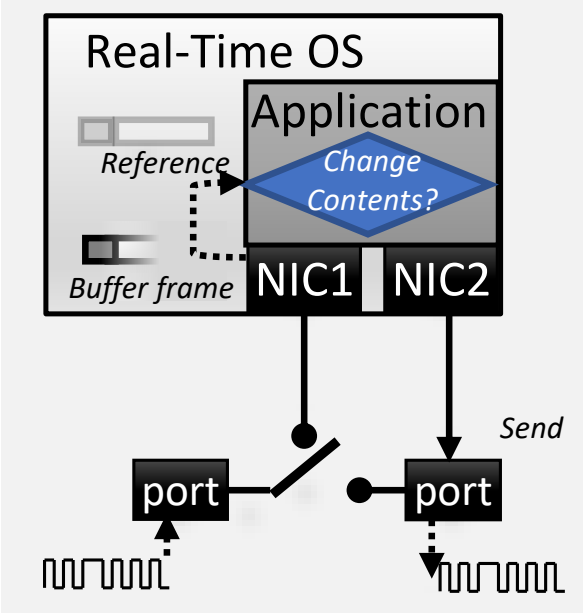


Figure 33: The packet interception process of an IL-DH MitM device, with a bypass evasion mechanism. The device can either actively or passively observe traffic.

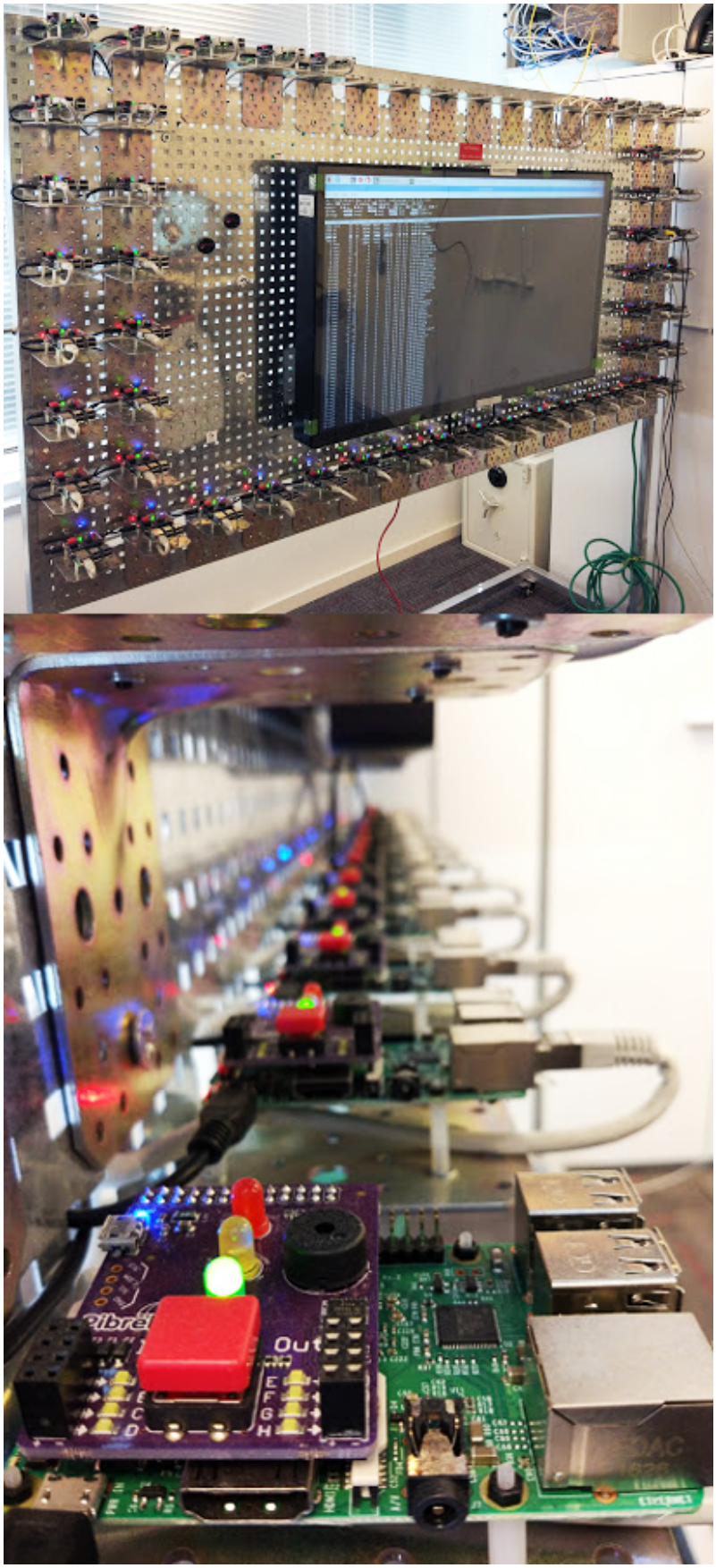


Figure 34: The 46 Raspberry Pi 3Bs used in the experiments. All of the devices were connected to a single Ethernet switch.